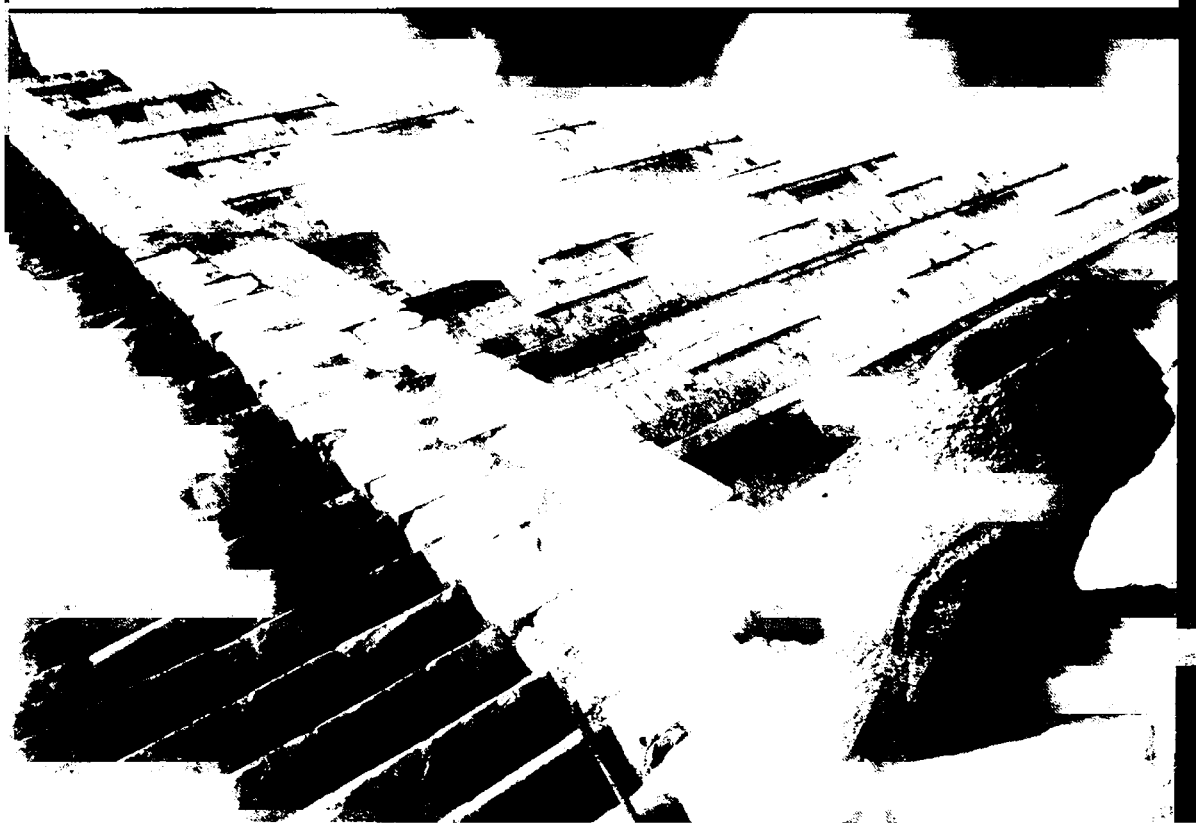
MySQL™

Полное и исчерпывающее руководство по применению и администрированию баз данных MySQL 4, а также программированию приложений

Поль Дюбуа

Второе издание



MySQL

The definitive guide to using, programming, and administering MySQL 4 databases

---

Second Edition

Paul DuBois

DEVELOPER'S LIBRARY

Sams Publishing, 201 West 103rd Street, Indianapolis, Indiana 46290

MySQL

Полное и исчерпывающее руководство по применению и администрированию баз данных MySQL 4, а также программированию приложений

Второе издание

Поль Дюбуа



Москва • Санкт-Петербург • Киев
2004

ББК 32.973.26-018.2.75
Д95
УДК 681.3.07

Издательский дом "Вильямс"

Зав. редакцией *С.Н. Тригуб*

Перевод с английского и редакция *Н.В. Воронина*

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Дюбуа, Поль.

Д95 MySQL, 2-е изд. : Пер. с англ. — М. : Издательский дом "Вильямс", 2004. — 1056 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0515-X (рус.)

В книге описана одна из самых популярных реляционных систем управления базами данных MySQL. Вместо простого обзора возможностей MySQL автор показывает, как можно максимально эффективно использовать эту СУБД. На примере тестовых баз данных он приводит решения проблем, с которыми вы обязательно столкнетесь при практическом использовании MySQL. Вы научитесь интегрировать MySQL с программными средствами, разработанными сторонними компаниями, такими как компиляторы языков PHP и Perl, которые позволяют с помощью запросов к базе данных создавать динамические Web-страницы. Кроме того, здесь приведен обширный справочник по типам столбцов, операторам, функциям, синтаксическим конструкциям языка SQL, интерфейсам C API, Perl DBI и PHP API. В книге предоставлена уникальная информация, которую вряд ли можно получить где-нибудь еще.

Книга рассчитана на пользователей и разработчиков систем клиент/сервер на основе MySQL.

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Sams Publishing.

Authorized translation from the English language edition published by Sams Publishing, Copyright © 2003
All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Russian language edition published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2004

ISBN 5-8459-0515-X (рус.)
ISBN 0-7357-1212-3 (англ.)

© Издательский дом "Вильямс", 2004
© Sams Publishing, 2003

Оглавление

Часть I. Использование СУБД MySQL **33**

Глава 1. Знакомство с СУБД MySQL и SQL 35
Глава 2. Работа с данными в MySQL 113
Глава 3. Синтаксис и использование языка SQL 191
Глава 4. Оптимизация запросов 263

Часть II. Программные интерфейсы СУБД MySQL **293**

Глава 5. Введение в программирование в СУБД MySQL 295
Глава 6. Программный интерфейс MySQL C API 315
Глава 7. Программный интерфейс Perl API 379
Глава 8. Интерфейс API для языка написания сценариев PHP 465
Часть III. Администрирование MySQL 515
Глава 9. Введение в администрирование MySQL 517
Глава 10. Каталог данных MySQL 523
Глава 11. Общее администрирование MySQL 547
Глава 12. Безопасность 625
Глава 13. Резервное копирование и восстановление баз данных 661

Часть IV. Приложения **697**

Приложение А. Получение и инсталляция программного обеспечения 699
Приложение Б. Типы данных 719
Приложение В. Операторы и функции 731
Приложение Г. Синтаксис SQL 783
Приложение Д. Программы MySQL 861
Приложение Е. Программный интерфейс приложений C 929
Приложение Ж. Программный интерфейс DBI языка написания сценариев Perl 965
Приложение З. Программный интерфейс языка PHP 993
Приложение И. Провайдеры услуг Internet 1023
Предметный указатель 1036

Содержание

Об авторе 19
О рецензентах 19
Благодарности 19
Ждем ваших отзывов! 21
Введение 22
Все-таки, почему MySQL? 24
Если вы уже работаете с другой СУБД 26
Инструментарий, поставляемый с MySQL 26
Для кого предназначена эта книга 27
Структура книги 28
Как читать эту книгу 29
Версии программного обеспечения, которые упоминаются в книге 30
Соглашения, используемые в книге 31
Дополнительные источники информации 31

Часть I. Использование СУБД MySQL 33

Глава 1. Знакомство с СУБД MySQL и SQL 35

Назначение СУБД MySQL 36
Пример базы данных 40
"Историческая Лига" 40
Проект "Учет успеваемости" 43
Каким образом пример базы данных можно использовать в конкретном случае 43
Основная терминология баз данных 44
Структурная терминология 44
Терминология языка запросов SQL 46
Терминология архитектуры СУБД MySQL 47
Учебный курс по СУБД MySQL 48
Пример тестовой базы данных 49
Предварительные требования 49
Установка и завершение связи с сервером 51
Ввод запросов 53
Создание базы данных 55
Создание таблиц 56
Дополнение таблиц 72
Выборка информации 75
Удаление и модификация существующих записей 103
Как работать с mysql 105
Упрощение процесса подключения 105
Упрощение процесса создания запросов 107
Изменение приглашения mysql 111
Что дальше? 111

Глава 2. Работа с данными в MySQL 113

Типы данных СУБД MySQL 115
Цифровые данные 115
Строковые (символьные) данные 115
Дата и время (календарные данные) 117
Пустое значение (NULL) 117
Типы столбцов СУБД MySQL 118
Краткий обзор типов столбцов 118
Создание таблиц 120
Цифровые типы столбцов 121
Строковые типы столбцов 130
Календарные типы столбцов 143
Обработка последовательностей 150
Атрибут AUTO_INCREMENT для таблиц ISAM 150
Атрибут AUTO_INCREMENT для таблиц MyISAM 152
Атрибут AUTO_INCREMENT для таблиц HEAP 154
Атрибут AUTO_INCREMENT для таблиц BDB 154
Атрибут AUTO_INCREMENT для таблиц InnoDB 155
Вопросы, которые необходимо учитывать при использовании атрибутов AUTO_INCREMENT 155
Отключение повторного использования значений в последовательности 156
Добавление столбца с порядковым номером в таблицу 156
Перенумерация существующего столбца 157
Генерирование последовательностей без атрибута AUTO_INCREMENT 158
Поддержка кодировок 160
Поддержка кодировок до версии 4.1 161
Поддержка различных кодировок в СУБД MySQL 4.1 и более поздних версиях 162
Выбор типа столбца 165
Данные какого типа будут храниться в столбце? 166
Каков диапазон значений? 169
Как повысить производительность и эффективность 171
Каким образом сравниваются значения 173
Будете ли вы индексировать столбец? 174
Как учесть внутреннюю зависимость при выборе типа столбца 174
Вычисление выражений и преобразование типов 175
Правила написания выражений 175

Глава 3. Синтаксис и использование языка SQL 191

Правила присвоения имен в СУБД MySQL 193
Ссылка на элементы баз данных 193
Чувствительность к регистру в операторах SQL 196
Создание, удаление и выборка баз данных 197
Выбор баз данных 197
Создание баз данных 197
Удаление баз данных 198
Изменение структуры базы данных 198
Создание, удаление, индексирование и изменение структуры таблиц 198
Типы таблиц 199
Создание таблиц 204
Удаление таблиц 211

Индексирование таблиц 212
Изменение структуры таблиц 217
Получение информации о базах данных и таблицах 221
Определение типов, поддерживаемых сервером 222
Проверка существования таблиц или типов таблиц 223
Выборка записей из нескольких таблиц 224
Тривиальное объединение 225
Полное объединение 225
Левое и правое объединения 227
Использование вложенных выборок 231
Выборка из нескольких таблиц с помощью объединения UNION 234
Удаление и обновление нескольких таблиц 238
Выполнение транзакций 240
Для чего нужны транзакции 241
Решение проблемы транзакций без применения механизма транзакций 242
Использование транзакций для обеспечения безопасного выполнения операций 245
Внешние ключи и ссылочная целостность 249
Как обходиться без внешних ключей 254
Полнотекстный поиск типа FULLTEXT 255
Комментарии 260
Возможности, которые не поддерживает СУБД MySQL 261

Глава 4. Оптимизация запросов 263

Индексирование 264
Преимущества индексирования 264
Недостатки индексирования 267
Выбор индекса 268
Оптимизатор MySQL 270
Как работает оптимизатор 271
Подавление оптимизации 274
Выбор типа столбцов и эффективность запросов 276
Эффективная загрузка данных 278
Проблемы планирования и блокировки 282
Уровни блокировки и параллельность 285
Оптимизация для администраторов 287
Кэш запросов 289
Проблемы, связанные с используемой аппаратной платформой 290

Часть II. Программные интерфейсы СУБД MySQL 293

Глава 5. Введение в программирование в СУБД MySQL 295

Написание программ, работающих с СУБД MySQL 295
Интерфейсы API для СУБД MySQL 300
Интерфейс API для языка C 302
Интерфейс Perl DBI API 303
Интерфейс PHP API 305
Выбор API-интерфейса 306
Среда выполнения 307
Производительность 308

Время разработки программ 311
Переносимость 313

Глава 6. Программный интерфейс MySQL C API 315

Общая процедура создания клиентских программ 317
Основные требования к системе 317
Компиляция и редактирование связей клиентской программы 317
Client1 — подключение к серверу 319
Client2 — добавление диагностики ошибок 322
Client3 — передача параметров соединения во время выполнения 327
Доступ к содержимому конфигурационного файла 328
Синтаксический анализ аргументов командной строки 331
Подключение обработки параметров в клиентскую программу СУБД MySQL 340
Обработка запросов 344
Обработка запросов, не возвращающих результатов 345
Обработка запросов, возвращающих результаты 346
Обработка общих запросов 350
Альтернативные методы обработки запросов 352
Сравнение возможностей функций mysql_store_result() и mysql_use_result() 353
Метаданные результирующего набора 355
Client4 — программа интерактивного ввода запросов 360
Создание клиентов с поддержкой протокола SSL 362
Библиотека встроенного сервера 366
Создание приложений со встроенным сервером 367
Создание программы 369
Разное 369
Выполнение вычислений на основе данных, полученных в результирующем наборе 370
Кодирование неоднозначных данных в запросах 371
Обработка изображений 374
Получение информации из таблицы 375
Наиболее часто встречающиеся ошибки при программировании клиентских программ 375

Глава 7. Программный интерфейс Perl API 379

Свойства сценариев языка Perl 380
Основы интерфейса базы данных для Perl 381
Типы данных интерфейса DBI 381
Простой DBI-сценарий 382
Обработка ошибок 386
Обработка запросов, не возвращающих результирующего набора данных 390
Обработка запросов, возвращающих результирующий набор данных 392
Использование кавычек 401
Маркеры и привязка параметров 404
Определение параметров соединения 407
Отладка 411
Метаданные результирующего набора 414
Выполнение транзакций 418
Подготовка интерфейса к работе 420
Создание каталога общества "Историческая Лига" 421
Рассылка напоминаний об истечении срока членства 426

Редактирование записей о членах "Исторической Лиги" 431
Подбор членов "Исторической Лиги" по общности интересов 436
Размещение каталога "Исторической Лиги" на Web-узле 437
Работа с интерфейсом DBI в Web-приложениях 440
Настройка сервера Apache для работы с CGI-сценариями 442
Знакомство с модулем CGI.pm 443
Подключение к серверу MySQL из Web-сценария 449
Броузер базы данных sampdb 452
Броузер проекта "Учет успеваемости" 456
Подбор членов "Исторической Лиги" по общности научных интересов 459

Глава 8. Интерфейс API для языка написания сценариев PHP 465

Основы языка PHP 467
Работа с функциями и включаемыми файлами 473
Web-страница с простейшим запросом 477
Обработка результатов запроса 480
Проверка результатов запроса на значения "NULL" 485
Обработка ошибок 486
Проблема кавычек 488
Работа с PHP 490
Ввод оценок учащихся 490
Викторина "Президенты США" 502
Интерактивное редактирование данных о членах "Исторической Лиги" 507

Часть III. Администрирование MySQL 515

Глава 9. Введение в администрирование MySQL 517

Обзор задач администрирования 518
Общее администрирование 519
Безопасность 521
Восстановление и поддержка баз данных 521

Глава 10. Каталог данных MySQL 523

Размещение каталога данных 524
Структура каталога данных 527
Как обеспечивается доступ к данным сервера MySQL 527
Представление баз данных в файловой системе 529
Представление таблиц баз данных 530
Взаимодействие операторов SQL с табличными файлами 531
Ограничения операционной системы на имена баз данных и таблиц 532
Факторы, влияющие на максимальный размер таблиц 534
Влияние структуры каталога данных на производительность системы 536
Файлы состояния и регистрационные файлы MySQL 538
Перемещение содержимого каталога данных 540
Методы перемещения 541
Прогнозирование эффекта от перемещения 542
Перемещение каталога данных 543
Перемещение баз данных 544
Перемещение таблиц баз данных 545

Перенос табличного пространства InnoDB 546
Перемещение файлов состояния 546

Глава 11. Общее администрирование MySQL 547

Защита новой инсталляции MySQL 548
Как установлены таблицы при запуске 548
Установка паролей для первоначальных учетных записей MySQL 551
Установка паролей для второго сервера 553
Настройка процедур запуска и завершения работы сервера MySQL 554
Запуск сервера MySQL под управлением ОС UNIX 554
Работа сервера MySQL под управлением ОС Windows 559
Определение параметров запуска 561
Завершение работы сервера 563
Когда нельзя подключиться к серверу 563
Управление пользовательскими учетными записями 565
Создание новых пользователей и предоставление привилегий 567
Отмена привилегий и удаление пользователей 576
Замена или обновление утерянных паролей 578
Ведение файлов журналов 578
Общий журнал запросов 581
Журнал медленных запросов 581
Журнал обновлений 582
Двоичный журнал обновления и индексный файл двоичного журнала 583
Журнал диагностических сообщений 584
Обновление файлов регистрации 585
Дополнительные вопросы настройки сервера 591
Как сервер "прослушивает" соединения 591
Активизация или отключение возможности LOCAL для оператора LOAD DATA 592
Проблемы интернационализации и локализации 592
Выбор типов таблиц 596
Настройка табличного пространства InnoDB 597
Настройка сервера 601
Запуск нескольких серверов 607
Общие проблемы одновременной эксплуатации нескольких серверов 607
Инсталляция и настройка нескольких серверов 610
Стратегии определения параметров запуска 611
Использование сценария mysqld_multi для управления несколькими серверами 612
Эксплуатация нескольких серверов в ОС Windows 614
Установка серверов репликации 616
Концепция репликации 616
Установка взаимоотношений репликации главный-подчиненный 618
Обновление MySQL 622

Глава 12. Безопасность 625

Внутренняя безопасность: защита доступа к каталогу данных 626
Как похитить данные 627
Защита инсталляции MySQL 628
Внешняя безопасность: защита сетевого доступа 633
Структура и содержимое таблиц разрешений MySQL 634
Как сервер управляет доступом клиентов 642

Головоломка с привилегиями 647
Как минимизировать риск при работе с таблицами разрешений 650
Настройка учетных записей без помощи оператора GRANT 653
Установка безопасных соединений 655

Глава 13. Резервное копирование и восстановление баз данных 661

Координация работы с сервером 663
Координирование взаимодействия с применением внутренней блокировки 665
Предотвращение взаимодействия с помощью внешней блокировки 667
Устранение неполадок 669
Использование возможностей автовосстановления сервера 669
Резервирование и копирование баз данных 670
Восстановление таблиц и данных 680
Проверка и восстановление таблиц баз данных 681
Использование резервных копий для восстановления данных 689

Часть IV. Приложения 697

Приложение А. Получение и инсталляция программного обеспечения 699

Получение и инсталляция дистрибутива базы данных sampdb 700
Получение MySQL и сопутствующего программного обеспечения 700
Инсталляция MySQL в системе UNIX 702
Еще об инсталляции MySQL 703
Создание учетной записи для пользователя MySQL 704
Получение и инсталляция дистрибутива MySQL 705
Инициализация каталога данных и таблиц разрешений 708
Запуск сервера 709
Инсталляция поддержки Perl DBI в ОС UNIX 710
Инсталляция PHP и Apache 712
Инсталляция MySQL в Windows 713
Инсталляция поддержки Perl DBI 716
Инсталляция PHP и Apache в ОС Windows 716
Инсталляция MyODBC 716

Приложение Б. Типы данных 719

Числовые типы 720
TINYINT[(*M*)] 721
SMALLINT[(*M*)] 721
MEDIUMINT[(*M*)] 722
INT[(*M*)] 722
BIGINT[(*M*)] 722
FLOAT[(*p*)] 722
FLOAT[(*M,D*)] 723
DOUBLE[(*M,D*)] 723
DECIMAL[(*M,D*)] 724
Строковые типы 724
CHAR(*M*) 724
VARCHAR(*M*) 725
TINYBLOB 725

BLOB 725
MEDIUMBLOB 726
LONGBLOB 726
TINYTEXT 726
TEXT 726
MEDIUMTEXT 726
LONGTEXT 727
ENUM("value1","value2",...) 727
SET("value1","value2",...) 727
Типы даты и времени 727
DATE 728
TIME 728
DATETIME 728
TIMESTAMP 728
YEAR 729

Приложение В. Операторы и функции 731

Операторы 732
Приоритет операторов 732
Операторы группировки 733
Арифметические операторы 733
Операторы сравнения 735
Битовые операторы 739
Логические операторы 740
Операторы преобразования 742
Операторы сравнения с образцом 743
Функции 746
Функции сравнения 747
Функции преобразования 749
Числовые функции 750
Строковые функции 754
Функции даты и времени 764
Суммирующие функции 772
Функции безопасности 774
Другие функции 777

Приложение Г. Синтаксис SQL 783

Операторы языка SQL 784
ALTER DATABASE 784
ALTER TABLE 784
ANALYZE TABLE 788
BACKUP TABLE 788
BEGIN 789
CHANGE MASTER 789
CHECK TABLE 790
COMMIT 791
CREATE DATABASE 791
CREATE FUNCTION 791
CREATE INDEX 792
CREATE TABLE 792

DELETE 799
DESCRIBE 800
DO 801
DROP DATABASE 801
DROP FUNCTION 801
DROP INDEX 801
DROP TABLE 802
EXPLAIN 802
FLUSH 803
GRANT 804
HANDLER 808
INSERT 808
KILL 809
LOAD DATA 810
LOAD ... FROM MASTER 814
LOCK TABLE 814
OPTIMIZE TABLE 815
PURGE MASTER LOGS 816
RENAME TABLE 816
REPAIR TABLE 816
REPLACE 817
RESET 817
RESTORE TABLE 818
REVOKE 818
ROLLBACK 819
SELECT 819
SET 825
SET TRANSACTION ISOLATION LEVEL 830
SHOW 830
SLAVE 855
TRUNCATE 855
UNION 856
UNLOCK TABLE 856
UPDATE 856
USE 857
Переменные языка SQL 857
Синтаксис комментариев 858

Приложение Д. Программы MySQL 861

Определение параметров программ 863
"Великое разделение" процедуры обработки параметров в MySQL 4.0.2 864
Стандартные параметры программ MySQL 865
Конфигурационные файлы 868
Переменные среды 873
Библиотека libmysqld 874
Программы isamchk и myisamchk 874
Использование 875
Стандартные параметры, поддерживаемые программами isamchk и myisanchk 875
Общие параметры программ isamchk и myisamchk 876
Специальные параметры программы myisamchk 877
Переменные программ isamchk и myisamchk 879

Программы myisampack и pack_isam 880
Использование 880
Стандартные параметры, поддерживаемые программами myisampack и pack_isam 880
Общие параметры программ myisampack и pack_isam 881
Специальные параметры программы pack_isam 881
Программа mysql 882
Использование 883
Стандартные параметры, поддерживаемые программой mysql 883
Специальные параметры mysql 883
Переменные mysql 887
Команды mysql 888
Определение последовательностей подсказки mysql 889
Сценарий mysqlaccess 890
Использование 890
Стандартные параметры, поддерживаемые сценарием mysqlaccess 890
Специальные параметры mysqlaccess 890
Программа mysqladmin 891
Использование 891
Стандартные параметры, поддерживаемые программой mysqladmin 891
Специальные параметры mysqadmin 892
Переменные программы mysqladmin 892
Команды mysqladmin 892
Программа mysqlbinlog 894
Использование 894
Стандартные параметры, поддерживаемые программой mysqlbinlog 894
Специальные параметры mysqlbinlog 895
Сценарий mysqlbug 895
Использование 896
Программа mysqlcheck 896
Использование 896
Стандартные параметры, поддерживаемые mysqlcheck 896
Специальные параметры mysqlcheck 897
Программа mysql_config 899
Использование 899
Специальные параметры mysql_config 899
Программа mysqld 899
Использование 900
Стандартные параметры, поддерживаемые программой mysqld 900
Специальные параметры mysqld 901
Переменные mysqld 913
Сценарий mysqld_multi 914
Использование 914
Стандартные параметры, которые поддерживаются программой mysqld_multi 914
Специальные параметры mysqld_multi 914
Сценарий mysqld_safe 915
Использование 915
Специальные параметры mysld_safe 915
Подсказки по модернизации 916
Программа mysqldump 916
Использование 916
Стандартные параметры, поддерживаемые программой mysqldump 917

Специальные параметры mysqldump 917
Параметры форматирования данных 920
Переменные mysqldump 920
Сценарий mysqlhotcopy 921
Использование 921
Стандартные параметры, поддерживаемые программой mysqlhotcopy 921
Специальные параметры mysqlhotcopy 922
Программа mysqlimport 924
Использование 924
Стандартные параметры, поддерживаемые программой mysqlimport 924
Специальные параметры mysqlimport 924
Параметры форматирования данных 925
Сценарий mysql_install_db 926
Использование 926
Специальные параметры mysql_install_db 926
Сценарий mysql.server 927
Использование 927
Программа mysqlshow 927
Использование 927
Стандартные параметры, поддерживаемые программой mysqlshow 928
Специальные параметры mysqlshow 928
Сценарий safe_mysqld 928

Приложение Е. Программный интерфейс приложений C 929

Компилирование и редактирование связей 930
Типы данных программного интерфейса приложений C 931
Скалярные типы данных 931
Нескалярные типы данных 932
Макросы 936
Функции программного интерфейса приложений C 937
Функции управления соединением 937
Функции состояния и диагностики ошибок 944
Функции создания и выполнения запросов 945
Функции обработки результирующих наборов 947
Информационные функции 956
Административные функции 958
Функции поддержки процессовых клиентов 958
Функции обмена данными со встроенным сервером 959
Функции отладки 960
Исключенные функции 960

Приложение Ж. Программный интерфейс DBI языка написания сценариев Perl 965

Написание сценариев 966
Методы интерфейса DBI 966
Методы класса DBI 968
Методы дескрипторов баз данных 972
Методы дескрипторов операторов 977
Методы общих дескрипторов 981
Специальные административные методы MySQL 981

Функции программ интерфейса DBI 982
Атрибуты интерфейса DBI 983
Атрибуты дескрипторов баз данных 984
Общие атрибуты дескрипторов 984
Специальные атрибуты дескрипторов баз данных MySQL 985
Атрибуты дескрипторов операторов 986
Специальные атрибуты дескрипторов операторов MySQL 988
Динамические атрибуты 990
Переменные среды интерфейса DBI 990

Приложение З. Программный интерфейс языка PHP 993

Написание сценариев PHP 994
Функции 995
Функции управления соединением 996
Функции состояния и диагностики ошибок 999
Функции построения и выполнения запросов 1001
Функции обработки результирующих наборов 1006
Информационные функции 1019
Устаревшие функции 1021

Приложение И. Провайдеры услуг Internet 1023

Выбор провайдера 1024
Пропускная способность 1025
Службы 1025
Оценка провайдера 1026
Различные аспекты работы с MySQL 1032

Предметный указатель 1036

Об авторе

Поль Дюбуа (Paul DuBois) – писатель, администратор баз данных и глава сообщества "Open Source". Он является также автором книг *Применение MySQL и Perl в Web-приложениях, MySQL Cookbook, Using csh and tcsh* и *Software Portability with imake.*

О рецензентах

Большое спасибо рецензентам, которые поделились своим огромным опытом и внесли неоценимый вклад во второе издание книги *MySQL*. Они проанализировали техническое содержание материала, его организацию и порядок изложения. Их критические замечания способствовали тому, что второе издание в полной мере соответствует потребностям читателей в высококачественной технической информации.

Шейн Кирк (Shane Kirk) получил степень бакалавра компьютерных наук в университете штата Кентукки. В настоящее время он проживает в городе Цинциннати, штат Огайо, и работает администратором баз данных и разработчиком компьютерных программ для компании Opinion One (`www.opinionone.com`). Эта компания специализируется на разработке программного обеспечения в области маркетинговых исследований и методов принятия решений.

Доктор Ханг Т. Лау (Dr. Hang T. Lau) – профессор факультета компьютерных наук в университете Конкордия (Монреаль, Канада). Он работает в этой отрасли больше 20 лет как ученый в областях планирования телекоммуникационных сетей, прикладных систем распознавания речи в телекоммуникациях и систем транспортировки данных по доступным радиосетям.

Благодарности

Благодарности представлены по изданиям этой книги.

Первое издание

Книга очень выиграла от комментариев, советов и критических замечаний, полученных от рецензентов Дэвида Аксмарка (David Axmark), Виджея Чогула (Vijay Chaugule), Чэда Кэннингэма (Chad Cunningham), Билла Джерарда (Bill Gerrard), Джиджо Джорджа Джона (Jijo George John), Фрэда Рида (Fred Read), Эгона Шмидта (Egon Schmid) и Яни Толонена (Jani Tolonen). Особую благодарность хочется выразить Михаэлю "Monty" Видениусу (Michael "Monty" Widenius), основному разработчику СУБД MySQL, который не только принял живое участие в рецензировании этой книги, но и без промедления отвечал на все мои вопросы, которые я задавал в процессе работы над книгой. Хотелось бы также поблагодарить Томаса Карлссона (Tomas Karlsson), Колина МакКинона (Colin McKinnon), Сашу Пачева (Sasha Pachev), Эрика Сэвэджа (Eric Savage), Дерика Сидовэя (Derick H. Siddoway) и Боба Ворти (Bob Worthy). Благодаря всем им книга приобрела свой "окончательный" вид. Хочется поблагодарить и весь штат сотрудников издатель-

ства New Riders, которому мы обязаны за саму идею этой книги, а потом и за чудесное превращение моих каракулей в законченный труд, который вы сейчас держите в своих руках. Благодарю Лори Петрицки (Laurie Petrycki), который был исполнительным редактором. Кэти Педам (Katie Purdum), ведущего редактора, направлявшую всю мою работу и принявшую бой, когда я не вписывался в сроки. Лию Вильямс (Leah Williams), взявшую на себя двойную обязанность редактора, отвечающего за проект, и выпускающего редактора, за то, что ей приходилось допоздна засиживаться на работе особенно на завершающей стадии проекта. Шерил Ленсер (Cheryl Lenser) и Тима Райта (Tim Wright), разработавших предметный указатель. Джона Рама (John Rahm), выполнявшего обязанность редактора проекта. Еще раз большое спасибо всем им.

И больше всего хочу выразить признательность моей жене Карэн (Karen) за то, что освободила меня от всякой другой работы. За ее понимание моих частых исчезновений в "писательскую зону" и долготерпение. Ее поддержка облегчила мою задачу, и я с удовольствием хочу поблагодарить ее за неоценимый вклад; ее помощь нашла свое отражение на каждой странице.

Второе издание

Во втором издании рецензенты вновь сыграли решающую роль в процессе вычитывания, внесения поправок и уточнений. Ханг Лау (Hang Lau) и Шейн Кирк (Shane Kirk) приняли участие в этом проекте как рецензенты. Мне бы хотелось также поблагодарить Монти Видениуса (Monty Widenius), Александра Баркова (Alexander Barkov), Яни Толонена (Jani Tolonen) и других разработчиков MySQL за терпение к моим многочисленным вопросам и за ответы, которые нашли отражение на страницах настоящей книги.

Хочется поблагодарить весь штат издательства New Riders, который принял участие в воплощении этого издания в жизнь. Это были помощник издателя Стэфани Уолл (Stephanie Wall), редактор, отвечающий за проект, Крис Зан (Chris Zahn), главный редактор проекта Лори Лайонс (Lori Lyons), выпускающий редактор Пэт Кинион (Pat Kinyon), составитель предметного указателя Шерил Ленсер (Cheryl Lenser) и наборщик Стэйси Ричвайн-ДеРом (Stacey Richwine-DeRome).

И, как обычно, моя жена Карэн (Karen) обеспечивала поддержку, которую читатели не видят, но без которой книга была бы беднее.

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо, либо просто посетить наш Web-сервер и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг. Наши координаты:

E-mail: info@williamspublishing.com

WWW: http://www.williamspublishing.com

Информация для писем из:

России: 115419, Москва, а/я 783

Украины: 03150, Киев, а/я 152

Введение

В настоящее время реляционные системы управления базами данных (СУБД) являются важным инструментом во многих областях, начиная с таких традиционных областей применения, как бизнес, научные исследования, образование, и заканчивая разработкой поисковых серверов в Internet. Однако, несмотря на важность наличия хорошей базы данных для ведения информационных ресурсов и доступа к ним, многие организации не применяют их в своей работе. Исторически сложилось так, что СУБД стоили очень дорого, а продавцы устанавливали очень высокие цены как на программное обеспечение, так и на свои услуги по технической поддержке. Кроме того, механизмы СУБД требовали удовлетворения существенных требований по производительности от аппаратных платформ, что еще больше повышало стоимость таких решений.

В последние годы ситуация резко изменилась как с точки зрения программного обеспечения, так и с точки зрения аппаратуры. Персональные компьютеры одновременно дешевеют и становятся мощнее. При этом обозначилась тенденция создания высокопроизводительных операционных систем, которые можно купить по цене дешевых лазерных дисков или получить бесплатно через Internet, таких как операционные системы, созданные на базе ОС BSD UNIX (FreeBSD, NetBSD, OpenBSD), а также различные версии ОС Linux (RedHat, Caldera, LinuxPPC).

Создание операционных систем, позволяющих максимально использовать возросшие возможности компьютеров, произошло прежде всего благодаря тому, что были разработаны и свободно распространялись такие средства разработки, как компилятор GNU C gcc. Эти попытки создания программного обеспечения, которое было бы доступно для каждого, кто хочет его получить, дало толчок движению, которое сейчас известно как "Open Source movement", и дало жизнь многим важным и нужным программам. В качестве примера успешного применения идеологии можно вспомнить сервер Apache, который является самым широко используемым сервером в Internet. К числу успешных проектов "Open Source" относятся также язык написания сценариев Perl и быстро завоевывающий поклонников язык PHP, который популярен во многом благодаря простоте, с которой он позволяет создавать динамичные Web-страницы. Все это находится в разительном контрасте с патентованными решениями за очень высокую цену, которые при этом даже не сопровождаются исходными текстами.

Программное обеспечение баз данных тоже стало более доступным. Достаточно вспомнить такие СУБД, как Postgres, которую можно получить бесплатно. Совсем недавно такие мощные коммерческие производители, как Informix и Oracle, начали предлагать свое программное обеспечение бесплатно для таких ОС, как Linux. Однако эти продукты поставляются обычно в двоичном виде и без поддержки, что резко снижает их эффективность.

Одним из новейших явлений на "арене" недорогих или бесплатных баз данных является MySQL, реляционная СУБД типа клиент/сервер, созданная в Скандинавии. СУБД MySQL включает в себя SQL-сервер и программы-клиенты, осуществ-

ляющие доступ к серверу, средства администрирования и программный интерфейс для программирования своих собственных программ.

СУБД MySQL уходит своими корнями в 1979 год и берет свое начало от СУБД UNIREG, разработанной Михаэлем "Монти" Видениусом по заказу шведской компании TcX. В 1994 году компания TcX начинает искать SQL-сервер для создания Web-приложений. Было опробовано несколько коммерческих серверов, но те оказались слишком медленными для больших таблиц данных, которые использовались в компании TcX. Они также обратили внимание на СУБД mSQL, но та не совсем удовлетворяла задачам компании TcX. Поэтому Монти начал работать над созданием нового сервера. Программный интерфейс был разработан как аналог mSQL, так как тогда в наличии было несколько бесплатных средств mSQL. Пользуясь аналогичным интерфейсом, эти же средства можно использовать для СУБД MySQL с минимальными затратами на перенос.

В 1995 году Дэвид Аксмарк (David Axmark), работающий в компании Detron HB, начал "давление" на TcX с тем, чтобы она начала распространять СУБД MySQL через Internet. Кроме того, Дэвид принял участие в работе над документацией. Версия 3.11.1 СУБД MySQL была выпущена в свет в 1996 году в виде двоичного дистрибутива для работы под управлением ОС Linux и Solaris. Сегодня MySQL работает на многих платформах и распространяется как в двоичных кодах, так и в исходных текстах. Для распространения СУБД MySQL, оказания помощи и обучения была создана компания MySQL AB.

А СУБД MySQL продолжает развиваться. Добавление таких возможностей, как транзакции, блокировки на низких уровнях, применение внешних ключей и репликации, позволило обратить на себя внимание специалистов, которые для решения своих прикладных задач раньше привлекали только "большие СУБД".

MySQL – это проект сообщества "Open Source", который может быть использован свободно почти во всех случаях, что способствует быстрому росту его популярности в кругу членов сообщества "Open Source". Но популярность СУБД MySQL не ограничивается только сообществом "Open Source". Да, она работает на персональных компьютерах (при этом многие разработки, производящиеся на MySQL, создаются на недорогих Linux-системах). Но MySQL обладает отличной переносимостью и может с тем же успехом использоваться на дорогих коммерческих операционных системах (таких как Solaris, Mac OS X или Windows) и на любой аппаратуре, вплоть до мощных серверов. Более того, так же как и ее более "дорогие соперники", она позволяет обрабатывать большие базы данных, содержащие миллионы записей.

Теперь СУБД MySQL предстала перед нами во всей красе: бесплатные операционные системы, работающие на мощных, но не очень дорогих персональных компьютерах, предоставляющих в распоряжение пользователей значительную вычислительную мощность и более широкий выбор операционных систем, чем прежде. Снижение экономических барьеров позволяет получить доступ к базам данных большему количеству людей и организаций, чем когда бы то ни было. Организации, которые когда-то могли только мечтать о вводе в работу мощности высокопроизводительной СУБД для работы на них, могут сделать это сейчас по очень низкой цене. Это же верно для индивидуальных пользователей. Я, например, общаюсь с СУБД MySQL с помощью языков Perl и PHP, сервера Apache на

ноутбуке Apple iBook, работающем под управлением ОС Mac OS X. Это позволяет мне работать в любой точке земного шара. Полная стоимость такого проекта равна стоимости моего ноутбука.

Все-таки, почему MySQL?

Если вы ищете бесплатную или недорогую СУБД, можно остановить свой выбор на одной из трех кандидатур: MySQL, PostgreSQL или на бесплатных (но несопровождаемых СУБД) от крупных производителей. Сравнивая СУБД MySQL с другими базами данных, всегда помните о том, какой фактор является для вас самым важным. Производительность, поддержка, возможности (соответствие стандартам языка SQL, возможность расширения и т.д.), условия лицензирования и ограничения и, наконец, цена. Принимая во внимание все эти соображения, СУБД MySQL имеет ряд преимуществ.

- **Быстродействие.** MySQL — достаточно быстродействующая СУБД. Разработчики склоняются к мнению, что СУБД MySQL является одной из самых быстрых баз данных из имеющихся на современном рынке. В этом можно удостовериться, посетив Web-узел `http://www.mysql.com/benchmark.html`. Эта страница позволяет выполнить сравнительную проверку производительности на Web-узле MySQL.
- **Простота использования.** СУБД MySQL является высокопроизводительной и относительно простой в использовании СУБД, которую значительно проще инсталлировать и администрировать, чем многие большие системы.
- **Поддержка языка запросов.** MySQL "понимает" команды языка SQL, языка запросов, нашедшего применение во всех современных СУБД.
- **Возможности.** Сервер позволяет одновременно подключаться неограниченному количеству пользователей. Доступ к серверу СУБД MySQL можно осуществить в интерактивном режиме с помощью различных интерфейсов, позволяющих вводить запросы и просматривать полученные результаты: это программы-клиенты, работающие с командной строкой, Web-броузеры или программы-клиенты, работающие в оболочке X Window. Кроме того, в наличии имеются программные интерфейсы для таких языков, как C, Perl, Java, PHP и Python. Доступ к базам данных СУБД MySQL возможен и с помощью приложений, поддерживающих ODBC (открытое взаимодействие с базами данных) протокола связи с базами данных, разработанного компанией Microsoft. Таким образом, можно использовать как готовое клиентское программное обеспечение, так и создать свое собственное.
- **Взаимодействие и безопасность.** MySQL предназначена для работы в сети и может быть доступна через Internet. Таким образом, с данными можно работать в любой точке земного шара. Но при этом СУБД MySQL снабжена развитой системой защиты от несанкционированного доступа. Для обеспечения дополнительной защиты СУБД MySQL теперь поддерживает криптируемые соединения с использованием протокола SSL.
- **Переносимость.** СУБД MySQL отлично работает как под управлением самых различных версий UNIX, так и под управлением систем, не использую-

щих UNIX, таких как Windows и OS/2. СУБД MySQL может работать как на домашних ПК, так и на мощных серверах.

- **Малый размер.** СУБД MySQL имеет ограниченный размер, особенно по сравнению с огромным дисковым пространством, необходимым большинству коммерческих СУБД.
- **Работоспособность и цена.** MySQL является проектом Open Source, легкодоступным по условиям GNU General Public License (GPL). Это означает, что СУБД MySQL распространяется бесплатно для большинства домашних пользователей. (Если вы хотите продавать СУБД MySQL или услуги, для которых она требуется, то это совсем другая ситуация и в этом случае вам следует связаться с компанией MySQL AB.)
- **Свободное распространение.** Дистрибутив СУБД MySQL легкодоступен. Для этого достаточно воспользоваться Web-броузером. Если вы не понимаете, как что-либо работает, просмотрите исходный код. Если вам что-то не нравится в работе, можно внести коррективы. Если вы нашли ошибку, сообщите о ней разработчикам, чтобы они проверили ее.

А как обстоят дела с поддержкой? Хороший вопрос! Тем более что от базы данных, которой нельзя пользоваться, польза невелика. Естественно, мне бы хотелось верить, что эта книга будет той поддержкой, которая вам потребуется, но, конечно, после прочтения этой книги у вас возникнут вопросы, о которых я сам никогда не задумывался или просто не хватало времени для описания такой проблемы. А ответы на вопросы можно найти и в других источниках. Кроме того, СУБД MySQL имеет хорошую поддержку. MySQL легкодоступна, и вы не остаетесь в полном одиночестве после ее инсталляции.

- В составе дистрибутивов СУБД MySQL имеется *Справочное руководство по СУБД MySQL*, которое доступно также в режиме "online". Справочное руководство по СУБД MySQL регулярно получает хорошие оценки в кругу пользователей СУБД MySQL. Это очень важно, так как ценность хорошего продукта снижается, когда он не находит правильного применения.
- Те, кто предпочитает формальные договоренности, может заключить контракты с компанией MySQL AB на техническую поддержку и проведение курсов обучения.
- Есть список рассылки, на который может подписаться любой желающий. В нем принимают участие очень много грамотных пользователей, в том числе и сами разработчики MySQL. Этого будет вполне достаточно для большинства пользователей СУБД MySQL.

Общество MySQL, разработчики и не разработчики — очень ответственные люди. Ответы на вопросы, размещенные в списке рассылки, можно получить за несколько минут. Обнаружив ошибку, разработчики исправят ее за считанные дни, и исправления можно будет получить по Internet немедленно. Это резко контрастирует с нередко разочаровывающим опытом работы с бюрократическими отделами поддержки больших компаний, специализирующихся на продажах СУБД. (Вы уже это знаете? Я тоже. Я сам знаю, что мне лучше делать, когда у меня возникает вопрос об использовании того или иного программного продукта. Воз-

никает альтернатива: ожидать момента, когда продавец соблаговолит, наконец, найти время для ответа, или просто разместить свой вопрос в списке рассылки и проверять полученные ответы в удобное для меня время).

СУБД MySQL является идеальным кандидатом для оценки при выборе базы данных. СУБД MySQL можно протестировать без какого-либо риска или финансовых издержек. Если возникнут проблемы, для получения помощи можно воспользоваться списками рассылки. Оценка потребует какого-то времени, но это не зависит от рассматриваемой СУБД и совершенно определенно, что фаза инсталляции и установки в случае выбора СУБД MySQL потребует значительно меньше времени, чем для многих других систем.

Если вы уже работаете с другой СУБД

Нужно ли переходить на MySQL, если вы уже работаете с другой СУБД? Это совсем необязательно. Зачем что-то менять, если вас вполне удовлетворяет система, на которой вы работаете в данный момент? Но если вас уже не удовлетворяет то, что вы используете, вам определенно стоит рассмотреть MySQL как возможную альтернативу используемой СУБД. Вероятно, что проблемой является производительность вашей системы или права собственности и вы не хотите быть ограниченными этими рамками. Возможно, вы планируете перейти на новое аппаратное обеспечение, которое не поддерживает работу с вашей системой, или ваше программное обеспечение поставляется только в двоичном виде, а вам хотелось бы иметь исходные тексты, но ваша СУБД слишком дорого стоит! Это все может стать причиной того, чтобы обратить внимание на MySQL. Воспользуйтесь этой книгой, чтобы получить достаточно полное представление о возможностях MySQL, кроме того, можно задавать вопросы, используя список рассылки, и, вероятно, вы сможете найти ответы на все свои вопросы.

Планируя переход с другой СУБД, работающей с SQL, обратитесь к сравнительной таблице на Web-узле MySQL по адресу `http://www.mysql.com/information/crash-me.php`. После этого просмотрите главы, где описаны типы данных СУБД MySQL и диалект языка SQL. Вероятно, вы найдете, что диалекты языка SQL вашей СУБД и СУБД MySQL сильно отличаются и перенос ваших приложений потребует больших затрат.

Частично процедура оценки должна включать несколько пробных переносов, потому что на практике это может оказаться значительно проще, чем вы думаете. Это может оказаться не настолько затруднительным, даже если ваша СУБД очень "древняя" и не "понимает" команд SQL. Недавно я занимался преобразованием данных из формата СУБД, которую никак не назовешь СУБД SQL-типа. Типы данных не имели своих SQL-эквивалентов. Этот проект заключался в преобразовании методов доступа к сети, дюжины экранных программ и стандартных запросов. Переход потребовал полтора месяца напряженной работы. Вышло достаточно солидно.

Инструментарий, поставляемый с MySQL

Дистрибутив MySQL включает следующие программные продукты.

- **SQL-сервер.** Эта программа обеспечивает доступ к базам данных.

- **Клиентские программы доступа к серверу.** Это интерактивная программа, позволяющая делать запросы и просматривать полученные результаты, и несколько административных и сервисных программ, помогающих работать с вашим Web-узлом. Одна сервисная программа позволяет управлять сервером. Другие позволяют импортировать или экспортировать данные, проверять права доступа и т. д.
- **Клиентская программа, позволяющая создавать свои собственные программы.** Клиентские программы можно писать на C, так как сама библиотека написана на C, но библиотека также позволяет создавать шлюзы для написания программного обеспечения на других языках программирования.

В дополнение к программному обеспечению, которое поставляется с MySQL, СУБД MySQL используется многими талантливыми и способными людьми, которые имеют обыкновение создавать программное обеспечение для повышения своей производительности и хотят, чтобы эти достижения были доступны всем желающим. В результате, если вы остановите свой выбор на СУБД MySQL, у вас в арсенале будет множество программных продуктов других производителей, которые значительно облегчат вам работу с MySQL.

Для кого предназначена эта книга

В этой книге можно познакомиться с методами эффективной работы как с СУБД MySQL, так и с СУБД вообще. Это сделает вашу работу с MySQL более продуктивной. Вы познакомитесь с процессом наполнения базы данных информацией, узнаете, как формулировать запросы к базе данных, чтобы получить ответы на вопросы, которые могут вас заинтересовать.

Совсем не надо быть программистом или уметь работать с языком запросов SQL. Эта книга рассказывает, как это делается. Книга больше посвящена проблеме правильной работы с СУБД, чем тонкостям синтаксиса языка SQL. Она акцентирует внимание на уникальных возможностях СУБД MySQL, показывая, как можно их использовать с максимальной эффективностью.

Читатель познакомится также с методами интеграции MySQL с другими средствами. В книге показаны приемы работы с MySQL с помощью языков PHP и Perl для создания динамических Web-страниц на основе запросов к базе данных. Читатель узнает, как можно написать свою собственную несложную программу, которая сможет работать с базами данных СУБД MySQL. Все это расширяет возможности СУБД MySQL и позволяет с ее помощью решать задачи, стоящие перед вами.

Если вы несете ответственность за администрирование СУБД MySQL, эта книга разъяснит вам ваши обязанности и то, как их выполнять. Читатель узнает, как создаются учетные записи пользователей, как делается резервное копирование баз данных, как обеспечивается полная безопасность вашего Web-узла.

Структура книги

Данная книга состоит из четырех частей.

Часть I. Использование СУБД MySQL

- **Глава 1, "Знакомство с СУБД MySQL и SQL".** Разъясняется, для чего используется СУБД MySQL, описывается процесс обучения работе с интерактивной клиентской программой MySQL и даются основные положения языка SQL.
- **Глава 2, "Работа с данными в MySQL".** Описываются типы столбцов СУБД MySQL, свойства и ограничения каждого типа, когда и как их можно использовать, как делать выбор между подобными типами данных, производить оценку выражения и преобразование типов.
- **Глава 3, "Синтаксис и использование языка SQL".** Описывается специфический диалект языка SQL, применяемого в MySQL. В этой главе особый акцент делается на тех особенностях SQL, которые присущи диалекту MySQL. Тут также описаны особенности, присущие СУБД MySQL.
- **Глава 4, "Оптимизация запросов".** Затронуты проблемы повышения эффективности работы запросов.

Часть II. Программные интерфейсы СУБД MySQL

- **Глава 5, "Введение в программирование в СУБД MySQL".** Описываются программные интерфейсы, которые имеются в MySQL, и сравниваются программные интерфейсы API, детально описанные в этой книге.
- **Глава 6, "Программный интерфейс MySQL C API".** Изложены примеры создания программ на языке C с использованием интерфейса API из клиентской библиотеки, поставляемой с дистрибутивом MySQL.
- **Глава 7, "Программный интерфейс Perl API".** Показаны методы использования модуля DBI при создании сценариев Perl. Обсуждается проблема автономных и CGI-сценариев, предназначенных для программирования Web-узлов.
- **Глава 8, "Интерфейс API для языка написания сценариев PHP".** Рассказывается, как можно использовать язык написания сценариев PHP для создания динамических Web-страниц, осуществляющих доступ к MySQL.

Часть III. Администрирование MySQL

- **Глава 9, "Введение в администрирование MySQL".** Показано, в чем состоят обязанности администратора базы данных и что ему нужно знать, чтобы Web-узел функционировал успешно.
- **Глава 10, "Каталог данных MySQL".** Детально описываются структура и содержимое каталога данных; кроме того, вы узнаете, где СУБД MySQL хранит данные и файлы состояния.
- **Глава 11, "Общее администрирование MySQL".** Показано, как можно убедиться в том, что сервер запускается и останавливается правильно во время запуска и остановки вашей системы. Кроме того, в этой главе содержатся

инструкции по установке учетных записей MySQL, описываются процедуры по обслуживанию файлов регистрации, настройке сервера, приводятся стратегии резервного копирования и восстановления.

- **Глава 12, "Безопасность".** Разъясняется, что надо сделать для того, чтобы обезопасить СУБД MySQL от несанкционированного доступа пользователей на узле сервера и клиентов, подключающихся по сети. Описывается, как настраивается сервер MySQL для поддержки надежных связей с использованием протокола SSL.
- **Глава 13, "Резервное копирование и восстановление баз данных".** Рассказывается, как с помощью профилактического обслуживания можно сократить вероятность отказа сервера и как происходит восстановление баз данных после отказа, если он все-таки произошел.

Часть IV. Приложения

- **Приложение А, "Получение и инсталляция программного обеспечения".** Объясняется, где можно приобрести и как установить программные средства, представленные в этой книге.
- **Приложение Б, "Типы данных".** Описываются типы данных СУБД MySQL.
- **Приложение В, "Операторы и функции".** Описываются операторы и функции диалекта SQL.
- **Приложение Г, "Синтаксис SQL".** Приводятся все операторы SQL.
- **Приложение Д, "Программы MySQL".** Описываются все программы, имеющиеся в дистрибутиве MySQL.
- **Приложение Е, "Программный интерфейс приложений C".** Приводятся все типы данных и функции клиентской библиотеки MySQL C.
- **Приложение Ж, "Программный интерфейс DBI языка написания сценариев Perl".** Описываются все методы и атрибуты, поддерживаемые модулем DBI языка Perl.
- **Приложение З, "Программный интерфейс языка PHP".** Описываются все функции, которые имеются в языке PHP, предназначенные для поддержки СУБД MySQL.
- **Приложение И, "Провайдеры услуг Internet".** Перечисляются факторы, которые принимаются во внимание при выборе провайдера для обеспечения доступа к MySQL (что нужно предусмотреть провайдеру, предоставляющему доступ к MySQL-серверу).

Как читать эту книгу

Независимо от того, какую часть книги вы будете читать, это лучше всего сопровождать выполнением примеров. Если СУБД MySQL не установлена, вам необходимо ее установить либо попросить кого-то сделать это для вас, а затем получить все файлы, необходимые для установки sampdb — текстовой базы данных, ссылка на которую будет делаться во всей книге. В приложении А, "Получение и инсталляция программного обеспечения", объясняется, где можно получить все инструкции по установке.

Если вы ничего не знаете о MySQL или языке SQL, начните с главы 1. Там есть введение, в котором даются базовые знания о MySQL и языке SQL, что позволит упростить восприятие книги. После этого можно перейти к остальному материалу. В главах 2 и 3 речь пойдет о том, как создается структура данных и как можно манипулировать ими.

Даже если вы уже знакомы с языком SQL, вам все-таки будет полезно ознакомиться с главами 2 и 3. Все диалекты языка SQL различаются, и нужно знать, в чем заключается отличие данного диалекта SQL от тех диалектов, с которыми вы уже знакомы.

Вероятно, вы уже имеете определенное представление о СУБД MySQL, но нуждаетесь в освоении методов решения конкретных задач. В этом случае данную книгу можно использовать в качестве справочника, знакомясь с темами по мере необходимости. В книге имеется несколько приложений, которые могут пригодиться в справочных целях.

Вы собираетесь создавать программы, обрабатывающие данные из баз данных? В этом случае рекомендуется ознакомиться с главами, посвященными программным интерфейсам, начиная с главы 5. Если вам необходимо создать Web-интерфейс для работы со своей СУБД MySQL или расширить возможности своего Web-узла, добавив динамические возможности СУБД, ознакомьтесь с содержанием глав 7 и 8.

Если вы уже пользуетесь какой-то СУБД и хотите получить сравнительную характеристику этой СУБД с MySQL, то в этом вам поможет несколько глав данной книги. Для сравнения MySQL и SQL, которую вы сейчас используете в своей работе, прочтите главы, посвященные типам данных и синтаксису SQL (часть I). Если вас интересует проблема адаптации существующего прикладного программного обеспечения, ознакомьтесь с соответствующими главами из части II и главами, посвященными решению задач администрирования, из части III, чтобы получить руководство по административной поддержке, которая требуется для установки MySQL. Эта информация будет также полезна, если вы не используете базу данных, а делаете сравнительный анализ СУБД MySQL с другими СУБД, чтобы выбрать одну из них.

Вам нужен доступ к MySQL, и вы ищете подходящего провайдера услуг Internet (Internet Service Provider — ISP), который предоставляет такой сервис? Для этого обратитесь к приложению И, "Провайдеры услуг Internet". Кроме того, это приложение содержит советы для провайдеров, планирующих предоставлять услуги сервера MySQL.

Версии программного обеспечения, которые упоминаются в книге

На момент написания этой книги стабильно работающими версиями MySQL являются все версии серии 4.0, а также ведется активная разработка серии 4.1. Эта книга опирается на обе вышеупомянутые серии, а также на 3.22 и 3.23.

Для других основных пакетов, описанных здесь, все сказанное в книге справедливо для любой последней версии. Текущими версиями других описанных в этой книге программных продуктов являются следующие.

Пакет	Версия
PerlDBI	1.32
Драйвер Perl MySQL DBI	2.1020
PHP	4.2.3
Apache	1.3.27/2.0.43
CGI.pm	2.87

Все программное обеспечение, описанное в этой книге, можно получить непосредственно из Internet. Инструкции, каким образом можно получить MySQL, поддержку Perl DBI, PHP, Apache и CGI.pm вашей системы, даны в приложении A, "Получение и инсталляция программного обеспечения". Это приложение также содержит руководство по получению тестовой базы данных, которая используется в примерах, приведенных в этой книге, и примеры программ, которые разработаны в главах этой книги.

Соглашения, используемые в книге

В этой книге приняты следующие типографские соглашения.

Специальным шрифтом обозначаются имена узлов, файлов, каталогов, а также команды, параметры и Web-узлы. Команды отображены в том виде, в котором они выводятся. **Полужирным** шрифтом выделяется часть, которая вводится. *Курсивом* обозначается часть команды, которая должна быть заменена вводимым значением.

Подсказка в командах указывает, как выполняется команда. Для большинства команд используется подсказка `%`. В общем случае это может быть как подсказкой в операционной системе UNIX, так и подсказкой ОС DOS. Более специализированными подсказками являются подсказка #, указывающая на выполнение команды пользователем `root` ОС UNIX, и подсказка `C:\>`, отражающая готовность ОС Windows к вводу команд. При работе с программой `mysql` операторы SQL вводятся после появления подсказки `mysql>` на экране.

Операторы SQL, ключевые слова SQL и функциональные наименования представлены в верхнем регистре. Базы данных, таблицы и названия столбцов представлены в нижнем регистре. В синтаксическом описании в квадратных скобках ([]) указывают дополнительную информацию.

Термин "Системы Windows, базирующиеся на NT" совокупно отражает семейство Windows, базирующихся на NT-технологии. Теперь это семейство включает ОС Windows NT, Windows 2000 и Windows XP. ОС Windows 95, 98 и Windows Me в это семейство не входит.

Дополнительные источники информации

Целью этой книги является изложение материала, необходимого для работы с СУБД MySQL. Но куда обратиться, если возникает вопрос, на который в книге нет ответа?

В числе полезных источников информации можно назвать следующие Web-узлы.

Пакет	Web-узел
MySQL	`http://www.mysql.com/documentation/`
Perl DBI	`http://dbi.perl.org/`
PHP	`http://www.php.net/`
Apache	`http://www.apache.org/`
CGI.pm	`http://stein.cshl.org/WWW/software/CGI/`

Эти узлы содержат ссылки на различные виды информации, такие как справочные руководства, самые часто задаваемые вопросы и списки рассылки.

- **Справочное руководство.** СУБД MySQL поставляется в комплекте с основной документацией — "Справочным руководством по СУБД MySQL". Оно предлагается вашему вниманию в самых различных форматах, включая и интерактивную версию. Руководство по языку PHP тоже имеется в нескольких форматах, но формат "online" имеет большую тенденцию к полноте изложения, чем печатные версии. Отдельно документированы модуль DBI и его драйвер для MySQL. В документе DBI изложены общие концепции. В документе, описывающем драйвер MySQL, перечислены специфические возможности СУБД MySQL.
- **Вопросы.** Приводятся разъяснения по наиболее популярным вопросам использования DBI, PHP и Apache.
- **Списки рассылки.** Существует несколько списков рассылки, специализирующихся на вопросах, которым посвящена данная книга. Не лишним будет подписаться на один из них. Но помните, что вам необходимо научиться пользоваться архивами, которые имеются в этих списках. Во время ознакомления с новым программным средством у вас обязательно появятся вопросы, возникавшие уже миллионы раз (и получившие ответ). Поэтому нет никакого смысла задавать их снова и снова, если можно быстро найти ответы в архивах.

Правила подписки на списки рассылки варьируются, информацию можно найти по адресам, перечисленным ниже.

Пакет	Правила подписки на списки рассылки
MySQL	`http://www.mysql.com/documentation/`
Perl DBI	`http://dbi.perl.org/`
PHP	`http://www.php.net/support.php`
Apache	`http://www.apache.org/foundation/mailinglists.html`

- **Вспомогательные Web-узлы.** Кроме основных Web-узлов, программные средства могут обсуждаться и на вспомогательных Web-узлах. Обязательно просматривайте ссылки "Links" на посещаемых вами официальных узлах.

Как работать со справочным руководством по СУБД MySQL

Чтобы быть в курсе последних событий и модификаций, обязательно периодически обращайтесь к интерактивному справочному руководству по MySQL. Это руководство модифицируется сразу же после внесения новых изменений.

I

Использование СУБД MySQL

В этой части...

Глава 1. Знакомство с СУБД MySQL и SQL
Глава 2. Работа с данными в MySQL
Глава 3. Синтаксис и использование языка SQL
Глава 4. Оптимизация запросов

1

Знакомство с СУБД MySQL и SQL

В этой главе...

Назначение СУБД MySQL
Пример базы данных
Основная терминология баз данных
Учебный курс по СУБД MySQL
Как работать с `mysql`
Что дальше?

Эта глава — введение в систему управления реляционными базами данных MySQL и язык структурированных запросов (SQL), который "понимает" СУБД MySQL. Здесь описаны основные термины и концепции, а также база данных, которая будет использована во всех примерах, приведенных в этой книге. В главе изложен обучающий материал, на примере которого можно научиться пользоваться СУБД MySQL.

Если вы — новичок в вопросах использования баз данных, если вы ничего не знаете о СУБД MySQL или языке SQL и вам нужно ознакомительное руководство, без колебаний начните чтение книги с этой главы. Читатели, которые уже имеют опыт работы с СУБД MySQL или с любой другой СУБД, могут пропустить этот материал. Однако всем будет полезно ознакомиться с разделом "Пример базы данных". Этот материал будет использоваться на протяжении всей книги.

Назначение СУБД MySQL

В этом разделе приведены примеры использования СУБД MySQL. Читатель получит представление о том, что может СУБД MySQL и как ее использовать. Если вам все это уже известно, можно смело перейти к разделу "Пример базы данных".

По сути *система баз данных* представляет собой способ манипулирования информационными списками. При этом информация может поступать из различных источников. Например, это могут быть данные, полученные в ходе исследований, деловые записи, заказы потребителей, спортивная статистка, отчеты о продажах, отчеты об ошибках или оценки учащихся. Несмотря на то что СУБД могут содержать широкий диапазон информации, вы не будете использовать СУБД только ради нее самой. Хорошим примером является список продуктов, из которого постепенно вычеркиваются купленные продукты, а потом этот список удаляется. Вряд ли вы воспользуетесь СУБД в этих целях. Даже в том случае, если у вас есть переносной ПК, вы воспользуетесь программой Блокнот, а не возможностями СУБД.

СУБД проявляет себя как мощный инструмент в том случае, когда информация, которую необходимо организовать и которой уже необходимо манипулировать, становится объемной и сложной, вследствие чего записи становятся непонятными и трудно обрабатываемыми вручную. Конечно, базы данных могут использоваться большими корпорациями, обрабатывающими миллионы транзакций в день. Но СУБД может также потребоваться для немасштабных операций, которые обрабатывает один человек в личных целях. Можно привести множество примеров ситуаций, когда СУБД будет использоваться даже в том случае, когда объемы информации не возросли до огромных размеров. Некоторые из таких ситуаций описаны ниже.

- В вашем столярном бизнесе заняты несколько работников. Нужно упорядочить записи о работниках и выплатах таким образом, чтобы иметь представление, кому и когда вы платили, получить возможность составлять отчеты о доходах в налоговые инстанции. Могут также понадобиться записи о нарядах, которые выполнял работник, и информация о том, кого планировали привлечь для выполнения этой работы.
- Вы содержите сеть складов запасных автомобильных частей. Необходимо получить оперативную информацию о том, на каком из складов лежит запасная часть, заказанная покупателем.
- Вы — продавец игрушек и зависите от колебания спроса на игрушки. Вам необходимо знать тенденции в продажах определенных товаров, чтобы иметь возможность оценить необходимость увеличения складских накоплений (для товаров, которые становятся более популярными) или уменьшения (нет нужды хранить на складе то, что уже не будет хорошо продаваться).
- Нужно проанализировать множество данных, полученных в ходе исследований за долгие годы, чтобы фраза "опубликуй или погибни" не стала эпитафией вашей карьеры. Вам необходимо "переварить" массу данных, чтобы получить итоговую информацию и более детальные выборки для подробного статистического анализа.

- Вы — известный докладчик, объезжающий страну с докладами в различных аудиториях: будь то выпускные церемонии, деловые встречи и т.д. Часто вспомнить, что вы говорили в той или иной аудитории, бывает довольно затруднительно. Поэтому потребовалось вести протоколы своих встреч, чтобы использовать их в своих планах на будущее. Это нужно для того, чтобы не повторяться в одной и той же аудитории. В этом случае запись о докладе, который вы прочитали в данной аудитории, поможет избежать повторений. Кроме того, можно оставить запись о том, насколько хорошо здесь был принят ваш доклад.
- Учитель ведет записи об успеваемости и посещаемости своих уроков. После каждого теста он записывает оценку, полученную каждым учащимся. Простым решением является запись оценок в журнале, но последующее использование этих записей затруднительно. При этом затруднительна сортировка учеников по баллам. Кроме того, учет посещаемости также становится большой проблемой.
- Вы работаете секретарем в организации и ведете список ее членов. (Организация может быть самой разной — профессиональное общество, клуб, театр, симфонический оркестр или атлетический клуб.) На основании постоянно корректируемого документа ежегодно издается список членов, который редактируется по мере изменения информации о них.

  Из-за ограничений, присущих этому методу, вы устали вести список вручную. Записи с трудом поддаются сортировке. Довольно проблематично выбрать только определенную часть каждой записи (например, перечень, состоящий только из фамилий и телефонов), или выбрать ту часть зарегистрированных членов организации, срок членства которых истек и которым следует немедленно возобновить свое членство.

  Кроме того, вы хотите избежать редактирования этого списка собственноручно, но бюджет организации не позволяет нанять кого-либо для выполнения этой работы. Вы что-то где-то слышали о "безбумажном офисе", когда информация хранится в электронном виде, но не видели никакой выгоды от внедрения этой технологии для себя. Записи о членстве хранятся в электронном виде, но, по иронии, они хранятся в труднодоступной форме.

Такие сценарии применимы как для ситуаций с большими объемами информации, так и с малыми объемами. Всех их можно охарактеризовать как класс задач, которые в принципе можно выполнить вручную, хотя эффективнее они могут обрабатываться с помощью СУБД.

Каких преимуществ можно ожидать от использования такой СУБД, как СУБД MySQL? Это зависит от ваших конкретных задач и требований и, как видно из приведенных примеров, может сильно варьироваться. Давайте представим себе ситуацию, которая возникает достаточно часто, а потому очень показательна при использовании СУБД MySQL.

СУБД часто применяется для таких задач, для решения которых обычно используются картотеки. Действительно, базу данных можно представить в некотором роде в виде большой картотеки. Можно назвать несколько очень серьезных преимуществ ведения данных в электронном виде перед хранением информации вручную.

Преимущества применения СУБД MySQL описаны ниже.

- **Сокращение времени, необходимого для ведения записей.** В случае использования СУБД не требуется много времени на просмотр всей картотеки, чтобы добавить новую запись. Вы просто вводите ее в систему, не заботясь о месте размещения.
- **Сокращение времени, необходимого для поиска записей.** При поиске данных в СУБД нет необходимости последовательно просматривать все записи, чтобы найти интересующую. Предположим, что вы работаете в стоматологическом кабинете и вам надо разослать приглашения всем пациентам, которые забыли пройти ежегодный профилактический осмотр. Для этого достаточно сделать запрос к информационной системе. Конечно, это происходит не так, как при обычном общении с людьми, когда вы формулируете свой запрос примерно таким образом: "Найдите, пожалуйста, тех пациентов, которые не посещали нас на протяжении последних 6 месяцев". Работая с базой данных, вы вводите следующее:

  ```
  SELECT last_name, first_name, last_visit FROM patient WHERE
  last_visit < DATE_SUB(CURRENT_DATE, INTERVAL 6 MONTH);
  ```

  Возможно, вы никогда не встречали ничего подобного, но перспектива получить ответ через секунду или две вместо изнурительного многочасового просмотра записей должна показаться привлекательной. (В любом случае не стоит волноваться. Эта абракадабра скоро не будет казаться вам такой уж странной.)
- **Гибкость поиска.** Нет необходимости искать записи строго в соответствии с порядком, в котором они были записаны (по фамилии пациента, например). Информационной системе можно приказать расположить записи, отсортированные в любом порядке: по фамилии, названию страховой компании, дате последнего визита и т.д.
- **Гибкость формата вывода.** После того как необходимые записи найдены, копировать записи вручную не нужно. Можно сделать запрос информационной системе на вывод нужного списка. Иногда достаточно просто распечатать информацию. В других случаях вам может понадобиться воспользоваться этими данными в другой программе. (Например, после получения списка пациентов, которые забыли сделать ежегодный профилактический осмотр, эти данные можно переслать в текстовый редактор и уже на основании этой информации распечатать приглашения этим пациентам посетить ваш кабинет.) Предположим, что вам нужна только итоговая информация, такая как счетчик выбранных записей. И это совсем не обязательно делать вручную. Информационная система сгенерирует такой отчет сама.
- **Одновременный многопользовательский доступ к записям.** Предположим, что сразу два человека хотят просмотреть одну запись. При бумажном способе ведения дел второй кандидат всегда вынужден ждать, пока первый закончит просмотр бумаг. СУБД позволяет получить доступ к одной и той же записи одновременно.
- **Удаленный доступ и передача записей в электронном виде.** Бумажная технология ведения дел требует от вас быть там, где находятся сами бу-

маги, в противном случае кто-то должен скопировать и переслать их вам. Электронный способ ведения записей позволяет производить удаленный доступ к записям и передавать их в электронном виде. Предположим, что ваша стоматологическая фирма состоит из подразделений, удаленных территориально, электронный способ ведения учета позволяет получить доступ к вашим записям из удаленных офисов. Нет необходимости посылать копии записей о пациентах курьером, даже если кто-либо не имеет в своем распоряжении базы данных, аналогичной вашей, но имеет электронную почту. Ему можно будет послать содержимое базы данных в электронном виде.

Тот, кто имел опыт работы с СУБД, знает об этих преимуществах и может уже подумать не только о банальном замещении бумажной картотеки. СУБД теперь используются для предоставления услуг, о которых еще недавно не могло быть и речи. Примером тому может служить использование многими организациями баз данных в совокупности с доступом к Internet.

Предположим, что ваша компания имеет базу данных учета материальных ценностей, которой пользуются менеджеры при поступлении запросов от потребителей о наличии и стоимости товаров. Это достаточно традиционное использование баз данных, однако, если ваша компания может создать свой Web-узел, в перечне сервисов которого потребителям предоставляется возможность самим узнать о наличии товара на складе и его цене, это позволит потребителям самим получать нужную им информацию о товарах и способах их поставки. При этом потребитель получает информацию немедленно, не слушая раздражающую музыку в телефонной трубке. Такой магазин может работать круглые сутки. Для всех посетителей вашего Web-узла это будет означать, что им придется сделать на один звонок меньше.

Но базам данных можно найти еще лучшее применение. Запросы на поиск товара, сделанные через ваш Web-узел, могут служить источником информации не только для тех, кто их делает, но и для вас самих. Вы сможете получить ответ на очень важный вопрос: "Что нужно потребителю?" А результаты выполнения запросов потребителей покажут, сможете ли вы удовлетворить их запросы. Если у вас нет того, что им нужно в данный момент, ваш бизнес может прогореть. Поэтому есть смысл хранить информацию о спросе на товары и о имеющемся ассортименте. Имея под руками такую информацию, вы сможете эффективно управлять запасами товаров на складе и тем самым удовлетворять спрос покупателей.

Еще одним применением баз данных на Web-страницах может служить реклама на баннерах. У меня есть причины не любить их больше всех, но факт остается фактом. Они являются популярным применением СУБД MySQL, который может быть использован здесь для хранения рекламы и поиска ее Web-сервером для отображения. Кроме того, СУБД MySQL может хранить данные о том, как часто происходили обращения к данному роду деятельности, об отслеживании рекламы, которая сработала, сколько раз она запрашивалась, с каких узлов был произведен доступ, и т.д.

Итак, как же работает СУБД MySQL? Лучший способ узнать это — самостоятельно с ней поработать.

Пример базы данных

В этом разделе описывается тестовая база данных, которая будет использоваться в книге в дальнейшем. Приведены исходные тексты тестовых программ, которые помогут глубже понять принципы работы СУБД MySQL. Примеры созданы на базе двух сценариев.

- **Сценарий секретаря организации.** Нам нужен какой-то более определенный термин, чем "организация". Поэтому давайте воспользуемся такими характеристиками, как группа людей, объединенных интересом к истории Соединенных Штатов Америки (назовем ее за отсутствием лучшего названия "Исторической Лигой США"). Члены этого общества подтверждают свое членство, периодически выплачивая взносы. Взносы покрывают такие накладные расходы организации, как публикация журнала "Хроника прошлого США". Организация имеет свой небольшой, не очень хорошо разработанный Web-узел. Он содержит только информацию о том, что представляет собой "Историческая Лига США" (в дальнейшем — "Историческая Лига"), каков штат и как можно стать ее членом.
- **Сценарий учета успеваемости учащихся.** В процессе обучения вы ведете учет результатов опросов и тестов, записываете оценки и присваиваете степени. После этого определяются окончательные оценки, которые представляются руководству школы вместе с данными о посещаемости.

Теперь давайте оценим более детально эти сценарии, исходя из следующих критериев.

- Вам предстоит решить, для чего вам нужна база данных, т.е. какие цели вы ставите перед собой.
- Необходимо четко определить ту информацию, которой вы хотите наполнить базу данных, т.е. какие данные должны отслеживаться.

Возможно, такая последовательность критериев может показаться странной. Ведь очевидно, что до того как делать выборку информации из базы данных, ее нужно наполнить данными. Но способ использования вами базы данных зависит от конкретных задач. А задачи больше связаны с назначением базы данных, чем с тем, что вы внесли в нее. Наверняка вы не будете попусту тратить время и силы на ввод информации в базу данных, если не имеете намерений использовать эту информацию в дальнейшем.

"Историческая Лига"

Отправная точка этого сценария: вы как секретарь этой организации и ведете список членов в текстовом редакторе. Этого достаточно для распечатки списка, но совершенно недостаточно при любом другом использовании информации. У вас есть три цели, которые перечислены ниже.

- Вам необходимо получить возможность выводить информацию в разных форматах, получая при этом только необходимые данные. Одна цель — получить возможность распечатывать список каждый год — традиция, которой придер-

живается "Историческая Лига". Можно придумать и другие применения информации, хранящейся в списке, например, получать список действительных членов "Исторической Лиги" для включения их имен в программу ежегодного банкета организации. Для выполнения этой задачи требуются различные наборы данных. В печатном списке используется все содержимое записи о члене "Исторической Лиги". Для программы банкета достаточно только получить имена и фамилии членов организации (это не так просто в случае работы с текстовым редактором).

- Необходимо производить поиск в членском списке, руководствуясь самыми разнообразными критериями. Например, необходимо определить, кому из членов нужно вскоре обновить членство. Другая задача, также требующая поиска, связана со списком ключевых слов, которые вы определяете для каждого члена. Эти ключевые слова очерчивают те периоды истории США, которые входят в круг научных интересов данного члена (например, Гражданская война в США, Великая Депрессия, гражданские права или биография Томаса Джефферсона). Члены "Исторической Лиги" иногда интересуются тем, чем интересуются другие их коллеги, чтобы найти единомышленников.
- Вы хотите разместить список членов на Web-узле "Исторической Лиги". Это будет полезно как для самих членов организации, так и для вас. Если вам удастся преобразовать список в Web-страницу с помощью автоматического процесса, интерактивный вариант списка всегда будет более актуальным, чем печатный. И если этот список еще будет снабжен функцией поиска, члены организации получат возможность просматривать информацию самостоятельно. Например, член "Исторической Лиги", который хочет узнать, кто из других членов интересуется историей Гражданской войны в США, сможет сделать это, не ожидая, пока это сделает для него секретарь. Да и вам не потребуется тратить на это свое время.

Я отлично понимаю, что база данных не является верхом совершенства. Поэтому я далек от того, чтобы провозгласить, что базы данных стимулируют созидательное мышление. Тем не менее, когда вы прекратите думать об информации как о чем-то, требующем усилий (как это бывает при работе с обычными документами, созданными в текстовом редакторе), а будете думать о том, что можно достаточно просто манипулировать данными (как это должно быть в случае с СУБД MySQL), это поможет вам быстрее создавать новые подходы к использованию или отображению информации.

- Информация может быть выведена из базы данных на Web-узел "Исторической Лиги". Это позволяет организовать ведение информации по-другому. Например, членам общества можно предоставить возможность самим редактировать информацию о себе в базе данных. Таким образом, полностью отпадает необходимость выполнять редактирование самостоятельно. Это позволит иметь в базе данных более актуальную информацию.
- В базе данных могут храниться адреса электронной почты членов организации. Ими можно воспользоваться для рассылки сообщений членам, которые не обновили свое членство. Эти сообщения могут содержать текущее со-

держимое их записей, просьбу обновить их, а также указания, как это сделать прямо на Web-узле.

- База данных может помочь вам сделать Web-узел более функциональным и полезным, даже без списка членов. "Историческая Лига" публикует журнал "Хроника прошлого США", в котором ведется детский раздел с викториной. Один из последних выпусков викторины был посвящен биографиям президентов США. Web-узел тоже может иметь детский раздел со своей викториной, работающий в режиме "on-line". Вероятно, что этот раздел может быть интерактивным, и информация для викторины будет заноситься прямо в базу данных. Таким образом, при генерации вопросов Web-сервер делает запрос базы данных случайным образом.

Отлично! С этого момента число пользователей базы данных позволит вам понять, что вы немного отклонились от предначертанной цели. После небольшого тайм-аута вернитесь с небес на землю и задайте себе пару вопросов, затрагивающих практические аспекты.

Не слишком ли много амбиций? Не слишком ли много требуется выполнить работы? Нет ничего проще, чем задумать что-либо и не сделать это. Я совсем не хочу сказать, что все это просто реализовать. Тем не менее к концу этой книги вы уже сможете выполнить все то, что здесь было запланировано. Просто надо помнить, что ничего нельзя сделать сразу. Следует разделить работу на отдельные этапы. И постепенно, шаг за шагом, их осуществлять.

- **Способна ли СУБД MySQL делать все это?** Нет. Например, СУБД MySQL не обладает непосредственной способностью работать с Internet. Но если СУБД MySQL сама не обладает такими возможностями, для этого есть программное обеспечение, дополняющее и расширяющее ее возможности.

  Для написания сценариев доступа к базам данных СУБД MySQL мы воспользуемся языком написания сценариев Perl и модулем DBI (database interface — интерфейс базы данных). Язык Perl обладает отличными возможностями и позволяет выводить данные в различных форматах. Например, с помощью Perl список можно выдавать в формате Rich Text Format (RTF), который воспринимается любым текстовым редактором.

  Мы также можем воспользоваться языком написания сценариев PHP. Язык PHP полностью адаптирован к написанию Web-приложений и позволяет работать с базами данных. Это позволит генерировать запросы СУБД MySQL прямо с Web-страниц и генерировать новые страницы, содержащие результаты запросов к базе данных. Этот язык хорошо адаптирован для работы с Web-сервером Apache (самый популярный Web-сервер), упрощающим выборку данных и отображение результатов выборки.

  СУБД MySQL хорошо интегрируется с этим программным обеспечением и позволяет гибко комбинировать его по своему усмотрению в соответствии с поставленными задачами. Это интегрированный компонент, обладающий высокой степенью интеграции "все в одном".

- **И наконец, самый больной вопрос — сколько все это стоит?** Бюджет организации довольно ограничен. Но как ни удивительно, это не будет вам ничего сто-

ить. Если вы знакомы с основными базами данных, имеющимися на рынке, то должны знать, что они достаточно дороги. В отличие от них, СУБД MySQL обычно распространяется бесплатно. (Для ознакомления с условиями лицензирования обратитесь к руководству по СУБД MySQL.) Бесплатно распространяется и другое программное обеспечение (Perl, DBI, PHP, Apache).

Выбор операционной системы ложится целиком и полностью на вас. Все программное обеспечение работает под управлением ОС UNIX (это название я использую как общее для ОС BSD UNIX, Linux, Mac OS X и т.д.) и Windows. Для ОС UNIX и Windows существуют исключения, такие как консольные и пакетные сценарии.

Теперь давайте рассмотрим другой сценарий использования баз данных.

Проект "Учет успеваемости"

Отправным пунктом этого сценария является учитель, на которого возложены обязанности учета успеваемости учащихся. У вас существует намерение заменить ручной учет, который проводился с помощью журнала, на электронное ведение дел с помощью СУБД MySQL. В этом случае вы хотите извлечь из базы данных то же, что и из вашего журнала успеваемости.

- По результатам викторин и тестов записываются баллы, полученные учащимися. Баллы тестов имеют такое обозначение, чтобы с первого взгляда на них можно было определить оценку (A, B, C, D и F).
- В конце каждого учебного периода подсчитывается суммарный балл каждого учащегося. Затем они сортируются, и на их основании выставляются оценки. Суммы могут содержать взвешенные коэффициенты, так как вы, вероятно, захотите акцентировать внимание на результатах тестов, а не на результатах викторин.
- По завершении учебного периода вы представляете оценку посещаемости школьному руководству.

Целью этого проекта является попытка избежать сортировки и суммирования оценок успеваемости и посещаемости вручную. Другими словами, СУБД MySQL вам нужна для сортировки оценок и вычислений, необходимых для получения суммарных баллов учащихся и числа пропусков занятий по завершении учебного периода. Для этого необходимы список учащихся, баллы, полученные за каждую викторину или тест, даты пропуска занятий учащимися.

Каким образом пример базы данных можно использовать в конкретном случае

Вас не интересует ни пример с "Исторической Лигой", ни пример учета успеваемости/посещаемости? Ответом может быть то, что эти примеры не являются самоцелью. Это только средство, с помощью которого демонстрируются возможности СУБД MySQL и соответствующего программного обеспечения.

Теперь рассмотрим, как примеры запросов к базе данных применимы к конкретным задачам, которые вы собираетесь решать. Предположим, что вы рабо-

таете в стоматологическом кабинете. Автор уже ссылался на этот пример. В этой книге вы не найдете никаких примеров, связанных со стоматологией, но вы встретите много запросов, которые смогут вас заинтересовать. Например, поиск членов "Исторической Лиги", которые должны возобновить свое членство, аналогичен определению пациентов, пропустивших свой очередной профилактический осмотр. И, действительно, оба запроса базируются на датах. Так, если вы знаете, как написать запрос на обновление членства, вы легко сможете создать запрос о недисциплинированных пациентах.

Основная терминология баз данных

Вы, наверное, почувствовали, что, пролистав достаточно много страниц этого фолианта, все еще не окунулись в жаргон и техническую терминологию. Я не сказал ни слова о том, как в действительности выглядит "база данных". Коротко были определены цели применения базы данных. Однако это возможно только на этапе проектирования базы данных. Этому вопросу и посвящен данный раздел. Здесь описаны термины, встречающиеся в этой книге. К счастью, концепция реляционных баз данных достаточно проста. И действительно, многие преимущества реляционных баз данных заложены в самой простоте базовых концепций.

Структурная терминология

СУБД MySQL классифицируется как реляционная система управления базами данных (RDBMS — relational database management system). Эта аббревиатура разбивается на части следующим образом.

- База данных (БД) (DB в RDBMS) — это совокупность информации, разбитой простым, регулярным образом.
  - Совокупность данных в базе данных объединена в таблицы.
  - Таблица состоит из строк и столбцов.
  - Строка таблицы является ее записью.
  - Записи содержат несколько единиц информации, каждый столбец таблицы содержит одну такую единицу.
- Система управления (СУ) (MS — management system) является программным обеспечением, позволяющим вставлять, выбирать, модифицировать и удалять записи.
- Слово "реляционная" обозначает популярную разновидность СУБД, в которых отслеживается "соответствие" записей в одной таблице на "соответствие" записей в другой таблице. Мощь реляционных СУБД заключается в их способности выбирать соответствующие данные из этих таблиц и создавать ответы на вопросы, которые нельзя получить только из одной такой таблицы.

Приведем пример того, как реляционная база данных объединяет данные в таблицы и связывает данные из одной таблицы с данными из другой таблицы. Предположим, что вы сопровождаете Web-узел, представляющий сервис размещения баннеров. Вы поддерживаете контакт с компаниями, которые хотят

разместить свою рекламу. Эта реклама будет появляться при каждом посещении вашего Web-узла. Всякий раз, когда посетитель вашего узла попадает на одну из ваших страниц, вы вместе с ней посылаете встроенную рекламу на броузер посетителя и получаете от рекламируемой компании небольшой гонорар. Для отображения этой информации воспользуемся тремя таблицами (рис. 1.1). Первая таблица, company, содержит столбцы с именами компаний, номером, адресом и телефонным номером. Вторая таблица, ad, содержит номера реклам, компаний, которым принадлежат эти рекламы, сумму, которую вы берете с них за один доступ к их рекламе. Третья таблица, hits, регистрирует количество обращений к рекламе и дату, когда это произошло.

Ответы на некоторые вопросы можно получить, пользуясь информацией из одной таблицы. Для определения количества компаний, с которыми у вас имеются контракты, достаточно посчитать строки в таблице company. Аналогично, для подсчета количества обращений за определенный период времени достаточно таблицы hit. Другие вопросы будут посложнее, и для получения ответов на них необходимо опросить не одну, а сразу несколько таблиц. Например, для определения, сколько раз 14 июля было сделано обращение ко всем рекламам компании Pickles, Inc., вам потребуется просмотреть три таблицы.

1. По названию компании (Pickles, Inc.) в таблице company определить номер компании (14).
2. По номеру компании найти соответствующие ему записи в таблице ad, содержащие номера рекламы. В нашем примере мы видим две такие записи — 48 и 101.
3. С помощью номера рекламы для всех найденных в таблице ad записей определить соответствующие записи в таблице hit, попадающие в нужный диапазон дат, а затем подсчитать количество совпадений. Найдены три совпадения для рекламы под номером 48 и два совпадения для рекламы под номером 101.

На первый взгляд, все это непросто. Но это лишь основа, на которой зиждется совершенство реляционных баз данных. Здесь сложность несколько иллюзорна, ибо каждый только что описанный шаг содержит нечто большее, чем простые операции выборки: вы связываете одну таблицу с другой с подбором значений из строк одной таблицы со строками другой таблицы. Эта простая операция может быть использована различными способами для получения ответов на следующие вопросы. Сколько различных реклам имеет каждая компания? Рекламы какой компании наиболее популярны? Сколько дохода приносит каждая реклама? Каков полный гонорар, полученный от каждой компании за текущий период?

Теперь вы достаточно знаете теорию реляционных баз данных, чтобы понять оставшийся материал, изложенный в этой книге. Здесь мы не будем углубляться в изучение третьей нормализованной формы, диаграмм "сущность-связь" и других высоконаучных проблем. Если вы хотите вникнуть в эти вопросы, что само по себе уже страшно, я могу предложить вам обратиться к трудам таких столпов науки о базах данных, как Дейт (*C.J. Date*) или Кодд (*E.F. Codd*.)

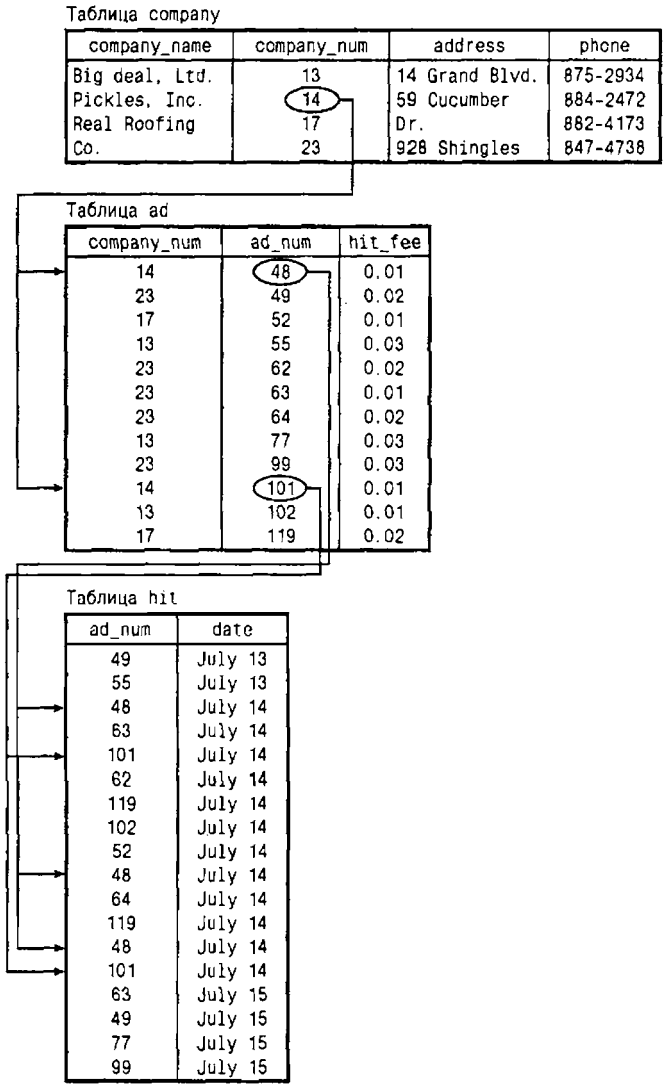


Рис. 1.1. Таблицы учета баннерной рекламы

Терминология языка запросов SQL

Для общения с СУБД MySQL применяется язык SQL (Stuctured Query Language – язык структурированных запросов). В настоящее время SQL является стандартом работы с базами данных, и все основные СУБД понимают его. SQL включает много разных типов операторов, разработанных для взаимодействия с базами данных.

Как и в случае с другими языками, SQL при первом знакомстве может показаться несколько странным. Например, создавая таблицу, необходимо указать СУБД MySQL, какую структуру она должна иметь. Вы и я можем представить себе

таблицу в виде диаграммы или рисунка, но СУБД этого не может. Таким образом, создать таблицу можно, только сказав СУБД MySQL что-то вроде следующего:

```
CREATE TABLE company
(
   company_name CHAR(30),
   company_num INT,
   address CHAR(30),
   phone CHAR(12)
);
```

Новичку такой оператор может показаться странным, но совсем не надо быть программистом, чтобы научиться эффективно применять его с SQL. Познакомившись с этим языком ближе, вы будете смотреть на оператор CREATE TABLE в другом свете: как на союзника, помогающего описывать информацию, а не как на колдовскую головоломку, состоящую из битов с байтами.

Терминология архитектуры СУБД MySQL

СУБД MySQL использует традиционную архитектуру клиент/сервер, поэтому, работая с СУБД MySQL, пользователь реально работает с двумя программами.

- Программа сервера базы данных, расположенная на компьютере, где хранится база данных. Она "прослушивает" запросы клиентов, поступающие по сети, и осуществляет доступ к содержимому базы данных для предоставления информации, которую запрашивают клиенты.
- Клиентская программа, которая осуществляет подключение к серверу и передает запросы на сервер.

Дистрибутив СУБД MySQL включает в себя несколько клиентских программ и сервер. Клиентские программы используются в соответствии с поставленными задачами. Одной из наиболее популярных и наиболее привлекательных клиентских программ является `mysql`. Это интерактивная программа, позволяющая создавать запросы и просматривать полученные результаты. Двумя административными программами являются `mysqldump`, которая позволяет делать дамп базы данных в файл и восстанавливать его обратно, и `mysqladmin`, позволяющая проверять состояние сервера и выполнять административные задачи наподобие отключения сервера. Кроме того, дистрибутив содержит другие клиентские программы.

Клиентские программы могут не подходить по своим функциональным возможностям для ваших целей, тогда вам может пригодиться библиотека для написания своей собственной программы-клиента СУБД MySQL. Эту библиотеку можно вызывать прямо из программ, написанных на языке C. В случае, если вы предпочитаете язык, отличающийся от C, доступно несколько других языков — Perl, PHP, Python, Java, C++.

Архитектура клиент/сервер СУБД MySQL имеет ряд преимуществ.

- Сервер обеспечивает контроль параллельного доступа, что не позволяет двум пользователям одновременно модифицировать одну и ту же запись. Все клиентские запросы проходят через сервер; таким образом, сервер знает, кто производит доступ к базе, что и когда он хочет сделать. При

одновременном доступе нескольких пользователей к одной и той же таблице базы данных им нет необходимости предварительно искать друг друга и договариваться между собой. Они просто делают свои запросы, а в остальном полностью полагаются на сервер.

- Нет необходимости регистрироваться на компьютере, где расположена база данных. СУБД MySQL может работать через Internet. Поэтому клиентская программа может быть запущена везде, где угодно, а клиент сам производит соединение с сервером через сеть. Расстояние не имеет значения, и к серверу можно подключиться из любой точки мира. Даже если сервер находится в Австралии, к нему можно подключиться с помощью переносного компьютера из Исландии. Значит ли это, что *любой человек* может подключиться к вашей базе данных через Internet? Конечно нет. СУБД MySQL снабжена гибкой системой защиты, которая позволяет иметь доступ к базе данных только тем, кто имеет на это право. Кроме того, этим пользователям должно быть разрешено делать только то, что они имеют право делать. Очевидно, что Салли из биллингового управления должна иметь право на чтение или модификацию записей, а Фил из отдела обслуживания — только на их просмотр. СУБД MySQL позволяет соответствующим образом определять их права.

В MySQL версии 4 есть другие варианты работы сервера. Вдобавок к обычному серверу `mysqld`, который используется в среде клиент/сервер, MySQL содержит сервер в виде библиотеки `libmysqld`, ссылки на которую вы можете поместить в программы, чтобы создать самостоятельные программы, работающие с MySQL. Это называется библиотекой встроенного сервера, поскольку он встраивается в автономные программы. Использование встроенного сервера отличается от варианта клиент/сервер, так как не требуется сеть. Таким образом, легче создать программы, которые могут распространяться сами по себе, с некоторыми поправками на их внешнюю рабочую среду. С другой стороны, этот метод должен быть использован только в ситуациях, когда такой программе единственной потребуется доступ к базам данных, управляемых сервером.

Различие между MySQL и `mysql`

Во избежание разночтений, хочу подчеркнуть, что MySQL относится ко всей СУБД MySQL, а "`mysql`" — это название конкретной клиентской программы. В разговорной речи это звучит одинаково, но в написании одно наименование пишется прописными буквами, а другое — строчными.

С одной стороны, MySQL произносится как "май-эс-кью-элл" (это известно из руководства по использованию MySQL). С другой стороны, SQL произносится как "сиквел". Это зависит от того, кого вы спрашиваете. Я не настаиваю ни на одном из вариантов. Произносите это слово как вам угодно, но будьте готовы к тому, что кто-нибудь вас поправит.

Учебный курс по СУБД MySQL

Теперь вы обладаете базовыми знаниями, которые необходимы вам на этом этапе. Приступим к работе с СУБД MySQL!

В этом разделе вы познакомитесь с СУБД MySQL. Для этого здесь приведен учебный курс по СУБД MySQL. По мере ознакомления с учебным курсом вы соз-

дадите базу данных с таблицами, а потом попробуете поработать с ней, добавляя, удаляя и модифицируя информацию, а также осуществляя ее поиск. Кроме того, в процессе работы с учебной базой данных вы узнаете следующее.

- **Основы диалекта языка SQL, который применяется в работе с СУБД MySQL.** (Для тех, кто уже знаком с языком SQL по опыту работы с другой СУБД, также будет полезно просмотреть этот учебный материал, чтобы получить понятие о различиях, имеющихся в данной версии языка).

- **Основные принципы работы с сервером СУБД MySQL с помощью клиентской программы `mysql`.** Эта программа получает SQL-запросы от пользователя, посылает их на сервер для выполнения и возвращает полученный результат пользователю. Клиент `mysql` работает на любой платформе, имеющей поддержку СУБД MySQL, и предоставляет наиболее прямые средства взаимодействия с сервером. Поэтому логичнее начать наш учебный курс с сервера. В некоторых примерах используются клиенты `mysqlimport` и `mysqlshow`.

Назовем учебную базу данных `sampdb`. Однако может потребоваться воспользоваться другим именем для обозначения базы данных. Вероятно, что кто-то уже использует имя `sampdb` для обозначения своей базы данных, поэтому ваш администратор предоставит вам другое имя. В любом случае вы можете свободно заменить имя базы данных `sampdb` другим именем.

Таблицам рекомендуем давать имена, аналогичные тем, которые приведены в примерах. Это допустимо, даже если аналогичную учебную базу данных уже создало несколько пользователей вашей системы. В СУБД MySQL не имеет никакого значения то, что несколько пользователей применяют одноименные таблицы. Таблицы хранятся в различных базах данных, и СУБД MySQL не позволяет им пересекаться.

Пример тестовой базы данных

Этот обучающий материал в определенной мере имеет отношение к файлам из примера тестовой базы данных (также известным как дистрибутив sampdb). Это файлы, содержащие запросы и данные, которые могут оказаться полезными при установке учебной базы данных. О том, как можно получить тестовую базу данных, читатель узнает в приложении А, "Получение и инсталляция программного обеспечения". После разархивирования примера будет автоматически создан каталог `sampdb`, содержащий все необходимые файлы. Я рекомендую сделать этот каталог текущим.

Предварительные требования

Для того чтобы выполнить примеры, приведенные в этой книге, должно быть удовлетворено несколько предварительных условий.

- У вас должно быть установлено программное обеспечение MySQL.
- Для того чтобы вы могли подключиться к серверу в СУБД MySQL, на вас должна быть заведена учетная запись.

- Наконец, вам понадобится база данных.

Требуемое программное обеспечение содержит клиента и сервер MySQL. Клиентские программы должны размещаться на компьютере, на котором вы будете работать. Сервер может размещаться на вашей машине, хотя это не обязательно. Пока у вас есть доступ к нему, вы можете подключаться откуда угодно. Для получения MySQL обратитесь к приложению А, "Получение и инсталляция программного обеспечения". Если у вас доступ к сети через поставщика Internet-услуг (ISP), узнайте, предоставляет ли он MySQL-хостинг. В противном случае обратитесь к приложению И, "Провайдеры услуг Internet", чтобы узнать, как выбрать подходящего поставщика.

После этого необходимо получить право на создание своей тестовой базы данных и ее таблиц. (Если у вас уже есть учетная запись MySQL, вы можете использовать ее, хотя, возможно, вы захотите создать отдельную запись для работы с материалами, представленными в этой книге.)

Чтобы настроить учетную запись MySQL, с помощью которой вы будете подключаться к серверу, необходимо подключиться к серверу. Обычно это делается привилегированным пользователем на сервере, на котором установлен сервер MySQL, и вам предоставляются права на создание новой учетной записи. Если вы установили MySQL на собственном компьютере и сервер работает, вы можете подключиться к нему и создать новую учетную запись администратора с именем `sampadm` и паролем `secret` (по всей книге можно изменить имя и пароль на нужные).

```
% mysql -p -u root
Enter password: ******
mysql> GRANT ALL ON sampdb.* TO 'sampadm'@'localhost' IDENTIFIED BY
'secret';
```

Команда `mysql` содержит ключ `-p` для запроса пароля привилегированного пользователя MySQL. Введите пароль там, где в примере вы видите ******. (Я полагаю, что вы уже указали пароль привилегированного пользователя и знаете его. Если же вы еще не указали его, нажмите клавишу <Enter> при запросе "Enter password:". Однако отсутствие пароля небезопасно, и вы должны указать его как можно скорей.)

Оператор GRANT действителен только при входе с машины, на которой работает сервер. Он позволяет подключиться к серверу с именем `sampadm` и паролем `secret` и предоставляет полный доступ к базе данных `sampdb`. Но он не создает базу данных. К этому мы вернемся позже.

Если вы не планируете подключаться с компьютера, на котором работает сервер, измените `localhost` на имя машины, с которой вы будете работать. Например, если вы будете подключаться к серверу с адреса `asp.snake.net`, утверждение GRANT должно выглядеть следующим образом:

```
mysql> GRANT ALL ON sampdb.* TO 'sampadm'@'asp.snake.net' IDENTIFIED
BY 'secret';
```

Если у вас нет контроля над сервером, попросите администратора MySQL создать для вас учетную запись. Затем измените имя, пароль и имя базы данных, которые назначил администратор, на `sampadm`, `secret` и `sampdb`. Дополнительную информацию об утверждении GRANT, настройке учетных записей MySQL и изменении паролей можно найти в главе 11, "Общее администрирование MySQL".

Установка и завершение связи с сервером

Для подключения к серверу вызовите программу `mysql` из своей оболочки (т.е. из подсказки ОС UNIX или из DOS-консоли под управлением ОС Windows). Вот эта команда:

```
% mysql options
```

Для отображения подсказки операционной системы воспользуемся обозначением `'%'`. Это одна из стандартных подсказок ОС UNIX, другой такой подсказкой является подсказка `'$'`. В ОС Windows подсказка будет иметь вид `'C:\>'`.

Часть параметров командной строки `mysql` может быть пустой, но, вероятнее всего, эта строка будет иметь вид

```
% mysql -h host_name -p -u user_name
```

Вряд ли вам потребуется указывать все ключи при запуске клиентской программы `mysql`, но, возможно, вам придется изменить хотя бы имя и пароль.

- `-h host_name` (альтернативная форма: `- -host=host_name`).

  Указывает имя серверного узла, к которому вы хотите подключиться. Если сервер работает на том же компьютере, что и клиент, этот параметр можно смело опустить.

- `-u user_name` (альтернативная форма: `- -user= user_name`).

  Указывает имя пользователя, зарегистрированное в СУБД MySQL. Если имена пользователей ОС UNIX и СУБД MySQL совпадают, этот ключ тоже можно опустить. При этом клиентская программа `mysql` будет использовать ваше регистрационное имя в качестве имени пользователя СУБД MySQL.

  При работе в Windows, имя пользователя по умолчанию будет ODBC. Возможно, это вам не очень пригодится. Существует возможность указать свое имя в командной строке или установить свою среду по умолчанию, указав имя пользователя `sampadm` в команде `set`:

  ```
  C:\> set USER=sampadm
  ```

  Если вы пропишете эту команду в файл AUTOEXEC.BAT, она будет срабатывать при каждой загрузке и вам не придется задавать ее в запросе.

- `–p` (альтернативная форма: `- -password`).

  Этот параметр подсказывает СУБД MySQL, что нужно запросить ваш пароль. Например:

  ```
  % mysql –h host_name –u user_name –p
  Enter password:
  ```

  Когда пользователь получает подсказку `Enter password:`, он должен вводить пароль. (Пароль традиционно не выводится на экран на тот случай, если кто-то заглядывает через ваше плечо.) Отметим, что ваш пароль в СУБД MySQL не обязательно совпадает с паролем в ОС UNIX или ОС Windows.

  При полном отсутствии ключа `-p` `mysql` решает, что пароль вам не нужен и не выводит подсказки.

  Также можно указать пароль прямо в командной строке как `-pyour_pass` (альтернативная форма: `-password=your_pass`). Но из соображений

безопасности лучше этого не делать, хотя бы потому, что пароль будет виден окружающим. Если вы все же решили указать пароль в командной строке, учтите, что между ключом -p и паролем нет пробела. Это часто является причиной путаницы, так как ключ -p отличается от ключей -h и -u, после которых всегда стоит пробел.

Например, предположим, что имя пользователя и пароль в СУБД MySQL соответственно sampadmin и secret и что пользователь хочет подключиться к серверу, запущенному на том же компьютере, на котором он работает. Для этого необходимо задать следующую команду mysql:

```
% mysql -p -u sampadm
Enter password: ******
```

После ввода команды mysql ответит подсказкой Enter password:. Пользователь вводит пароль, который отображается по мере ввода ****** (здесь был введен пароль secret).

Если пароль правильный, mysql ответит подсказкой mysql>. Это свидетельствует о том, что СУБД MySQL ожидает ввода новых запросов. В целом последовательность команд для подключения к серверу СУБД выглядит следующим образом:

```
% mysql -p -u sampadm
Enter password: ******
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 7575 to server version: 4.0.4-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

Для подключения к серверу, работающему на каком-либо другом компьютере, дополнительно требуется с помощью ключа –h указать имя узла. Предположим, что узел имеет имя cobra.snake.net, тогда команда будет иметь вид

```
% mysql -h cobra.snake.net -p -u sampadm
```

В большинстве из приведенных здесь примеров демонстрируется применение ключей –h, –u и –p. Это делается с одной целью – научить вас правильно использовать эти ключи.

Установленное соединение с сервером можно прервать в любой момент командой QUIT:

```
mysql> QUIT
Bye
```

Кроме того, завершить сеанс можно с помощью комбинации клавиш <Ctrl+D> (по крайней мере в ОС UNIX).

Первоначально новичок может подумать, что система безопасности СУБД MySQL будет только раздражать и затруднять работу с СУБД. (Для создания базы данных и получения доступа к ней необходимо получить разрешение и при каждом подключении к серверу вводить свое имя и пароль.) Однако после того, как вы начнете создавать и использовать свои собственные записи, ситуация кардинально изменится. Тогда вы уже сможете оценить то, что СУБД MySQL не позволяет другим пользователям просматривать (или, что еще хуже, уничтожать!) ваши данные.

Существуют способы настроить учетные записи таким образом, что вам не придется каждый раз вводить параметры соединения. Они обсуждаются в раз-

делс "Как работать с mysql" в этой главс. Самый простой способ упростить процесс соединения — сохранить настройки в отдельном конфигурационном файле. Вы можете перейти к указанному выше разделу, чтобы узнать, как создать такой файл.

Ввод запросов

Запросы к базе данных можно вводить только после подключения к серверу. В этом разделе рассказывается о работе с клиентской программой `mysql`.

Для ввода запроса в `mysql` достаточно его напечатать, ввести в конце точку с запятой и нажать клавишу <Enter>. После ввода запроса `mysql` передает его серверу на выполнение. После обработки запроса сервер передает результат клиенту `mysql`. Клиент отображает полученный результат.

Вот пример простейшего запроса текущих даты и времени:

```
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2002-09-01 13:54:24 |
+---------------------+
1 row in set (0.00 sec)
```

`mysql` отображает результат запроса и строку, которая показывает, из скольких строк состоит результат и сколько времени затрачено на выполнение запроса. В следующих примерах я не буду показывать эту строку. Программа `mysql` воспринимает точку с запятой как окончание запроса, поэтому по мере необходимости запросы могут занимать несколько строк:

```
mysql> SELECT NOW(),
    -> USER(),
    -> VERSION()
    -> ;
+---------------------+-------------------+----------------+
| NOW()               | USER()            |      VERSION() |
+---------------------+-------------------+----------------+
| 2002-09-01 13:54:37 | sampadm@localhost | 4.0.4-beta-log |
+---------------------+-------------------+----------------+
```

Обратите внимание, что подсказка меняется с `mysql>` на `->` после ввода первой строки запроса. Это означает, что `mysql` ожидает продолжения ввода запроса. Это очень важный элемент обратной связи, показывающий пользователю, что даже если вы забыли ввести точку с запятой, `mysql` ожидает ее ввода и показывает, что ввод запроса еще не завершен. Другими словами, пока пользователь сидит и удивляется, почему так долго нет ответа от `mysql`, клиент `mysql`, в свою очередь, ожидает завершения ввода запроса! (В `mysql` также существуют другие запросы. Они описаны в приложении Д, "Программы MySQL".)

В большинстве случаев регистр, в котором вводится команда, не имеет особого значения. Запросы, приведенные ниже, эквивалентны.

```
SELECT USER();
select user();
SeLeCt UsEr();
```

Примеры, приведенные в этой книге, выполнены в соответствии со следующим правилом: прописными буквами вводятся все ключевые слова и функции языка SQL; для имен баз данных, таблиц и столбцов используются строчные буквы.

При вызове функции в запросе недопустим пробел между именем функции и последующими скобками.

```
mysql> SELECT NOW ();
ERROR 1064: You have an error in your SQL syntax near '()' at line 1
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2002-09-01 13:56:36 |
+---------------------+
```

Эти два запроса выглядят аналогично, но во втором сообщается об ошибке, поскольку за именем функции нет скобок.

Отменить запрос можно также, напечатав "\g":

```
mysql> SELECT NOW()\g
+---------------------+
| NOW()               |
+---------------------+
| 2002-09-01 13:56:47 |
+---------------------+
```

Вы также можете использовать параметр '\G', который отображает результат в вертикальном формате:

```
mysql> SELECT NOW(), USER(), VERSION()\G
*************************** 1. row ***************************
    NOW(): 2002-09-01 13:56:58
   USER(): sampadm@localhost
VERSION(): 4.0.4-beta-log
```

Для запроса, который генерирует короткие строки, параметр '\G' практически бесполезен, но если строки настолько длинны, что засоряют экран, '\G' делает вывод более читабельным.

Если вы начали вводить запрос, а затем передумали, введите '\c' для его отмены.

```
mysql> SELECT NOW(),
    -> VERSION(),
    -> \c
mysql>
```

Обратите внимание, что подсказка опять изменилась на mysql>. Это свидетельствует о том, что mysql ожидает ввода нового запроса.

Запросы можно сохранять в файле и затем запускать их на выполнение прямо из файла, не создавая вновь. Для этого можно воспользоваться функциями перенаправления ввода-вывода. Например, если запрос сохранен в файле my_file.sql, его можно выполнить с помощью команды.

```
% mysql < my_file.sql
```

Таким образом можно вызвать любой файл. Для того чтобы показать, что файл содержит операторы SQL, был указан суффикс .sql.

Работа mysql более детально будет описана в разделе "Дополнение таблиц". Таким образом, будут добавлены данные в базу данных sampdb. Действительно, намного удобнее заполнить таблицу с помощью операторов INSERT, считываемых из файла, чем вводить каждый оператор по отдельности.

В оставшейся части этого раздела будут продемонстрированы запросы, работу которых вы сможете повторить затем сами. Их можно найти по подсказке 'mysql>' перед запросом и завершающей точке с запятой. Как правило, эти примеры сопровождаются полученным результатом запроса. Вы сможете ввести запросы сами и получить аналогичный ответ. Запросы, показанные без подсказки и завершающей точки с запятой, несут смысловую нагрузку просто как разъяснение материала, и их выполнять нет смысла. (При желании их можно запустить на выполнение, не забывая при этом завершать оператор точкой с запятой.)

Когда требуется точка с запятой

В этой книге большинство запросов заканчивается точкой с запятой, что удобно для обозначения завершения каждого запроса (особенно в примерах, сразу содержащих несколько операторов). Она также указывает на способ ввода запросов программе mysql. Но точка с запятой — это не часть SQL-синтаксиса для операторов. Так, когда вы делаете запрос в другом контексте, например из сценария Perl или PHP, вы не должны использовать точку с запятой. Если вы этого не сделаете, вполне вероятно появление ошибки.

Создание базы данных

Начнем с процедуры создания базы данных sampdb, ее таблиц, заполнения таблиц и выполнения простейших запросов по этим данным.

Работа с базой данных предполагает несколько этапов.

1. Создание (инициализация) базы данных.
2. Создание таблиц в базе данных.
3. Взаимодействие с таблицами посредством операций вставки, выборки, модификации или удаления данных.

Выборка существующих данных фактически является самой частой операцией при работе с базами данных. В числе других наиболее часто используемых операций над таблицами баз данных являются вставка новых данных, их модификация и удаление. Далее частота использования операций располагается следующим образом: операция создания таблиц и операция создания базы данных.

Начнем с самого начала. Поэтому нашей первой операцией будет редко выполняемая операция — создание базы данных. Затем мы перейдем к созданию таблиц, загрузке начальных данных, а уже потом — к наиболее популярной в базах данных операции — выборке.

Для создания базы данных необходимо подключиться к серверу. Для этого воспользуемся клиентской программой mysql. После этого с помощью команды CREATE DATABASE sampdb; зададим имя базы данных.

```
mysql> CREATE DATABASE sampdb;
```

Очевидно, что, перед тем как приступить к созданию различных таблиц, необходимо создать саму базу данных, которая будет включать их в себя.

Каков механизм создания базы данных? Делает ли создание базы данных ее текущей базой данных? Нет. Это можно увидеть по результату запроса.

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| ...........|
+------------+
```

Для перевода базы данных sampdb в статус текущей необходимо ввести оператор USE.

```
mysql> USE sampdb
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| sampdb ....|
+------------+
```

В качестве другого способа активизации базы данных можно назвать следующий: достаточно указать имя базы данных в командной строке вызова клиента.

```
% mysql sampdb
```

Это обычный способ активизации базы данных. Перед именем базы данных можно задать другие инициализирующие параметры. Например, следующие две команды позволят подключиться к базе данных sampdb на локальном узле и на узле cobra.snake.net:

```
% mysql -p -u sampadm sampdb
% mysql -h cobra.snake.net -p -u sampadm sampdb
```

Во всех последующих примерах предполагается, что при запуске mysql в командной строке активизируется база данных sampdb.

Создание таблиц

В этом разделе вы узнаете, как создавать таблицы, составляющие базу данных sampdb. Сначала рассмотрим таблицы из проекта "Историческая Лига". Затем наступит очередь таблиц из проекта "Учет успеваемости". Сейчас мы находимся в той точке повествования, с которой во многих книгах начинаются рассуждения об анализе и о проектировании, диаграммах "сущность-связь", процедурах нормализации. В такой последовательности изложения есть рациональное зерно, но я предлагаю прежде всего задуматься над тем, как будет выглядеть наша база данных: что будут содержать таблицы и как отображать наши данные.

Есть масса вариантов представления данных. В зависимости от требований, предъявляемых к приложениям, и от предназначения данных, одни и те же данные можно представить различными способами.

Таблицы проекта "Историческая Лига"

Набор таблиц для проекта "Историческая Лига" достаточно прост.

- **Таблица** president. Содержит данные о президентах США. Это нужно для викторины в режиме on-line на Web-узле (интерактивный аналог напечатанной викторины, которая появлятся в детском разделе газеты, издаваемой "Исторической Лигой").

- **Таблица** `member`. Содержит текущую информацию о каждом члене "Исторической Лиги". Предназначена для печатной и интерактивной версии списка членов, а также для автоматической рассылки напоминаний о продлении членства.

Таблица `president`

Таблица `president` проще, поэтому с нее и начнем. Она содержит описание основных биографических данных из жизни всех президентов США.

- **Имя.** Имена можно хранить различными способами. Например, для хранения имени и фамилии можно воспользоваться одним столбцом, а можно создать отдельные столбцы. Конечно же, проще воспользоваться первым методом, но это создаст следующие проблемы в будущем.
  - Если в такой столбец сначала вводится имя, а затем фамилия, то сортировка по фамилии будет затруднена.
  - Если ввод производится в обратном порядке, то отображение имени, а затем фамилии будет несколько затруднено.
  - Выборка по именам будет затруднена. Например, при поиске конкретной фамилии можно использовать шаблон. Это менее эффективно и значительно медленнее, чем производить поиск непосредственно по фамилии.

  В таблице `president` для обхода этих ограничений имя и фамилию нужно будет хранить в отдельных столбцах.

  Столбец имени также будет содержать отчество или инициалы. Это не затруднит работу с этим столбцом, так как сортировать по отчеству нет необходимости. Отображение будет иметь вполне приемлемый вид, независимо от формата вывода, "Bush, George W." или "George W. Bush,".

  Существует еще одно небольшое затруднение. Всего один президент (Jimmy Carter) имеет окончание "Jr." в конце имени. Что это значит? В зависимости от формата отображения имя президента будет иметь вид "James E. Carter, Jr." или "Carter, James E., Jr.". Окончание "Jr." не ассоциируется ни с именем, ни с фамилией. Поэтому для хранения его создадим еще одно поле. Вот вам яркий пример того, как всего одно значение может создать проблемы при проектировании структуры базы данных. Отсюда видно, насколько важно иметь как можно более полную информацию о данных, с которыми придется работать на этапе создания базы данных. Но СУБД MySQL позволяет менять структуру баз данных во время ее использования. Это не беда, но этого по возможности нужно избегать.
- **Место рождения (город и штат).** Аналогично имени и фамилии, эти данные можно хранить как вместе, так и отдельно. Проще использовать один столбец. Но и в этом случае раздельное хранение позволит вам делать многие вещи, которые в противном случае будут существенно затруднены. Например, будет проще сделать выборку по президентам, родившимся в каком-то одном штате, если город и штат хранятся раздельно.
- **Даты рождения и смерти.** Единственной проблемой в этом случае является то, что нет даты смерти, если президент еще жив. СУБД MySQL име-

ет механизм присвоения пустого значения NULL, означающее, что значения нет. Это значение будем использовать для обозначения "в настоящее время жив".

Таблица member

Таблица member предназначена для фиксации членства в "Исторической Лиге". Она аналогична таблице president в том смысле, что в ней хранится вся основная описательная информация одного человека. При этом каждая запись таблицы member хранит больше столбцов.

- **Имя.** В этой таблице мы будем хранить имена подобно тому, как это сделано в таблице president в трех отдельных столбцах: фамилия, имя (и при наличии – отчество), суффикс.
- **Идентификационный номер.** Это уникальное значение, присваиваемое каждому новому члену организации. "Историческая Лига" никогда раньше не использовала идентификационных номеров для регистрации своих членов. Теперь это поможет систематизировать записи. (Я предвижу, как вы в дальнейшем будете расширять использование СУБД MySQL. Это произойдет тогда, когда вы начнете пользоваться информацией, хранящейся в базе данных. Когда это произойдет, вы увидите, что для ссылки на имена членов из других вновь созданных таблиц будет удобнее использовать идентификационные номера, хранящиеся в таблице member.)
- **Срок завершения полномочий.** Члены организации должны периодически возобновлять свое членство. В некоторых приложениях может представлять интерес дата самого последнего возобновления членства. Но это не имеет отношения к "Исторической Лиге", где членство может быть возобновлено на определенное количество лет (обычно один, два, три года или пять лет), и дата последнего возобновления не подскажет вам, когда было последнее возобновление. Кроме того, "Историческая Лига" имеет институт так называемого пожизненного членства. Пожизненное членство можно отобразить какой-то достаточно дальней датой, но очевидно, что значение NULL больше подойдет для этой цели. Значение "нет значения" больше соответствует "никогда не завершается".
- **Адрес электронной почты.** Это облегчит общение между членами "Исторической Лиги", имеющими электронную почту. Для секретаря это упростит рассылку напоминаний о возобновлении членства. Отпадет необходимость ходить на почту, да и обойдется это дешевле. Электронной почтой можно воспользоваться и для того, чтобы разослать членам общества их учетные записи с просьбой обновить их в случае, если появились какие-либо изменения в адресе, фамилии и т.д.
- **Почтовый адрес.** Эти данные необходимы для тех членов организации, у которых нет электронной почты (или которые не отвечают на письма). Название улицы, города, почтового индекса будем хранить в отдельных столбцах. Я предполагаю, что все члены "Исторической Лиги" живут в США. Для международных организаций это будет упрощением, но для нашей задачи это вполне подходит. В случае хранения адресов из разных

стран читатель обязательно увязнет в проблемах хранения адреса в различных форматах. Например, применение почтового индекса не является международным стандартом, многие страны делятся на провинции и области, а не на штаты.

- **Телефонный номер.** Очень полезен для поддержания контактов с членами "Исторической Лиги".
- **Ключевые слова для определения области интересов.** Без сомнения, областью научных интересов всех членов "Исторической Лиги" является история США, но вероятно, что каждый из них имеет свои специфические интересы. Этот столбец предназначен для хранения соответствующей информации. Он будет полезен для поиска по интересам.

Создание таблиц

Теперь можно приступить к созданию таблиц базы данных "Историческая Лига". Воспользуемся для этого оператором CREATE TABLE, который имеет следующий синтаксис:

```
CREATE TABLE tbl_name (column_specs);
```

где атрибут *tbl_name* содержит имя таблицы, а *column_specs* — спецификацию столбцов и индексов в таблице (если они имеются). Индексы нужны для ускорения просмотра таблицы. О них мы поговорим позже, в главе 4, "Оптимизация запросов".

Оператор CREATE TABLE, предназначенный для создания таблицы president, имеет следующий вид:

```
CREATE TABLE president
(
  last_name VARCHAR(15) NOT NULL,
  first_name VARCHAR(15) NOT NULL,
  suffix VARCHAR(5) NULL,
  city VARCHAR(20) NOT NULL,
  state VARCHAR(2) NOT NULL,
  birth DATE NOT NULL,
  death DATE NULL
);
```

Но перед этим нужно запустить mysql, используя команду

```
% mysql sampdb
```

Затем введите оператор CREATE TABLE, приведенный выше (не забудьте ввести точку с запятой в конце оператора, в противном случае mysql не будет знать, где заканчивается оператор).

Для создания таблицы president с уже созданным описанием используйте файл create_president.sql из дистрибутива sampdb. Файл находится в папке sampdb, которая создается после распаковки дистрибутива.

Перейдите в эту папку и запустите команду

```
% mysql sampdb < create_president.sql
```

Во время каждого запуска mysql в командной строке перед именем базы данных указывайте необходимые параметры подключения (имя узла, имя пользователя и пароль).

Спецификация столбца в операторе CREATE TABLE состоит из имени столбца, типа (вид значений, которые может содержать столбец) и, возможно, некоторых атрибутов столбца.

В таблице president используются два типа столбцов: VARCHAR и DATE.

Тип VARCHAR(n) подразумевает, что столбец содержит символьные (строковые) данные переменной длины с максимальной длиной *n* каждый. Значение *n* выбирается в зависимости от того, какой максимальной длины ожидается значение столбца. Так, state объявлен как VARCHAR(2). Этого вполне достаточно, если для хранения названия штата будет использоваться его традиционная двухбуквенная аббревиатура. Другие строковые столбцы должны быть длиннее, чтобы вместить более длинные названия.

Еще одним типом данных, которым мы здесь воспользовались, является тип DATE. Вас может удивить то, что формат хранения данных начинается с года. Стандарт формата даты 'CCYY-MM-DD', где CC, YY, MM, и DD представляют соответственно столетие, год, месяц и дату (это стандарт представления данных ANSI SQL; также известен как формат ISO 8601). Например, дата 18 июля 2002 в MySQL выглядит как '2002-07-18', но не как '07-18-2002' или '18-07-2002'.

Единственным атрибутом столбцов, которым мы воспользуемся в таблице president, будет атрибут NULL (значения данного столбца могут быть пустыми) и NOT NULL (значения данного столбца не могут быть пустыми). Большинство столбцов имеют атрибут NOT NULL. Это означает, что они всегда должны быть заполнены. Только два столбца могут иметь пустые значения — это suffix (суффиксы большей частью в именах отсутствуют) и death (некоторые президенты живы, и дата смерти отсутствует).

Оператор CREATE TABLE, предназначенный для создания таблицы member, имеет такой вид:

```
CREATE TABLE member
(
  member_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (member_id),
  last_name VARCHAR(20) NOT NULL,
  first_name VARCHAR(20) NOT NULL,
  suffix VARCHAR(5) NULL,
  expiration DATE NULL DEFAULT '0000-00-00',
  email VARCHAR(100) NULL,
  street VARCHAR(50) NULL,
  city VARCHAR(50) NULL,
  state VARCHAR(2) NULL,
  zip VARCHAR(10) NULL,
  phone VARCHAR(20) NULL,
  interests VARCHAR(255) NULL
);
```

Введите этот оператор из mysql или запустите из оболочки следующую команду:

```
% mysql sampdb < create_member.sql
```

С точки зрения типов столбцов таблица member не представляет особого интереса: все столбцы, за исключением одного, заданы как строки переменной длины. Одно исключение — столбцы member_id и expiration, которые соответственно содержат порядковые номера и даты.

В столбце member_id указываются уникальные числа, чтобы избежать путаницы с учетными записями.

Столбец AUTO_INCREMENT нужен для автоматического генерирования кода учетной записи при добавлении новой учетной записи. Несмотря на то что он просто содержит числа, его объявление состоит из нескольких частей.

- INT указывает на то, что число содержит целые числа (без дробной части).
- UNSIGNED запрещает отрицательные числа.
- NOT NULL указывает на то, что столбец не может быть пустым. (Это значит, что никакой член не может быть без идентификационного номера.)
- AUTO_INCREMENT является специальным атрибутом. Он указывает, что столбец содержит порядковые номера. Механизм AUTO_INCREMENT работает следующим образом: если отсутствует значение столбца member_id (или стоит параметр NULL), когда вы создаете новую учетную запись, MySQL автоматически генерирует порядковый номер и присваивает его столбцу. Таким образом, будет легче присваивать новые номера, так как MySQL сделает это за вас.

PRIMARY KEY указывает на то, что столбец member_id проиндексирован для быстрого поиска, и на то, что каждое значение в столбце должно быть уникально. Последнее свойство требуется для значений идентификационных номеров пользователей, так как предотвращает повторное использование номера. Кроме того, MySQL требует, чтобы у каждого столбца AUTO_INCREMENT был своего рода уникальный индекс, поскольку описание таблицы без него считается недействительным.

Если вы не понимаете назначения AUTO_INCREMENT и PRIMARY KEY, просто рассматривайте их как возможность генерировать идентификационный номер для каждой записи. Неважно, какие значения, лишь бы они были уникальными. (Когда вы будете готовы узнать больше о том, как использовать столбцы AUTO_INCREMENT, обратитесь к главе 2, "Работа с данными в MySQL".)

Столбец expiration имеет тип данных DATE. По умолчанию он имеет значение "0000-00-00", не являющееся значением NULL, которое означает, что не была введена дата. Это объясняется тем, что столбец expiration может не иметь значения, что указывает на тот факт, что член организации имеет пожизненное членство. Однако значением по умолчанию столбца, который может иметь пустое значение, есть пустое значение при условии, что другое значение по умолчанию не определено. В этом случае это нежелательно. Возникает ситуация, когда при создании записи для нового члена не вводится дата истечения срока полномочий и член получает пожизненное членство! Воспользовавшись значением по умолчанию "0000-00-00", мы решаем эту проблему. Это также позволит проводить периодическую проверку на правильность ввода даты.

Давайте убедимся, что нам удалось создать именно такую таблицу, какую мы планировали создать.

```
mysql> DESCRIBE president;
+------------+-------------+------+-----+------------+-------+
| Field      | Type        | Null | Key | Default    | Extra |
+------------+-------------+------+-----+------------+-------+
| last_name  | varchar(15) |      |     |            |       |
| first_name | varchar(15) |      |     |            |       |
| suffix     | varchar(5)  | YES  |     | NULL       |       |
```

```
| city        | varchar(20) |      |     |                |       |
| state       | char(2)     |      |     |                |       |
| birth       | date        |      |     | 0000-00-00     |       |
| death       | date        | YES  |     | NULL           |       |
+-------------+-------------+------+-----+----------------+-------+
```

В некоторых версиях MySQL результаты работы оператора DESCRIBE содержат информацию о правах доступа. Здесь они не приводятся, так как они слишком длинны.

Все вроде бы нормально, за исключением одного: столбец state имеет тип CHAR(2). Это странно. Разве он не был объявлен как VARCHAR(2)? Да, был, но СУБД MySQL по умолчанию изменила тип с VARCHAR на CHAR. Причина – проблема оптимизации области хранения данных, которой мы здесь еще не касались. Более подробно это обсуждается в главе 3, "Синтаксис и использование языка SQL". А на этом этапе разница между двумя типами не играет существенной роли.

Важным является то, что столбец содержит два символьных значения.

Введя оператор DESCRIBE member, вы увидите аналогичную информацию для таблицы member.

Оператор DESCRIBE полезен в тех случаях, когда пользователь забыл имя столбца или необходимо уточнить тип данных, размер столбца и т. п. Это также полезно для того, чтобы узнать порядок, в котором СУБД MySQL хранит столбцы в таблице. Знать это очень важно при добавлении строк в таблицу и загрузке данных. Для операторов INSERT и LOAD DATA необходимо полное соответствие вводимых данных со структурой таблицы.

Информацию, создаваемую оператором DESCRIBE, можно просмотреть несколькими способами.

Оператор DESCRIBE можно сократить до DESC или ввести оператор EXPLAIN или SHOW.

Эти операторы эквивалентны:

```
DESCRIBE president;
DESC president;
EXPLAIN president;
SHOW COLUMNS FROM president;
SHOW FIELDS FROM president;
```

Кроме того, эти операторы также позволяют ограничить вывод определенных столбцов. Например, вы можете в конце оператора SHOW добавить параметр LIKE, чтобы отобразить информацию только для столбцов, которые соответствуют шаблону:

```
mysql> SHOW COLUMNS FROM president LIKE '%name';
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| last_name  | varchar(15) |      |     |         |       |
| first_name | varchar(15) |      |     |         |       |
+------------+-------------+------+-----+---------+-------+
```

Символ '%' *используется как групповой символ, который описан в разделе* "Соответствие шаблону". Похожие ограничения могут использоваться вместе с операторами DESCRIBE и EXPLAIN. Точный синтаксис приведен в приложении Г, "Синтаксис SQL".

Оператор SHOW имеет также другие формы, которые могут пригодиться при получении разных типов информации от MySQL. SHOW TABLES выводит таблицы в текущей базе данных, так что вместе с двумя таблицами, которые мы создали в базе данных sampdb, вывод выглядит так:

```
mysql> SHOW TABLES;
+------------------+
| Tables_in_sampdb |
+------------------+
| member           |
| president        |
+------------------+
```

Оператор SHOW DATABASES отображает таблицы, которые управляются сервером, к которому вы подключены:

```
mysql> SHOW DATABASES;
+-----------+
| Database  |
+-----------+
| menagerie |
| mysql     |
| sampdb    |
| test      |
+-----------+
```

На каждом сервере список баз данных может варьироваться, но две из них должны присутствовать всегда. Это sampdb и mysql. Последняя база данных хранит таблицы, содержащие привилегии и права доступа.

Программа mysqlshow предоставляет интерфейс командной строки для тех же видов информации, которые отображает оператор SHOW. Без параметров mysqlshow отображает список баз данных:

```
% mysqlshow
+-------------+
| Databases   |
+-------------+
| menagerie   |
| mysql       |
| sampdb      |
| test        |
+-------------+
```

При указании имени базы данных она отображает таблицы в указанной базе данных:

```
% mysqlshow sampdb
Database: sampdb
+-----------+
| Tables    |
+-----------+
| member    |
| president |
+-----------+
```

Вместе с указанными именами базы данных и таблицы mysqlshow позволяет отобразить информацию о столбцах в таблице подобно тому, как это делает оператор SHOW COLUMNS.

Таблицы проекта "Учет успеваемости"

Попробуем посмотреть на примере учета успеваемости в простом журнале, какого рода таблицы нам нужны для реализации проекта контроля успеваемости. На рис. 1.2 изображена страница из обыкновенного журнала. Основным телом журнала является матрица с оценками. Присутствуют и другие данные, необходимые для того, чтобы придать смысл этой информации. Перечислены имена и идентификационные номера учащихся. (Для простоты ограничимся только четырьмя учащимися.) Вверху записаны даты викторин и тестов. На рисунке видно, что викторины проводились 3, 6, 16 и 23 сентября, а тесты — 9 сентября и 1 октября.

Для контроля всей этой информации нам понадобится таблица итоговых оценок `score`. Что должны хранить записи этой таблицы? В каждой строке указываются имя учащегося, дата викторины или теста и результат. На рис. 1.3 нашло отражение такое представление информации. (Даты представлены в формате `'CCYY-MM-DD'`.)

Учащиеся		Оценки						
		Q	Q	T	Q	Q	T	
ID	name	9/3	9/6	9/9	9/1	9/2	10/	. .
1	Billy	14	10	73	14	15	67	.
2	Missy	17	10	68	17	14	73	.
3	Johnny	15	10	78	12	17	82	.
4	Jenny	14	13	85	13	19	79	.
. .	. . .	.	.	.	.	.	.	..

Рис. 1.2. Пример журнала успеваемости

Таблица score

name	date	score
Billy	2002-09-23	15
Missy	2002-09-23	14
Johnny	2002-09-23	17
Jenny	2002-09-23	19
Billy	2002-10-01	67
Missy	2002-10-01	73
Johnny	2002-10-01	82
Jenny	2002-10-01	79

Рис. 1.3. Первоначальная структура таблицы **`score`**

Однако похоже, что такая структура таблицы может создать определенные проблемы. Кажется, в ней недостает какой-то информации. Например, глядя на запись рис. 1.3, трудно сказать, результатом чего она является. Результатом викторины или теста? Возможно, при определении окончательной оценки потребуется определить типы результатов. Это нужно в том случае, когда результаты викторин и результаты тестов имеют различный приоритет. Конечно, можно определить тип результата по диапазону результатов, выпадающих на определенную дату (викторины обычно имеют более низкие оценки, чем тесты), но это будет неверно, так как полученный результат будет основываться не на явных данных, а на предположениях.

Можно просто добавить столбец, содержащий флаг T или Q, для определения типа оценки. На рис. 1.4 T — означает оценку за тест, Q — оценку за викторину. Теперь тип оценки объявлен явным образом. Недостатком является то, что эта информация несколько избыточна. Обратите внимание на то, что даты и типы оценок всегда совпадают. Все результаты за 23 сентября имеют тип Q, а за 1 октября — T. Это бесспорно. При записи результатов тестов и викторин мы несколько раз записываем одну ту же дату и один и тот же тип. Кому нужно вводить эту избыточную информацию?

Попробуем воспользоваться другим представлением данных. Вместо записи типов результатов в таблицу `score` мы будем вычислять его из даты. Будем хранить перечень дат и использовать его для определения "типа события": просто найдите в таблице `event` дату, соответствующую дате в таблице `score`. В результате получим тип события. На рис. 1.5 приведена структура этой таблицы и видно, как реагирует эта связь на дату 23 сентября. Подобрав соответствующую запись в таблице `event`, мы увидим, что этот результат является результатом викторины.

Это значительно лучше, чем пытаться определить тип результата, основываясь на догадках. Вместо этого тип результата определяется прямо из базы данных. Это предпочтительнее хранения результатов в таблице `score`, так как при этом тип записывается только один раз.

Однако сейчас мы сопоставляем информацию из нескольких таблиц. Но здесь будет правомочен вопрос о том, не слишком ли это увеличивает объемы работ, не затрудняет ли это работу?

Таблица score

name	date	score	type
Billy	2002-09-23	15	Q
Missy	2002-09-23	14	Q
Johnny	2002-09-23	17	Q
Jenny	2002-09-23	19	Q
Billy	2002-10-01	67	T
Missy	2002-10-01	73	T
Johnny	2002-10-01	82	T
Jenny	2002-10-01	79	T

Рис. 1.4. Измененная структура таблицы `score`, содержащая тип оценки

Таблица score

name	date	score
Billy	2002-09-23	15
Missy	[redacted]	14
Johnny	2002-09-23	17
Jenny	2002-09-23	19
Billy	2002-10-01	67
Missy	2002-10-01	73
Johnny	2002-10-01	82
Jenny	2002-10-01	79

Таблица event

date	type
2002-09-03	Q
2002-09-06	Q
2002-09-09	T
2002-09-16	Q
[redacted]	Q
2002-10-01	T

Рис. 1.5. Таблицы `score` и `event`, связанные по дате

Отчасти этот вопрос резонен. Хранить два списка записей труднее, чем один. Но давайте посмотрим еще раз на журнал (см. рис 1.2). Разве вы *уже* не храните два набора записей? Рассмотрим такие факты.

- Результаты отслеживаются в ячейках матрицы результатов. Каждая ячейка отмечена именем учащегося и датой (строки — имена, а столбцы — даты). Это один набор данных. Его аналог — содержимое таблицы `score`.
- Как можно узнать, какой тип события представляет каждая дата? Оказывается, над каждой датой надписано маленькое "T" или "Q"! Таким образом, вы отслеживаете взаимосвязь между датой и типом результата. Это второй набор данных. Его аналог — содержимое таблицы `event`.

Другими словами, даже невзирая на то, что вы думаете обо всем этом, на самом деле вы не делаете ничего, что отличалось бы от того, что предлагает автор. Единственным отличием является то, что в журнале эти два набора данных разделены не столь явно.

Страничка журнала иллюстрирует наше понимание информации и сложность переноса этих данных в базу данных. Обычно мы стараемся объединить различные типы информации и представить все это как единое целое. Базы данных работают не так. Вот почему они кажутся искусственными и неестественными. Природная человеческая потребность унифицировать информацию делает страшной мысль о множестве типов данных вместо одного. Поэтому для читателя может оказаться довольно проблематично "мыслить так, как мыслит база данных".

Единственное требование к таблице `event`: даты должны быть уникальными. Это требование появилось потому, что по дате устанавливается связь между записями из таблиц `score` и `event`. Другими словами, нельзя провести две викторины или викторину и тест в один день. В противном случае у вас получатся два набора записей в таблице `score` и две записи в таблице `event`, все с одной и той же датой. И у вас не будет возможности связать записи таблицы `score` с записями таблицы `event`.

Эта проблема не возникнет, если делать в один день не больше одного экзамена, но насколько вероятно обратное? На первый взгляд, это достаточно правдоподобно; в конце концов, не садист же вы, чтобы назначать и викторину, и тест на один и тот же день. Но я верю, что читатель простит мой скепсис. Очень часто я сталкивался с такой ситуацией, когда мне говорят, что этого никогда не случится, но это вдруг происходит, и мне постфактум приходится переделывать структуру таблиц.

Проблемы, которые могут возникнуть в будущем, надо предвидеть и заранее продумать, как избежать их возникновения. Итак, предположим, что вам может понадобиться записать два результата за день. Что делать в этом случае? Оказывается, эту проблему решать не так уж трудно. При минимальных изменениях в структуре данных мы добиваемся хранения нескольких событий по одной дате.

1. Добавим столбец в таблицу `event` и будем в ней хранить номер, уникальный для каждой записи в таблице. Это присвоит каждой записи таблицы свой собственный идентификационный номер. Назовем ее `event_id`. (Просмотрите журнал на рис. 1.2, если вам покажется это странным: идентификационный номер события очень напоминает номер столбца на матрице результатов вашего журнала. Номер может быть записан явно, даже назван "идентификатором события", но это именно он.)

2. При записи результатов в таблицу score вводите идентификатор, а не дату события.

Результат этих изменений показан на рис. 1.6. Связь между таблицами score и event осуществляется с помощью идентификатора. Теперь в таблице event будет храниться не только тип события, но и дата, когда оно произошло. Кроме того, уникальность должна соблюдаться теперь не для даты события, а для его идентификатора. Это означает, что у вас может быть дюжина тестов и викторин в один и тот же день. (Без сомнения, ваши учащиеся начнут нервничать, узнав это.)

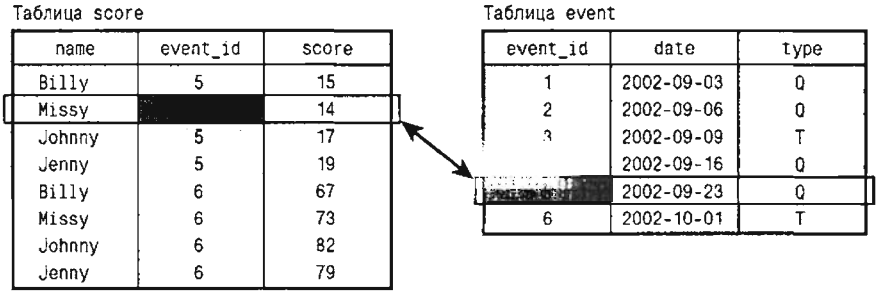


Таблица score

name	event_id	score
Billy	5	15
Missy	[redacted]	14
Johnny	5	17
Jenny	5	19
Billy	6	67
Missy	6	73
Johnny	6	82
Jenny	6	79

Таблица event

event_id	date	type
1	2002-09-03	Q
2	2002-09-06	Q
3	2002-09-09	T
[redacted]	2002-09-16	Q
[redacted]	2002-09-23	Q
6	2002-10-01	T

Рис. 1.6. Таблицы score и event, связанные по идентификатору события

К сожалению, с моей точки зрения, структура таблицы на рис. 1.6 кажется менее удовлетворительной, чем предыдущая. Таблица score является более абстрактной, чем ее предыдущие воплощения. Это явилось следствием того, что стало меньше столбцов с явным содержанием. Структура таблицы score, представленная на рис. 1.4, более понятна, так как содержит столбцы с датами и типами результатов. Последняя структура таблицы (см. рис. 1.6) имеет столбцы, назначение которых объяснить уже трудно. Она достаточно далека от понятных категорий. Кому захочется смотреть на таблицу score с "идентификатором события"? Для непосвященного это ничего не значит.

Теперь мы стоим на распутье. Вероятно, вы заинтересовались возможностью хранить журнал успеваемости в электронном виде и не заниматься вычислениями вручную при подсчете баллов. Но по мере знакомства с тем, как можно хранить информацию в базе данных, вы заметили, как эта информация абстрактна и несвязанна.

Возникает естественный вопрос: "Насколько целесообразно использовать базу данных? Может быть, СУБД MySQL не для нас?" Как можно догадаться, мой ответ может быть только отрицательным, иначе эта книга теряет свой смысл. Ведь когда вы обдумываете, как сделать работу, не мешает рассмотреть различные альтернативы, например, выбор какой-либо базы данных (предположим, СУБД MySQL) или что-то наподобие электронных таблиц.

- Электронная таблица тоже может хранить данные в виде строк и столбцов. Это делает электронную таблицу и журнал успеваемости концептуально и визуально подобными.
- Используя электронную таблицу, можно производить вычисления. Это позволит суммировать оценки учащихся. Задача немного усложняется "присутствием" весового коэффициента. Но и это преодолимо.

Однако если вам захочется просмотреть только какие-то определенные данные (только тесты или викторины), сравнить оценки девочек с оценками мальчиков, отобразить суммарные данные различными способами — это уже другое дело. Электронной таблице с этими задачами будет уже справиться труднее. А СУБД с этим справляются запросто.

Абстрактность и несвязанность информации, находящейся в таблицах базы данных, — не проблема. Во время планирования и проектирования структуры базы данных необходимо думать о представлении базы данных. Так что во время работы с базой данных ее механизм поможет отображать информацию в понятном виде. Поэтому не надо смотреть на таблицы базы данных просто как на груду несвязной информации.

Например, при выборке результатов из таблицы `scores` идентификаторы событий вам не нужны, а нужны даты. По идентификатору события база данных просмотрит даты из таблицы `event` и выдаст их. Может, просмотреть, с какими результатами вы имеете дело? С результатами тестов или результатами викторин? Это — также не проблема. База данных найдет типы результатов аналогичным образом, — пользуясь идентификатором события. Помните, для чего удобна СУБД, подобная СУБД MySQL: для связывания информации из одного источника с информацией из другого. В случае с данными по успеваемости СУБД MySQL связывает информацию с помощью идентификатора события.

Теперь посмотрим, как работает связь одной таблицы с другой в СУБД MySQL. Для этого сделаем запрос. Предположим, нам требуются все результаты за 23 сентября 2002 года. Получится вот такой запрос:

```
SELECT score.name, event.date, score.score, event.type
FROM score, event
WHERE event.date = '2002-09-23'
AND score.event_id = event.event_id;
```

Невероятно! Посредством объединения (связывания) записей из таблиц `score` и `event` этот запрос производит выборку имени учащегося, даты, оценки и типа оценки. Полученный результат выглядит следующим образом:

```
+--------+------------+-------+------+
| name   | date       | score | type |
+--------+----------- +-------+------+
| Billy  | 2002-09-23 |    15 | Q    |
| Missy  | 2002-09-23 |    14 | Q    |
| Johnny | 2002-09-23 |    17 | Q    |
| Jenny  | 2002-09-23 |    19 | Q    |
+--------+------------+-------+------+
```

Заметили что-то знакомое в формате этой информации? А должны были! Это похоже на структуру таблицы, показанную на рис. 1.4! Нет необходимости знать идентификатор события для того, чтобы получить этот результат. Просто вы описываете дату, которая вас интересует, и предоставляете возможность СУБД MySQL определить, какие оценки принадлежат этой дате. Удивительно, куда делась вся абстракция и связанность, присущая базе данных?

Конечно, посмотрев на все это, у вас останутся вопросы. Например, нельзя ли побыстрее узнать оценки для заданной даты? К сожалению, нет. Однако есть способы избежать ввода нескольких строк на языке SQL каждый раз, когда надо запустить запрос.

Впрочем, я, кажется, немного поторопился, показав этот запрос. Это, поверьте мне, немного проще, чем то, что мы будем использовать для получения оценок. Это значит, что настало время внести еще одну поправку в структуру таблицы. Вместо хранения имени учащегося в таблице `score` будем хранить там его идентификационный номер, т.е. столбец `name` будет заменен столбцом `ID`. Затем будет создана новая таблица `student`, содержащая столбцы `name` и `student_id` (рис. 1.7).

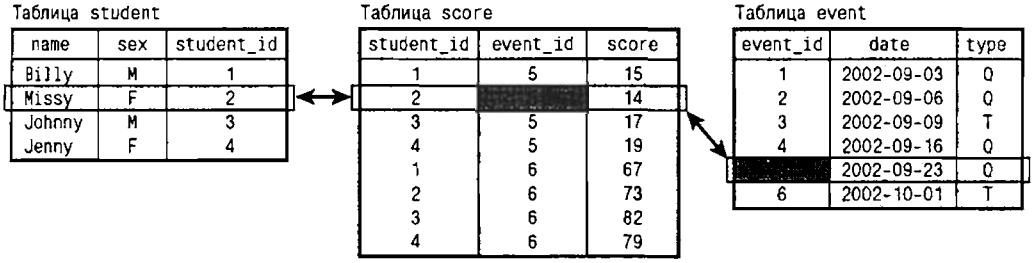


Рис. 1.7. Таблицы `score`, `event` и `student`, связанные по идентификаторам события и учащегося

Зачем вносить эти изменения? Дело в том, что могут существовать двое учащихся с одинаковым именем. Использование уникального идентификатора учащегося позволит вам получить возможность различать их оценки. (Это аналогично тому, как мы с помощью уникального идентификатора события сможем разделять оценки, полученные за викторину, и оценки, полученные за тесты.)

Повторим запрос после изменения структуры таблиц.

```
SELECT student.name, event.date, score.score, event.type
FROM event, score, student
WHERE event.date = '2002-09-23'
AND event.event_id = score.event_id
AND score.student_id = student.student_id;
```

Не расстраивайтесь, если вы не поняли смысл этого запроса сразу. Мы вернемся к нему в процессе более углубленного изучения. Но разница между "сейчас и потом" заключается в том, что потом вы его поймете. Конечно, я не шучу.

Вероятно, вы заметили, что я что-то изменил в таблице `student`, чего не было в журнале. Теперь она содержит столбец для хранения информации о поле учащегося. Это позволит производить более простые операции, например подсчет девочек и мальчиков в классе, или более сложные, такие как сравнение суммарных оценок девочек и мальчиков.

С проектом учета успеваемости учащихся мы почти закончили. Осталась еще одна таблица, предназначенная для отметок посещаемости. Ее содержимое относительно несложно: идентификационный номер учащегося и дата (рис. 1.8). Каждая строка в таблице свидетельствует о том, что определенный учащийся отсутствовал в определенный день. В конце этой главы мы обратимся к вычислительным возможностям СУБД MySQL для того, чтобы узнать, сколько раз учащийся пропустил занятия.

Теперь мы готовы к созданию таблиц проекта контроля успеваемости, поскольку знаем, что они собой представляют.

Таблица absence

student_id	date
2	2002-09-02
4	2002-09-15
2	2002-09-20

Рис. 1.8. Таблица absence

Оператор CREATE TABLE, предназначенный для создания таблицы student, имеет следующий вид:

```
CREATE TABLE student
(
  name VARCHAR(20) NOT NULL,
  sex ENUM('F','M') NOT NULL,
  student_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (student_id)
);
```

Введите этот оператор, работая в mysql, или запустите следующую команду из оболочки:

```
% mysql sampdb < create_student.sql
```

После этого оператор CREATE TABLE создаст таблицу student с тремя столбцами: name, sex и student_id.

Столбец name является столбцом переменной длины, который может хранить до 20 символов. Такое представление имени проще, чем в случае с таблицами "Исторической Лиги": один столбец используется для хранения имени и фамилии. Это потому, что я знаю наперед, что ни в одном примере запроса для базы данных контроля успеваемости не потребуется раздельная выборка имени и фамилии.

Столбец sex отражает пол учащегося. Это столбец перечисляемого типа ENUM, а следовательно, что он может принимать одно из двух значений: 'F' или 'M'. Этот тип столбца полезен тогда, когда столбец принимает значения только из ограниченного диапазона. В этом случае можно использовать тип CHAR(1), но ENUM будет явно определять, какие значения может принимать столбец. Оператор DESCRIBE tbl_name для такой таблицы покажет, какие точно значения может принимать этот столбец.

Столбец student_id является столбцом целого типа, в котором будет храниться уникальный идентификационный номер учащегося. Обычно идентификационные номера можно получить из какого-то центрального источника, каким может быть учительская. Мы же создадим их с помощью столбца AUTO_INCREMENT, практически идентичного столбцу member_id, который является частью таблицы member, созданной ранее. Если вы действительно собирались узнать идентификационные номера из учительской, можно было бы создать столбец student_id без атрибута AUTO_INCREMENT, оставив только атрибут PRIMARY KEY, чтобы избежать повторяющихся номеров. Таблица table выглядит следующим образом:

```
CREATE TABLE event
(
  date DATE NOT NULL,
  type ENUM('T','Q') NOT NULL,
  event_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
PRIMARY KEY (event_id)
);
```

Введите этот оператор в mysql или запустите из оболочки следующую команду:

```
% mysql sampdb < create_event.sql
```

Все столбцы объявлены здесь как NOT NULL потому, что ни одна из них не может быть пропущена.

Столбец date хранит стандартное для СУБД MySQL значение типа DATE в формате 'CCYY-MM-DD'.

Столбец type хранит тип оценки и так же, как столбец sex в таблице student type, является перечисляемым типом. Допустимые значения здесь T и Q, что соответствует "тесту" и "викторине".

Столбец event_id является столбцом типа AUTO_INCREMENT, полностью идентичным столбцу student_id в таблице student. Аналогично столбцу student_id, в таблице student конкретные значения менее важны, чем то, что они должны быть уникальными.

Таблица score имеет следующий вид:

```
CREATE TABLE score
(
  student_id INT UNSIGNED NOT NULL,
  event_id INT UNSIGNED NOT NULL,
  PRIMARY KEY (event_id, student_id),
  score INT NOT NULL
);
```

Введите этот оператор, работая в mysql, или запустите из оболочки следующую команду:

```
% mysql sampdb < create_score.sql
```

Столбцы student_id и event_id объявлены с типом INT, т.е. предполагается, что все значения оценок являются целочисленными и задают ученика и экзамен, к которому относится оценка. Связав по этим полям таблицы student и event, мы сможем узнать имя учащегося и дату экзамена. Комбинация этих столбцов является первичным ключом. Это предотвратит от дублирования оценок одного учащегося за один и тот же тест или викторину. Упростится также последующее изменение оценки. Например, если оценка была введена неверно, в дальнейшем будет нетрудно изменить ее с помощью оператора REPLACE. При этом нет необходимости выполнять операции DELETE и INSERT.

Обратите внимание на то, что комбинация student_id и event_id является уникальной. В таблице score нет столбца, уникального самого по себе. Каждому идентификатору event_id может соответствовать несколько записей (по одной на каждого учащегося). Аналогично, каждому идентификатору student_id может соответствовать сразу несколько записей (по одной на каждый тест или викторину).

Столбец score является целочисленным столбцом. Я предполагаю, что оценка всегда будет представляться целыми числами. Следовательно, если в оценках необходимо использовать числа, подобные 58,5, нужно задать типы данных с плавающей точкой, такие как FLOAT или DECIMAL. Таблица absence имеет следующую структуру:

```
CREATE TABLE absence
(
```

```
  student_id INT UNSIGNED NOT NULL,
  date DATE NOT NULL,
  PRIMARY KEY (student_id, date)
);
```

Введите этот оператор, работая в mysql, или запустите из оболочки следующую команду:

```
% mysql sampdb < create_absence.sql
```

Во избежание потери данных столбцы student_id и date объявлены как NOT NULL. Комбинация этих столбцов объявлена первичным ключом. Это воспрепятствует случайному созданию повторяющихся записей. Ведь действительно будет несправедливо посчитать одного учащегося отсутствующим дважды в день.

Дополнение таблиц

Наконец, наша база данных и ее таблицы созданы. Теперь нужно заполнить таблицы. И неплохо бы знать, как просмотреть содержимое таблицы после добавления новых записей. Несмотря на то что восстановление данных не рассматривается до следующего раздела, вы должны знать хотя бы оператор, который покажет вам содержание таблицы *tbl_name*:

```
SELECT * FROM tbl_name;
```

Например:

```
mysql> SELECT * FROM student;
Empty set (0.00 sec)
```

Сейчас видно, что таблица пуста, но после выполнения примеров в этом разделе мы добьемся другого результата.

Можно назвать несколько методов добавления информации в базы данных. Можно добавить записи в таблицы вручную с помощью оператора INSERT. Можно добавлять записи прямо из файла или как исходные данные с помощью оператора LOAD DATA; можно с помощью программы mysqlimport.

В этом разделе демонстрируются все эти методы. Вам остается только проверить их в работе. С помощью команд, описанных здесь, вы сможете удалять и заново создавать таблицы, а также заполнять их данными. Сделав это, можно быть уверенным в том, что эти таблицы будут содержать записи, аналогичные тем, с которыми я буду работать в следующем разделе. (Вы можете пропустить этот раздел, если уже знаете, как добавляются записи.)

Начнем добавление записей с оператора INSERT. Это оператор языка SQL, в котором определяется таблица, куда будет производиться добавление, строка добавляемых данных и значений.

Оператор INSERT имеет несколько форм.

- Определение значений всех столбцов.

  ```
  INSERT INTO tbl_name VALUES(value1,value2,...);
  ```

  Например:

  ```
  mysql> INSERT INTO student VALUES('Kyle','M',NULL);
  mysql> INSERT INTO event VALUES('2002-9-3','Q',NULL);
  ```

Список VALUE должен содержать все столбцы, имеющиеся в таблице. (Обычно это порядок следования названий столбцов, заданный в операторе CREATE TABLE.)

Если вы не знаете порядка следования столбцов, воспользуйтесь оператором DESCRIBE *tbl_name*.

Выделять строковые значения или значения типа "дата" можно как одинарными, так и двойными кавычками. Пустые значения здесь будут присвоены столбцам с атрибутом AUTO_INCREMENT в таблицах student и event. (Ввод недостающего значения приводит к генерированию следующего номера для student_id или event_id.)

Начиная с версии СУБД MySQL 3.22.5 можно добавить сразу несколько строк с помощью одного оператора INSERT:

```
INSERT INTO tbl_name VALUES(...),(...),...
```

Например:

```
mysql> INSERT INTO student VALUES('Abby','F',NULL),('Kyle','M',NULL);
```

Такой оператор потребует ввода меньшего количества информации, да и сервер будет эффективнее обработать эту команду.

- Вы можете давать имена столбцам, которым хотите присваивать значения, и потом выводить их. Это полезно, если вы хотите создать запись, для которой придется изначально создать несколько столбцов.

```
INSERT INTO tbl_name (col_name1,col_name2,...)
VALUES(value1,value2,...);
```

Например:

```
mysql> INSERT INTO member (last_name,first_name)
VALUES('Stein','Waldo');
```

Начиная с версии СУБД MySQL 3.22.5 можно добавить сразу несколько строк с помощью одного оператора INSERT:

```
mysql> INSERT INTO student (name, sex)
VALUES('Abby','F'),('Kyle','M');
```

Столбцы, не перечисленные в списке столбцов, получают значение по умолчанию. Например, поскольку предыдущие записи не содержат значений в столбце member_id или student_id, MySQL по умолчанию присвоит им значение NULL. (Столбцы member_id и student_id объявлены с атрибутом AUTO_INCREMENT, поэтому результатом будет присвоение порядкового номера точно так же, как это происходит при присвоении значения NULL.)

- Версия СУБД MySQL 3.22.10 позволяет задавать имена столбцов и их значение в форме *col_name = value*:

```
INSERT INTO tbl_name SET col_name1=value1, col_name2=value2, ... ;
```

Например:

```
mysql> INSERT INTO student SET last_name ='Stein',
first_name='Waldo';
```

Столбцы, не перечисленные в выражении SET, получают значение по умолчанию. Такая форма оператора INSERT неприменима для вставки нескольких строк.

Существуют и другие методы загрузки данных в таблицы базы данных прямо из файлов. Например, если у вас есть файл с именем `insert_president.sql`, который содержит оператор `INSERT` для добавления новых записей, вы можете запустить их следующим образом:

```
% mysql sampdb < insert_president.sql
```

(Файл `insert_president.sql` содержится в дистрибутиве `sampdb`.)

Если `mysql` уже запущен, вы можете использовать команду `SOURCE`, чтобы прочитать файл:

```
mysql> SOURCE insert_president.sql;
```

Команда `SOURCE` требует MySQL версии 3.23.9 или более поздней.

Если ваши записи хранятся в файле в необработанном виде, вы можете использовать оператор `LOAD DATA` или программу `mysqlimport` для загрузки этих данных.

Оператор `LOAD DATA` действует как загрузчик массовых данных, считывающий данные из файла. Так он работает из `mysql`:

```
mysql> LOAD DATA LOCAL INFILE "member.txt" INTO TABLE member;
```

Допуская, что файл `member.txt` находится в текущей папке на клиентском компьютере, этот оператор читает его и отправляет на сервер для внесения в таблицу `member`. (Файл `member.txt` вы найдете в дистрибутиве `sampdb`.)

Формат данных в файлах по умолчанию предполагает, что столбцы разделены табуляциями, строки заканчиваются с началом новой строки, значения располагаются в порядке следования столбцов таблицы. Но есть возможность загружать файлы в других форматах или определять другой порядок столбцов. Подробнее с этой проблемой можно ознакомиться в приложении Г, "Синтаксис SQL".

Вариант `LOAD DATA LOCAL` не работает в версиях до 3.22.15, так как начиная только с этой версии была добавлена возможность чтения данных прямо с компьютера клиента. (Без ключа `LOCAL` загружаемый файл должен быть расположен прямо на сервере, и поэтому для загрузки такого файла пользователь должен обладать широкими правами доступа к серверу, которых у него обычно нет.) Кроме того, в версии 3.23.49 функция `LOCAL` может присутствовать, но быть отключенной. Если при использовании оператора `LOAD DATA` возникает ошибка, попробуйте запустить `mysql` с ключом `- -local-infile`:

```
% mysql --local-infile sampdb
mysql> LOAD DATA LOCAL INFILE 'member.txt' INTO TABLE member;
```

Если так не получится, возможно, требуется использовать функцию `LOCAL` сервером. Чтобы узнать, как это делается, обратитесь к главе 11, "Операторы и функции".

Программу `mysqlimport` можно рассматривать как интерфейс между вводом на уровне оболочки операционной системы и оператором `LOAD DATA`.

```
% mysql - - local sampdb member.txt
```

Такая команда не сработает для версии СУБД MySQL ниже 3.22.15, поскольку для нее потребуется оператор `LOAD DATA LOCAL`. Здесь все делается так, как это делается в `mysql`: если нужны параметры для связи, указывайте их перед именем базы данных.

Программа `mysqlimport` получает имя таблицы, для которой предназначены данные из имени файла. (Для этого используется все, что указывается в имени

файла до первой точки.) Например, данные из файла member.txt будут загружены в таблицу member, а из файла president.txt — в таблицу president. Будьте осторожны, если возникла необходимость загрузить таблицы базы данных из нескольких файлов. Так, при использовании имен member1.txt и member2.txt для корректной работы программы mysqlimport должны использоваться таблицы member1 и member2. В нашем случае подойдут имена member.1.txt и member.2.txt или member.txt1 и member.txt2.

Если вы уже поэкспериментировали с добавлением данных в тестовую базу данных, удалите содержимое таблиц и загрузите данные, необходимые для ознакомления с материалом следующего раздела.

Для этого прямо из оболочки выполните следующие команды:

```
% mysql sampdb < create_president.sql
% mysql sampdb < insert_president.sql
% mysql sampdb < create_member.sql
% mysql sampdb < insert_member.sql
% mysql sampdb < create_student.sql
% mysql sampdb < insert_student.sql
% mysql sampdb < create_score.sql
% mysql sampdb < insert_score.sql
% mysql sampdb < create_event.sql
% mysql sampdb < insert_event.sql
% mysql sampdb < create_absence.sql
% mysql sampdb < insert_absence.sql
```

Этот ввод можно упростить и ввести команду (в системе UNIX):

```
% sh init_all_tables.sh sampdb
```

Для Windows применима следующая команда:

```
C:\> init_all_tables.bat sampdb
```

Если требуется указать параметры соединения, укажите их в командной строке перед именем базы данных.

Выборка информации

Теперь, когда таблицы созданы и заполнены данными, посмотрим, что можно сделать с этими данными. Оператор SELECT позволяет производить выборку и отображать информацию из таблиц любым способом. Можно сделать выборку сразу всех столбцов и всех строк таблицы:

```
mysql> SELECT * FROM president
```

Или сделать выборку одного столбца из одной строки таблицы:

```
mysql> SELECT birth_date FROM president WHERE last_name = "Eisenhover"
```

Оператор SELECT состоит из нескольких предложений (частей), которые можно сочетать в любом порядке, в зависимости от того, какая информация требуется для выборки. Любое предложение может быть сложным или простым, в зависимости от чего весь оператор SELECT может быть сложным или простым. Но я гарантирую, что в этой книге вы не найдете запросов размером со страницу, для объяснения которых потребуется еще одна такая же страница.

Общий синтаксис оператора SELECT имеет вид

```
SELECT что выбирается
FROM таблица или таблицы
WHERE условия, которые должны удовлетворяться
```

Для создания оператора SELECT необходимо определить, что требуется выбрать из таблиц. Для этой цели служат предложения FROM и WHERE, которые используются наиболее часто, и GROUP BY, ORDER BY и LIMIT, которые встречаются реже. Нужно помнить, что SQL является языком со свободным синтаксисом, поэтому вы не обязательно должны следовать авторскому стилю написания запросов.

Предложение FROM обычно присутствует в операторе SELECT, но в нем нет необходимости, если отображаются данные не из таблиц. Например, следующий запрос просто отображает значения выражений, которые могут вычисляться без ссылки на таблицу, так что в предложении FROM нет необходимости:

```
mysql> SELECT 2+2, 'Hello, world', VERSION();
+-----+--------------+----------------+
| 2+2 | Hello, world | VERSION()      |
+-----+--------------+----------------+
|   4 | Hello, world | 4.0.4-beta-log |
+-----+--------------+----------------+
```

При использовании предложения FROM для определения таблицы, из которой требуется произвести выборку, мы можем получить наиболее "общую" форму запроса. Для этого вместо указания конкретного столбца введем "*", что означает "все". Такой запрос выбирает и отображает все столбцы из таблицы student.

```
mysql> SELECT * FROM student;
+-----------+-----+------------+
| name      | sex | student_id |
+-----------+-----+------------+
| Megan     | F   |          1 |
| Joseph    | M   |          2 |
| Kyle      | M   |          3 |
| Katie     | F   |          4 |
...
```

Значения всех столбцов возвращаются в том же порядке, в котором они хранятся в таблице. Этот порядок совпадает с порядком, в котором столбцы перечислены оператором DESCRIBE student ('...' в конце распечатки означает, что запрос возвращает больше строк, чем показано).

В операторе SELECT имена столбцов можно указывать явно. Для выборки только имен учащихся нужно сделать следующий запрос:

```
mysql> SELECT name FROM student;
+-----------+
| name      |
+-----------+
| Megan     |
| Joseph    |
| Kyle      |
| Katie     |
...
```

Для отображения нескольких столбцов в операторе SELECT можно через запятую указать имена этих столбцов. Этот оператор аналогичен оператору SELECT * FROM student, но каждый столбец указывается здесь явным образом:

```
mysql> SELECT name, sex, student_id FROM student;
+-----------+-----+------------+
| name      | sex | student_id |
+-----------+-----+------------+
| Megan     | F   |          1 |
| Joseph    | M   |          2 |
| Kyle      | M   |          3 |
| Katie     | F   |          4 |
...
```

Столбцы можно перечислять в произвольном порядке.

```
SELECT name, student_id FROM student;
SELECT student_id, name FROM student;
```

Столбцы можно указывать сколько угодно раз.

MySQL позволяет вам выбирать столбцы больше чем из одной таблицы за раз. Вы узнаете об этом ниже, в разделе "Выборка данных из нескольких таблиц".

Имена столбцов можно указывать в любом регистре.

```
SELECT name, student_id FROM student;
SELECT NAME, STUDENT_ID FROM student;
SELECT nAmE, sTuDeNt_Id FROM student;
```

С другой стороны, имена таблиц и баз данных могут быть чувствительны к регистру. Все зависит от файловой системы ОС, работающей на узле сервера. На сервере, работающем под управлением ОС UNIX, имена таблиц и баз данных чувствительны к регистру, так как имена файлов в ОС UNIX чувствительны к регистру. В ОС Windows имена файлов не чувствительны к регистру, поэтому имена таблиц и баз данных тоже будут не чувствительны к регистру.

(Исключением является ОС Mac OS X, где системы с файловой системой HFS+ не чувствительны к регистру, а системы с файловой системой UFS – чувствительны.) Если вы хотите, чтобы таблицы в MySQL не зависели от регистра, обратитесь к разделу "Ограничения операционной системы на имена баз данных и таблиц" в главе 10, "Каталог данных MySQL".

Определение критериев выборки

Ограничение набора выбираемых оператором SELECT записей производится с помощью предложения WHERE, которым определяется набор выбираемых строк.

В качестве критериев можно задавать цифровые значения.

```
mysql> SELECT * FROM score WHERE score > 95;
+------------+----------+-------+
| student_id | event_id | score |
+------------+----------+-------+
|          5 |        3 |    97 |
|         18 |        3 |    96 |
|          1 |        6 |   100 |
|          5 |        6 |    97 |
|         11 |        6 |    98 |
|         16 |        6 |    98 |
+------------+----------+-------+
```

Можно задавать в качестве критериев строковые значения. (Обратите внимание на то, что операции сравнения строк обычно не чувствительны к регистру.)

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name='ROOSEVELT';
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Roosevelt | Theodore    |
| Roosevelt | Franklin D. |
+-----------+-------------+
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name='roosevelt';
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Roosevelt | Theodore    |
| Roosevelt | Franklin D. |
+-----------+-------------+
```

Можно также производить выборку по дате.

```
mysql> SELECT last_name, first_name, birth FROM president
    -> WHERE birth < '1750-1-1';
+------------+------------+------------+
| last_name  | first_name | birth      |
+------------+------------+------------+
| Washington | George     | 1732-02-22 |
| Adams      | John       | 1735-10-30 |
| Jefferson  | Thomas     | 1743-04-13 |
+------------+------------+------------+
```

Выборку можно производить и по комбинации значений.

```
mysql> SELECT last_name, first_name, birth, state FROM president
    -> WHERE birth < '1750-1-1' AND (state='VA' OR state='MA');
+------------+------------+------------+-------+
| last_name  | first_name | birth      | state |
+------------+------------+------------+-------+
| Washington | George     | 1732-02-22 | VA    |
| Adams      | John       | 1735-10-30 | MA    |
| Jefferson  | Thomas     | 1743-04-13 | VA    |
+------------+------------+------------+-------+
```

Выражения в предложениях WHERE могут содержать арифметические операторы (табл. 1.1), операторы сравнения (табл. 1.2) и логические операторы (табл. 1.3). Выражения группируются с помощью скобок. Операторы могут содержать константы, столбцы таблиц и вызовы функций. Позже вы познакомитесь с применением функций СУБД MySQL в запросах, а их полный список представлен в приложении В, "Операторы и функции".

Таблица 1.1. Арифметические операторы

Оператор	Значение
+	Сложение
-	Вычитание
*	Умножение
/	Деление

Таблица 1.2. Операторы сравнения

Оператор	Значение
<	Меньше
<=	Меньше или равно
=	Равно
!= или <>	Не равно
>=	Больше или равно
>	Больше

Таблица 1.3. Логические операторы

Оператор	Значение
AND	Логическое "и"
OR	Логическое "или"
NOT	Логическое отрицание

При создании запроса, требующего использования логических операторов, необходимо понимать разницу между оператором логического "и" и обычным использованием слова "и" в повседневной жизни. Предположим, что требуется найти "президентов, родившихся в штате Виргиния, и президентов, родившихся в штате Массачусетс". Обратите внимание на то, как при этом произносится "и", что предполагает следующее написание запроса:

```
mysql> SELECT last_name, first_name, state FROM president
    -> WHERE state='VA' AND state='MA';
Empty set (0.36 sec)
```

При отсутствии выборки ясно, что запрос трактуется "выбрать президентов, родившихся как в штате Виргиния, так и в штате Массачусетс", что совершенно бессмысленно. В английском языке этот запрос можно делать с использованием "и", но в языке SQL эти два условия объединяются оператором OR.

```
mysql> SELECT last_name, first_name, state FROM president
    -> WHERE state='VA' OR state='MA';
+------------+-------------+-------+
| last_name  | first_name  | state |
+------------+-------------+-------+
| Washington | George      | VA    |
| Adams      | John        | MA    |
| Jefferson  | Thomas      | VA    |
| Madison    | James       | VA    |
| Monroe     | James       | VA    |
| Adams      | John Quincy | MA    |
| Harrison   | William H.  | VA    |
| Tyler      | John        | VA    |
| Taylor     | Zachary     | VA    |
| Wilson     | Woodrow     | VA    |
| Kennedy    | John F      | MA    |
| Bush       | George H.W. | MA    |
+------------+-------------+-------+
```

Нужно принимать во внимание, что различие между разговорным языком и SQL проявляется не только при формулировании собственных запросов, но и

при написании запросов для других людей. Я советую сначала точно определить, чего они хотят. Например, вышеуказанный запрос проще сформулировать следующим образом: "Выбрать президентов, родившихся в штате Виргиния или в штате Массачусетс".

Значение NULL

NULL — это особенное значение, потому что оно обозначает ситуацию, когда "нет значения"; его нельзя выбрать как обычное значение и сравнить с другим значением, так как это можно сделать с обычными значениями. При сравнении пустого значения с помощью обычного арифметического оператора сравнения будет получен неопределенный результат.

```
mysql> SELECT NULL < 0, NULL = 0, NULL != 0, NULL > 0;
+----------+----------+-----------+----------+
| NULL < 0 | NULL = 0 | NULL != 0 | NULL > 0 |
+----------+----------+-----------+----------+
| NULL     | NULL     | NULL      | NULL     |
+----------+----------+-----------+----------+
```

Более того, нельзя сравнить два значения NULL. Это естественно — результат сравнения двух неизвестных значений сам не может быть известен.

```
mysql> SELECT NULL = NULL, NULL != NULL;
+-------------+--------------+
| NULL = NULL | NULL != NULL |
+-------------+--------------+
| NULL        | NULL         |
+-------------+--------------+
```

Для выборки пустых значений требуется особый синтаксис. Вместо использования символов "=" или "!=" для проверки равенства или неравенства нужно задавать IS NULL или IS NOT NULL.

Например, мы присвоили дате смерти живых президентов значение NULL, поэтому выбрать их будет проще всего, сделав следующий запрос:

```
mysql> SELECT last_name, first_name FROM president WHERE death IS NULL;
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Ford      | Gerald R    |
| Carter    | James E.    |
| Reagan    | Ronald W.   |
| Bush      | George H.W. |
| Clinton   | William J.  |
| Bush      | George W.   |
+-----------+-------------+
```

Поиск имен, имеющих суффикс, можно осуществить, пользуясь выражением IS NOT NULL.

```
mysql> SELECT last_name, first_name, suffix
    -> FROM president WHERE suffix IS NOT NULL;
+-----------+------------+--------+
| last_name | first_name | suffix |
+-----------+------------+--------+
| Carter    | James E.   | Jr.    |
+-----------+------------+--------+
```

В СУБД MySQL версии 2.23 есть уникальный оператор сравнения '<=>', работающий даже при сравнении пустых значений. Два предыдущих запроса можно изменить, воспользовавшись этим оператором.

```
mysql> SELECT last_name, first_name FROM president WHERE death <=> NULL;
+-----------+------------+
| last_name | first_name |
+-----------+------------+
| Ford      | Gerald R   |
| Carter    | James E.   |
| Reagan    | Ronald W.  |
| Bush      | George H.W.|
| Clinton   | William J. |
| Bush      | George W.  |
+-----------+------------+
mysql> SELECT last_name, first_name, suffix
    -> FROM president WHERE NOT (suffix <=> NULL);
+-----------+------------+--------+
| last_name | first_name | suffix |
+-----------+------------+--------+
| Carter    | James E.   | Jr.    |
+-----------+------------+--------+
```

Сортировка результатов запроса

Очень скоро пользователь заметит, что в результате выполнения команды SELECT * FROM *tbl_name* он получает выборку данных из нужной таблицы в том же порядке, в котором она была загружена. Нельзя полностью полагаться, что данные в таблице всегда будут храниться в нужном порядке. В процессе работы с таблицей записи модифицируются, удаляются и добавляются. (Удаление записей оставляет "дыры" неиспользуемого пространства, которые СУБД MySQL попытается заполнить позже, по мере добавления записей в таблицу.)

Вы должны знать следующее: при выборке данных сервер не гарантирует их порядок. Для их сортировки предназначено предложение ORDER BY.

Следующий запрос выводит имена президентов, отсортированные в алфавитном порядке по фамилии:

```
mysql> SELECT last_name, first_name FROM president
    -> ORDER BY last_name;
+------------+---------------+
| last_name  | first_name    |
+------------+---------------+
| Adams      | John          |
| Adams      | John Quincy   |
| Arthur     | Chester A.    |
| Buchanan   | James         |
...
```

Ключевым словом ASC или DESC можно задать порядок сортировки столбца по возрастанию или убыванию. Например, для сортировки имен президентов в обратном (убывающем) порядке можно сделать такой запрос:

```
mysql> SELECT last_name, first_name FROM president
    -> ORDER BY last_name DESC;
+------------+---------------+
```

```
| last_name   | first_name     |
+-------------+----------------+
| Wilson      | Woodrow        |
| Washington  | George         |
| Van Buren   | Martin         |
| Tyler       | John           |
...
```

По умолчанию принимается порядок сортировки по возрастанию.

Сортировка может задаваться по нескольким столбцам, при этом любой столбец может быть отсортирован независимо от других столбцов как в возрастающем порядке, так и в убывающем. Следующий запрос производит выборку строк из таблицы president, отсортированных в убывающем порядке по названию штата, где родились президенты, и по фамилии в убывающем порядке внутри каждого штата.

```
mysql> SELECT last_name, first_name, state FROM president
    -> ORDER BY state DESC, last_name ASC;
+-------------+----------------+-------+
| last_name   | first_name     | state |
+-------------+----------------+-------+
| Arthur      | Chester A.     | VT    |
| Coolidge    | Calvin         | VT    |
| Harrison    | William H.     | VA    |
| Jefferson   | Thomas         | VA    |
| Madison     | James          | VA    |
| Monroe      | James          | VA    |
| Taylor      | Zachary        | VA    |
| Tyler       | John           | VA    |
| Washington  | George         | VA    |
| Wilson      | Woodrow        | VA    |
| Eisenhower  | Dwight D.      | TX    |
| Johnson     | Lyndon B.      | TX    |
...
```

Если вы сортируете столбец, который может содержать значения NULL, будет ли он отсортирован? Все зависит от версии. В MySQL версии 4.0.2 они всегда будут в начале, даже если указать параметр DESC. Прежде они выводились в начале списка при сортировке по возрастанию и в конце — при сортировке по убыванию.

Если вы хотите быть уверены, что значения NULL появятся там, где надо, добавьте дополнительный столбец сортировки, который отличает пустые значения от непустых.

Например, чтобы отсортировать президентов по дате смерти, но при этом поместить живущих президентов (со значениями даты смерти NULL) в начало списка, используйте следующий запрос:

```
mysql> SELECT last_name, first_name, death FROM president
    -> ORDER BY IF(death IS NULL,0,1), death;
+-------------+----------------+------------+
| last_name   | first_name     | death      |
+-------------+----------------+------------+
| Ford        | Gerald R.      | NULL       |
| Carter      | James E.       | NULL       |
| Reagan      | Ronald W.      | NULL       |
| Bush        | George H.W.    | NULL       |
| Clinton     | William J.     | NULL       |
```

```
| Bush       | George W.     | NULL       |
| Washington | George        | 1799-12-14 |
| Adams      | John          | 1826-07-04 |
...
| Truman     | Harry S.      | 1972-12-26 |
| Johnson    | Lyndon B.     | 1973-01-22 |
| Nixon      | Richard M.    | 1994-04-22 |
+------------+---------------+------------+
```

А чтобы поместить живущих президентов в конец списка, используйте следующий запрос:

```
mysql> SELECT last_name, first_name, death FROM president
    -> ORDER BY IF(death IS NULL,1,0), death;
+------------+---------------+------------+
| last_name  | first_name    | death      |
+------------+---------------+------------+
| Washington | George        | 1799-12-14 |
| Adams      | John          | 1826-07-04 |
...
| Truman     | Harry S.      | 1972-12-26 |
| Johnson    | Lyndon B.     | 1973-01-22 |
| Nixon      | Richard M.    | 1994-04-22 |
| Ford       | Gerald R.     | NULL       |
| Carter     | James E.      | NULL       |
| Reagan     | Ronald W.     | NULL       |
| Bush       | George H.W.   | NULL       |
| Clinton    | William J.    | NULL       |
| Bush       | George W.     | NULL       |
+------------+---------------+------------+
```

Функция `IF()` определяет выражение и возвращает значение второго и третьего аргумента, в зависимости от того, является ли результат выражения "правдой" или "ложью". В первом запросе `IF()` присваивает строкам с пустыми датами смерти значения 0, а строкам с датами смерти — значения 1, тем самым помещая все строки без даты смерти перед строками с датой смерти. Во втором запросе все происходит наоборот. Это может использоваться начиная с MySQL версии 3.23.2, которая является первой версией, позволяющей помещать выражения в предложения `ORDER BY`.

Ограничение количества строк результатов запроса

Предложение `LIMIT` предназначено для ограничения количества строк, выводимых запросом. Этот элемент синтаксиса очень полезен в комбинации с предложением `ORDER BY`. СУБД MySQL позволяет ограничить вывод первыми n строками. Следующий запрос возвращает пять имен президентов, которым посчастливилось родиться первыми:

```
mysql> SELECT last_name, first_name, birth FROM president
    -> ORDER BY birth LIMIT 5;
+------------+------------+------------+
| last_name  | first_name | birth      |
+------------+------------+------------+
| Washington | George     | 1732-02-22 |
| Adams      | John       | 1735-10-30 |
| Jefferson  | Thomas     | 1743-04-13 |
| Madison    | James      | 1751-03-16 |
| Monroe     | James      | 1758-04-28 |
+------------+------------+------------+
```

Изменив порядок сортировки на сортировку по убыванию ORDER BY birth DESC, пользователь получит список пяти самых молодых президентов.

```
mysql> SELECT last_name, first_name, birth FROM president
    -> ORDER BY birth DESC LIMIT 5;
+-----------+-------------+------------+
| last_name | first_name  | birth      |
+-----------+-------------+------------+
| Clinton   | William J.  | 1946-08-19 |
| Bush      | George W.   | 1946-07-06 |
| Carter    | James E.    | 1924-10-01 |
| Bush      | George H.W. | 1924-06-12 |
| Kennedy   | John F.     | 1917-05-29 |
+-----------+-------------+------------+
```

Предложение LIMIT может помочь выбрать записи из середины выборки. Для этого необходимо указать два значения. Первое значение – это начальная запись результирующего набора, а второе значение – количество строк, которые будут возвращены. Этот запрос аналогичен предыдущему, но с той лишь разницей, что он возвращает пять строк, начиная с одиннадцатой.

```
mysql> SELECT last_name, first_name, birth FROM president
    -> ORDER BY birth LIMIT 10, 5;
+-----------+------------+------------+
| last_name | first_name | birth      |
+-----------+------------+------------+
| Tyler     | John       | 1790-03-29 |
| Buchanan  | James      | 1791-04-23 |
| Polk      | James K.   | 1795-11-02 |
| Fillmore  | Millard    | 1800-01-07 |
| Pierce    | Franklin   | 1804-11-23 |
+-----------+------------+------------+
```

Для того чтобы выбрать произвольную строку из таблицы president, воспользуйтесь ORDER BY RAND() вместе с LIMIT.

```
mysql> SELECT last_name, first_name FROM president
    -> ORDER BY RAND() LIMIT 1;
+-----------+------------+
| last_name | first_name |
+-----------+------------+
| McKinley  | William    |
+-----------+------------+
```

Сортировка по результату такой формулы работает начиная с MySQL версии 3.23.2. Исходя из этого, вы должны создать столбец, содержащий случайные числа (подробности – в разделе "Подавление оптимизации" главы 4, "Оптимизация запросов").

Подсчет и присвоение имен выводимым значениям столбцов таблиц

Все предыдущие запросы генерировали результат посредством выборки значений из таблицы. СУБД MySQL также позволяет производить подсчет значений из выводимых столбцов. Последующий запрос оценивает простое выражение (константа) и более сложное выражение, включающее несколько арифметических операций и пару вызовов функций.

```
mysql> SELECT 17, FORMAT(SQRT(3*3+4*4),0);
+----+-------------------------+
| 17 | FORMAT(SQRT(3*3+4*4),0) |
+----+-------------------------+
| 17 | 5                       |
+----+-------------------------+
```

Выражения могут содержать ссылки на столбцы таблиц.

```
mysql> SELECT CONCAT(first_name, ' ',last_name),CONCAT(city,',',state)
    -> FROM president;
+----------------------------------+-------------------------+
| CONCAT(first_name,' ',last_name) | CONCAT(city, ', ',state) |
+----------------------------------+-------------------------+
| George Washington                | Wakefield, VA           |
| John Adams                       | Braintree, MA           |
| Thomas Jefferson                 | Albemarle County, VA    |
| James Madison                    | Port Conway, VA         |
...
```

Этот запрос сводит имена и фамилии президентов в одну строку, разделив их пробелом, места рождения объединены в одну строку, разделенную запятой.

Выражение, по которому вычисляется результат, отображается в шапке. Отсюда и большая длина столбца (это хорошо видно в предыдущем примере.) Преодолеть это неудобство помогает конструкция AS *name*, присваивающая столбцу псевдоним. В итоге можно получить более вразумительный результат.

```
mysql> SELECT CONCAT(first_name, ' ',last_name) AS Name,
    -> CONCAT(city, ', ',state) AS Birthplace
    -> FROM president;
+----------------------+-------------------------+
| Name                 | Birthplace              |
+----------------------+-------------------------+
| George Washington    | Wakefield, VA           |
| John Adams           | Braintree, MA           |
| Thomas Jefferson     | Albemarle County, VA    |
| James Madison        | Port Conway, VA         |
...
```

Если столбец содержит пробелы, их необходимо заключать в кавычки.

```
mysql> SELECT CONCAT(first_name, ' ',last_name) AS 'President Name',
    -> CONCAT(city, ', ',state) AS 'Place of Birth'
    -> FROM president;
+----------------------+-------------------------+
| President Name       | Place of Birth          |
+----------------------+-------------------------+
| George Washington    | Wakefield, VA           |
| John Adams           | Braintree, MA           |
| Thomas Jefferson     | Albemarle County, VA    |
| James Madison        | Port Conway, VA         |
...
```

Работа с датами

Принципиальный момент, о котором надо всегда помнить, — формат представления дат в СУБД MySQL таков, что год представляется первым. Так, дата 27 июля 2002 года представляется записью "2002-07-27". Она не может быть

представлена в виде записи "07-27-2002" или "27-07-2002", хотя это может быть более удобно.

СУБД MySQL позволяет производить с датами следующие действия:

- сортировка по дате (это мы уже видели);
- выбор по определенной дате или диапазону дат;
- выделение фрагментов даты (год, месяц или день);
- вычисление разницы между датами;
- вычисление относительной даты.

Вот несколько примеров таких операций.

Выборка по определенной дате, по точной дате, по сравнению с определенной датой, сравнение значения столбца типа DATE со значением, которым вы интересуетесь.

```
mysql> SELECT * FROM event WHERE date = '2002-10-01';
+------------+------+----------+
| date       | type | event_id |
+------------+------+----------+
| 2002-10-01 | T    |        6 |
+------------+------+----------+
mysql> SELECT last_name, first_name, death
    -> FROM president
    -> WHERE death >= '1970-01-01' AND death < '1980-01-01';
+-----------+------------+------------+
| last_name | first_name | death      |
+-----------+------------+------------+
| Truman    | Harry S.   | 1972-12-26 |
| Johnson   | Lyndon B.  | 1973-01-22 |
+-----------+------------+------------+
```

Для выборки по фрагменту даты также можно воспользоваться функциями YEAR(), MONTH(), DAYOFMONTH(). Например, так можно сделать выборку всех президентов, которые родились в один и тот же месяц, что и автор этой книги (в марте). Для этого произведем выборку всех записей с датами рождения, в которых значится третий месяц.

```
mysql> SELECT last_name, first_name, birth
    -> FROM president WHERE MONTH(birth) = 3;
+-----------+------------+------------+
| last_name | first_name | birth      |
+-----------+------------+------------+
| Madison   | James      | 1751-03-16 |
| Jackson   | Andrew     | 1767-03-15 |
| Tyler     | John       | 1790-03-29 |
| Cleveland | Grover     | 1837-03-18 |
+-----------+------------+------------+
```

Запрос может также содержать название месяца.

```
mysql> SELECT last_name, first_name, birth
    -> FROM president WHERE MONTHNAME(birth) = 'March';
+-----------+------------+------------+
| last_name | first_name | birth      |
+-----------+------------+------------+
| Madison   | James      | 1751-03-16 |
```

```
| Jackson    | Andrew     | 1767-03-15 |
| Tyler      | John       | 1790-03-29 |
| Cleveland  | Grover     | 1837-03-18 |
+------------+------------+------------+
```

Усложним запрос и сделаем выборку президентов, день рождения которых полностью совпадает с днем рождения автора.

```
mysql> SELECT last_name, first_name, birth
    -> FROM president WHERE MONTH(birth) = 3 AND DAYOFMONTH(birth) =
29;
+-----------+------------+------------+
| last_name | first_name | birth      |
+-----------+------------+------------+
| Tyler     | John       | 1790-03-29 |
+-----------+------------+------------+
```

Результат этого запроса напоминает список "Люди, родившиеся сегодня" из раздела "Развлечения" вашей любимой газеты. Для выборки президентов, родившихся сегодня, необходимо сравнить их дни рождения со значением CURDATE() (текущей даты).

```
SELECT last_name, first_name, birth
FROM president WHERE MONTH(birth) = MONTH(CURDATE())
AND DAYOFMONTH(birth) = DAYOFMONTH(CURDATE());
```

Для получения интервала между датами их можно вычитать одну из другой. Например, для выяснения того, кто из президентов жил дольше всех, необходимо из даты смерти вычесть дату рождения. Для этого с помощью функции TO_DAYS() преобразуем значения death и birth, произведем вычитание и разделим полученный результат на 365. Получим приблизительный возраст в годах.

```
mysql> SELECT last_name, first_name, birth, death,
    -> TO_DAYS(death) - TO_DAYS(birth) AS age
    -> FROM president WHERE death IS NOT NULL
    -> ORDER BY age DESC LIMIT 5;
+-----------+------------+------------+------------+-------+
| last_name | first_name | birth      | death      | age   |
+-----------+------------+------------+------------+-------+
| Adams     | John       | 1735-10-30 | 1826-07-04 | 33119 |
| Hoover    | Herbert C. | 1874-08-10 | 1964-10-20 | 32943 |
| Truman    | Harry S.   | 1884-05-08 | 1972-12-26 | 32373 |
| Madison   | James      | 1751-03-16 | 1836-06-28 | 31150 |
| Jefferson | Thomas     | 1743-04-13 | 1826-07-04 | 30397 |
+-----------+------------+------------+------------+-------+
```

Для преобразования возраста в днях в приблизительный возраст в годах разделим возраст на 365. (Функция FLOOR(), которой мы воспользовались в этом запросе, отсекает дробную часть от полученного числа лет.)

```
mysql> SELECT last_name, first_name, birth, death,
    -> FLOOR((TO_DAYS(death) - TO_DAYS(birth))/365) AS age
    -> FROM president WHERE death IS NOT NULL
    -> ORDER BY age DESC LIMIT 5;
+-----------+------------+------------+------------+-----+
| last_name | first_name | birth      | death      | age |
+-----------+------------+------------+------------+-----+
| Adams     | John       | 1735-10-30 | 1826-07-04 | 90  |
```

```
| Hoover     | Herbert C. | 1874-08-10 | 1964-10-20 | 90  |
| Truman     | Harry S.   | 1884-05-08 | 1972-12-26 | 88  |
| Madison    | James      | 1751-03-16 | 1836-06-28 | 85  |
| Jefferson  | Thomas     | 1743-04-13 | 1826-07-04 | 83  |
+------------+------------+------------+------------+-----+
```

В этом частном случае значения возраста соответствуют действительному возрасту на момент смерти. Так получается не всегда, поскольку год не всегда равен 365 дням. Для подсчета лет мы обычно берем разницу между годами, а потом вычитаем единицу, если день смерти раньше дня рождения.

```
mysql> SELECT last_name, first_name, birth, death,
    -> (YEAR(death) - YEAR(birth)) - IF(RIGHT(death,5) <
RIGHT(birth,5),1,0)
    -> AS age
    -> FROM president WHERE death IS NOT NULL
    -> ORDER BY age DESC LIMIT 5;
+------------+------------+------------+------------+-------+
| last_name  | first_name | birth      | death      | age   |
+------------+------------+------------+------------+-------+
| Adams      | John       | 1735-10-30 | 1826-07-04 |    90 |
| Hoover     | Herbert C. | 1874-08-10 | 1964-10-20 |    90 |
| Truman     | Harry S.   | 1884-05-08 | 1972-12-26 |    88 |
| Madison    | James      | 1751-03-16 | 1836-06-28 |    85 |
| Jefferson  | Thomas     | 1743-04-13 | 1826-07-04 |    83 |
+------------+------------+------------+------------+-------+
```

Выражение `IF()` выполняет тест календарных дней, который основывается на простом построчном сравнении пяти последних символов дат. Так получается по двум причинам. Во-первых, MySQL трактует даты как строки, если обрабатывать их строковыми функциями, в данном случае функцией `RIGHT()`, которая возвращает n крайних правых символов строки. Во-вторых, MySQL генерирует даты с ограниченным числом цифр в каждой из частей даты. Сравнение не будет работать, если в значениях дней и месяцев меньше 10 не будет предшествующего нуля.

Получение разности между датами полезно также для определения, какой период времени разделяет эти две даты. С помощью подобных вычислений можно напоминать членам "Исторической Лиги" о необходимости обновить свое членство. Вычислите разность между датой истечения срока членства и текущей датой. Если она не превышает допустимого значения, скоро понадобится обновление. Ниже приведен запрос, делающий выборку членов, которые должны обновить членство в течение 60 дней.

```
SELECT last_name, first_name, expiration FROM member
WHERE (TO_DAYS(expiration) - TO_DAYS(CURDATE())) < 60;
```

Для вычитания одной даты из другой можно использовать функции `DATE_ADD()` и `DATE_SUB()`, которые по дате и интервалу возвращают новую дату, например:

```
mysql> SELECT DATE_ADD('1970-1-1', INTERVAL 10 YEAR);
+---------------------------------------+
| DATE_ADD('1970-1-1', INTERVAL 10 YEAR) |
+---------------------------------------+
| 1980-01-01                            |
+---------------------------------------+
mysql> SELECT DATE_SUB('1970-1-1', INTERVAL 10 YEAR);
+---------------------------------------+
| DATE_SUB('1970-1-1', INTERVAL 10 YEAR) |
```

```
+--------------------------------------+
| 1960-01-01                           |
+--------------------------------------+
```

Запрос, приведенный выше в этом разделе, делал выборку президентов, умерших в 1970-х годах. Для задания даты окончания интервала использовались литералы. Перепишем этот запрос по-другому: укажем с помощью литерала начальную дату.

```
mysql> SELECT last_name, first_name, death
    -> FROM president
    -> WHERE death >= '1970-1-1'
    -> AND death < DATE_ADD('1970-1-1', INTERVAL 10 YEAR);
+-----------+------------+------------+
| last_name | first_name | death      |
+-----------+------------+------------+
| Truman    | Harry S.   | 1972-12-26 |
| Johnson   | Lyndon B.  | 1973-01-22 |
+-----------+------------+------------+
```

Запрос на обновление членства можно переписать с применением функции DATE_ADD():

```
SELECT last_name, first_name, expiration FROM member
WHERE expiration < DATE_ADD(CURDATE(), INTERVAL 60 DAY);
```

Ранее в этой главе был представлен запрос для определения тех пациентов стоматолога, кто забыл пройти регулярный профилактический осмотр.

```
SELECT last_name, first_name, last_visit FROM patient
WHERE last_visit < DATE_SUB(CURDATE(),INTERVAL 6 MONTH);
```

Понятен ли этот запрос теперь?

Соответствие шаблону

СУБД MySQL позволяет производить выборку значений, которые соответствуют определенному шаблону. При этом не надо задавать определенное значение. Для выборки можно использовать специальные операторы (LIKE и NOT LIKE) и указать строку, содержащую символы-заменители. Символ "_" соответствует значению "любой одиночный символ". Символ "%" соответствует значению "любая последовательность символов" (включая и пустую последовательность). Шаблоны, удовлетворяющие LIKE и NOT LIKE, не чувствительны к регистру.

Ниже приведен шаблон, соответствующий всем фамилиям, начинающимся с буквы "W" или "w".

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name LIKE 'W%';
+------------+------------+
| last_name  | first_name |
+------------+------------+
| Washington | George     |
| Wilson     | Woodrow    |
+------------+------------+
```

С другой стороны, это ошибочный шаблон:

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name = 'W%';
Empty set (0.00 sec)
```

Это типичная ошибка при работе с шаблонами. Такой запрос даст результат только в том случае, если существует столбец, в действительности содержащий "W%" или "w%".

Вот запрос на выборку фамилий, в которых встречается "W" или "w".

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name LIKE '%W%';
+------------+------------+
| last_name  | first_name |
+------------+------------+
| Washington | George     |
| Wilson     | Woodrow    |
| Eisenhower | Dwight D.  |
+------------+------------+
```

Запрос на выборку фамилий, которые содержат четыре буквы, выглядит так:

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name LIKE '____';
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Polk      | James K.    |
| Taft      | William H.  |
| Ford      | Gerald R    |
| Bush      | George H.W. |
| Bush      | George W.   |
+-----------+-------------+
```

СУБД MySQL имеет также и другую форму выборки по шаблону с помощью расширенных регулярных выражений. Регулярные выражения описаны в приложении В, "Операторы и функции".

Настройка и использование переменных SQL

MySQL версий 3.23.6 и выше позволяет настроить переменные при помощи результатов запросов, которые предоставляют возможность сохранить переменные для использования в последующих запросах. Предположим, вы хотите узнать, какие президенты родились до Эндрю Джексона. Для этого следует ввести его дату рождения в переменную, а затем выбрать президентов с датой рождения меньше даты, указанной в переменной.[1]

```
mysql> SELECT @birth := birth FROM president
    -> WHERE last_name = 'Jackson' AND first_name = 'Andrew';
+-----------------+
| @birth := birth |
+-----------------+
| 1767-03-15      |
+-----------------+
mysql> SELECT last_name, first_name, birth FROM president
    -> WHERE birth < @birth ORDER BY birth;
+------------+------------+------------+
| last_name  | first_name | birth      |
+------------+------------+------------+
```

[1]Эту проблему можно решить, используя запрос с объединением, но нам еще рановато создавать объединения. Кроме того, иногда проще использовать переменную.

```
| Washington | George     | 1732-02-22 |
| Adams      | John       | 1735-10-30 |
| Jefferson  | Thomas     | 1743-04-13 |
| Madison    | James      | 1751-03-16 |
| Monroe     | James      | 1758-04-28 |
+------------+------------+------------+
```

Переменным присваиваются имена при помощи синтаксиса @name, а значения — посредством оператора SELECT, используя выражение типа @name := value. Первый запрос ищет дату рождения Эндрю Джексона и присваивает значение переменной @birth. (Результат оператора SELECT отображается, так как присвоение результата переменной не приводит к игнорированию вывода запроса.) Второй запрос обращается к переменной и использует ее значение, чтобы найти президента с более ранней датой рождения. Переменные также могут назначаться оператором SET, несмотря на то, что в этом случае не важен вид оператора присваивания (он может иметь вид либо =, либо := .)

```
mysql> SET @one_week_ago = DATE_SUB(CURDATE(),INTERVAL 7 DAY);
mysql> SELECT CURDATE(), @one_week_ago;
+------------+---------------+
| CURDATE()  | @one_week_ago |
+------------+---------------+
| 2002-09-03 | 2002-08-27    |
+------------+---------------+
```

Получение итоговых результатов

Одна из самых полезных возможностей СУБД MySQL (как, впрочем, и всех прочих СУБД) — обработка огромных объемов данных и их суммирование. По мере ознакомления с СУБД MySQL она становится мощным союзником в деле получения итогов и сумм, особенно принимая во внимание тот факт, что это очень нудная, требующая больших временных затрат и являющаяся источником ошибок работа, особенно если выполнять ее вручную.

Простым примером этого может служить необходимость определения значений, которые представлены в таблице. Для удаления повторений воспользуемся ключевым словом DISTINCT. Например, ниже представлен перечень штатов, в которых когда-либо рождались президенты США.

```
mysql> SELECT DISTINCT state FROM president ORDER BY state;
+-------+
| state |
+-------+
| AR    |
| CA    |
| CT    |
| GA    |
| IA    |
| IL    |
| KY    |
| MA    |
| MO    |
| NC    |
| NE    |
| NH    |
| NJ    |
| NY    |
```

```
| OH    |
| PA    |
| SC    |
| TX    |
| VA    |
| VT    |
+-------+
```

О другом способе получения итогов очень легко догадаться — это использование функции COUNT(). Выражение COUNT(*) выдает количество строк, выбранных запросом. Если запрос не содержит предложения WHERE, выражение COUNT(*) будет подсчитывать количество строк, содержащихся в таблице.

Следующий запрос показывает, сколько всего членов в таблице членства "Исторической Лиги":

```
mysql> SELECT COUNT(*) FROM member;
+----------+
| COUNT(*) |
+----------+
|      102 |
+----------+
```

При наличии предложения WHERE функция COUNT(*) покажет, сколько строк удовлетворяет заданному критерию выборки.

```
mysql> SELECT COUNT(*) FROM event WHERE type = 'Q';
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
```

Функция COUNT(*) подсчитывает каждую выбранную строку. Функция COUNT(col_name) подсчитывает только строки, содержащие ненулевые значения в данном столбце. Следующий запрос отчетливо демонстрирует это различие:

```
mysql> SELECT COUNT(*), COUNT(email), COUNT(expiration) FROM member;
+----------+--------------+-------------------+
| COUNT(*) | COUNT(email) | COUNT(expiration) |
+----------+--------------+-------------------+
|      102 |           80 |                96 |
+----------+--------------+-------------------+
```

Отсюда видно, что всего в таблице member 102 записи, только 80 из них имеют адрес электронной почты. Также видно, что 6 членов имеют пожизненное членство. (Отсутствие значения в столбце expiration указывает на пожизненное членство, а так как у 96 из 102 записей есть значение, получается 6 пожизненных членов.)

В версиях выше 3.23.2 уже можно совмещать функцию COUNT(*) с ключевым словом DISTINCT для подсчета количества точных ответов, например, для подсчета количества штатов, в которых когда-либо рождались президенты.

```
mysql> SELECT COUNT(DISTINCT state) FROM president;
+-----------------------+
| COUNT(DISTINCT state) |
+-----------------------+
|                    20 |
+-----------------------+
```

Можно производить подсчеты и по отдельным категориям. Например, общее число учащихся по вашему предмету можно определить с помощью такого запроса:

```
mysql> SELECT COUNT(*) FROM student;
+----------+
| COUNT(*) |
+----------+
|       31 |
+----------+
```

Но сколько из них мальчиков и девочек?

```
mysql> SELECT COUNT(*) FROM student WHERE sex='f';
+----------+
| COUNT(*) |
+----------+
|       15 |
+----------+
mysql> SELECT COUNT(*) FROM student WHERE sex='m';
+----------+
| COUNT(*) |
+----------+
|       16 |
+----------+
```

Несмотря на то что такой метод применим, он не иллюстративен и не очень подходит для случая, когда столбцы имеют несколько различных значений. Предположим, что необходимо определить, сколько президентов родилось в каждом штате. В соответствии с вышеприведенным методом сначала с помощью запроса `SELECT DISTINCT state FROM president` нужно узнать, сколько таких штатов, а затем сделать запрос `SELECT COUNT(*)` для каждого полученного штата. Очевидно, что это излишне трудоемкая процедура.

К счастью, СУБД MySQL позволяет производить подсчет с помощью одного лишь запроса. Количество девочек и мальчиков можно одновременно подсчитать следующим образом:

```
mysql> SELECT sex, COUNT(*) FROM student GROUP BY sex;
+-----+----------+
| sex | COUNT(*) |
+-----+----------+
| F   |       15 |
| M   |       16 |
+-----+----------+
```

Аналогичный запрос покажет, сколько президентов родилось в каждом штате.

```
mysql> SELECT state, COUNT(*) FROM president GROUP BY state;
+-------+----------+
| state | COUNT(*) |
+-------+----------+
| AR    |        1 |
| CA    |        1 |
| CT    |        1 |
| GA    |        1 |
| IA    |        1 |
| IL    |        1 |
| KY    |        1 |
| MA    |        4 |
```

```
| MO    |         1 |
| NC    |         2 |
| NE    |         1 |
| NH    |         1 |
| NJ    |         1 |
| NY    |         4 |
| OH    |         7 |
| PA    |         1 |
| SC    |         1 |
| TX    |         2 |
| VA    |         8 |
| VT    |         2 |
+-------+-----------+
```

При таком подсчете нам поможет предложение GROUP BY, которое сообщает базе данных, каким образом группировать значения. Его отсутствие будет расценено СУБД MySQL как ошибка.

Использование *COUNT(*)* и *GROUP BY* для подсчета значений имеет ряд преимуществ при подсчете количества вхождений определенных значений столбцов:

- нет необходимости знать все значения, имеющиеся в столбце, значения которого подсчитываются;
- нет необходимости в серии последовательных запросов, вполне достаточно одного;
- так как результат получен одним запросом, он вполне поддается сортировке.

Первые два преимущества позволяют создавать запросы более простым способом. Третье преимущество также важно, поскольку позволяет гибко манипулировать полученными результатами. При задании предложения GROUP BY результаты группируются по группируемому столбцу. Но это совсем не исключает применения группировки с помощью ORDER BY в другом порядке. Например, нужно подсчитать количество президентов по штату, в котором они родились, но отсортировать их по убыванию, начиная со штата, в котором родилось больше всего президентов США.

```
mysql> SELECT state, COUNT(*) AS count FROM president
    -> GROUP BY state ORDER BY count DESC;
+-------+-------+
| state | count |
+-------+-------+
| VA    |     8 |
| OH    |     7 |
| MA    |     4 |
| NY    |     4 |
| NC    |     2 |
| VT    |     2 |
| TX    |     2 |
| SC    |     1 |
| NH    |     1 |
| PA    |     1 |
| KY    |     1 |
| NJ    |     1 |
| IA    |     1 |
| MO    |     1 |
| CA    |     1 |
```

```
| NE    |     1 |
| GA    |     1 |
| IL    |     1 |
| AR    |     1 |
| CT    |     1 |
+-------+-------+
```

Столбцу, который нужно отсортировать, но который является результатом вычислений, можно присвоить псевдонимы и в предложении ORDER BY ссылаться прямо на него. Это продемонстрировано в последнем запросе. Столбцу COUNT(*) присвоен псевдоним count. Можно ссылаться на такой столбец и по-другому: указав порядок его следования в выводимых данных. Перепишем этот запрос следующим образом:

```
SELECT state, COUNT(*) FROM president
GROUP BY state ORDER BY 2 DESC;
```

Я не нахожу читабельной ссылку на столбец по его позиции. К примеру, при добавлении, удалении или изменении порядка следования столбцов в этом случае нужно помнить о необходимости проверки предложения ORDER BY и изменить номер столбца, если он менялся. Псевдонимы не приводят к таким проблемам.

Это справедливо для предложения GROUP BY. При использовании предложения GROUP BY с вычисляемым столбцом ссылка на него может производиться как по его позиции в списке столбцов, так и по его псевдониму. Вот запрос, который определяет, сколько президентов США родилось в каждом месяце.

```
mysql> SELECT MONTH(birth) AS Month, MONTHNAME(birth) AS Name,
    -> COUNT(*) AS count
    -> FROM president GROUP BY Name ORDER BY Month;
+-------+-----------+-------+
| Month | Name      | count |
+-------+-----------+-------+
|     1 | January   |     4 |
|     2 | February  |     4 |
|     3 | March     |     4 |
|     4 | April     |     4 |
|     5 | May       |     2 |
|     6 | June      |     1 |
|     7 | July      |     4 |
|     8 | August    |     4 |
|     9 | September |     1 |
|    10 | October   |     6 |
|    11 | November  |     5 |
|    12 | December  |     3 |
+-------+-----------+-------+
```

С указанием позиций столбцов запрос будет иметь следующий вид:

```
SELECT MONTH (birth), MONTHNAME(birth), COUNT(*)
FROM president GROUP BY 2 ORDER BY 1;
```

Функцию COUNT() можно указать в комбинации с ORDER BY и LIMIT для того, чтобы, например, найти четыре наиболее широко представленных штата из таблицы president.

```
mysql> SELECT state, COUNT(*) AS count FROM president
    -> GROUP BY state ORDER BY count DESC LIMIT 4;
```

```
+-------+-------+
| state | count |
+-------+-------+
| VA    |     8 |
| OH    |     7 |
| MA    |     4 |
| NY    |     4 |
+-------+-------+
```

Если вы не хотите ограничивать вывод предложением LIMIT, анализируя только значение, выдаваемое счетчиком COUNT(*), используйте предложение HAVING. Предложение HAVING похоже на предложение WHERE тем, что оно указывает условия, которым должны соответствовать выводимые строки. Отличается же это предложение тем, что может ссылаться на результаты агрегирующих функций, таких как функция COUNT(). Следующий запрос покажет, какие штаты представляются двумя или более президентами:

```
mysql> SELECT state, COUNT(*) AS count FROM president
    -> GROUP BY state HAVING count > 1 ORDER BY count DESC;
+-------+-------+
| state | count |
+-------+-------+
| VA    |     8 |
| OH    |     7 |
| MA    |     4 |
| NY    |     4 |
| NC    |     2 |
| VT    |     2 |
| TX    |     2 |
+-------+-------+
```

Более общий случай применения этого типа запросов – поиск повторений (или поиск неповторяющихся значений с использованием HAVING count = 1).

Кроме COUNT(), существуют и другие агрегирующие функции. Функции MIN(), MAX(), SUM() и AVG() предназначены для вычисления минимума, максимума, суммы и среднего значений столбца соответственно. Их можно использовать одновременно. Вот запрос, который позволяет производить подсчеты различных характеристик для любого теста или викторины. Здесь видно, как много подсчетов можно произвести по любому из значений. (Некоторые учащиеся отсутствуют, и их оценки не попадают в подсчеты.)

```
mysql> SELECT
    -> event_id,
    -> MIN(score) AS minimum,
    -> MAX(score) AS maximum,
    -> MAX(score)-MIN(score)+1 AS range,
    -> SUM(score) AS total,
    -> AVG(score) AS average,
    -> COUNT(score) AS count
    -> FROM score
    -> GROUP BY event_id;
+----------+---------+---------+-------+-------+---------+-----+
| event_id | minimum | maximum | range | total | average |count|
+----------+---------+---------+-------+-------+---------+-----+
|        1 |       9 |      20 |    12 |   439 | 15.1379 |  29 |
|        2 |       8 |      19 |    12 |   425 | 14.1667 |  30 |
|        3 |      60 |      97 |    38 |  2425 | 78.2258 |  31 |
```

```
|         4 |         7 |        20 |        14 |       379 |  14.0370 |      27 |
|         5 |         8 |        20 |        13 |       383 |  14.1852 |      27 |
|         6 |        62 |       100 |        39 |      2325 |  80.1724 |      29 |
+-----------+-----------+-----------+-----------+-----------+----------+---------+
```

Конечно, информация может быть более осмысленной, когда известно, откуда получены значения – из викторины или из теста. Для получения этой информации необходимо также обратиться к таблице `event_table`. Мы вернемся к этому запросу в разделе "Выборка данных из нескольких таблиц". Работать с агрегирующими функциями очень просто, так как они достаточно мощны. Но можно легко получить ошибочный результат. Рассмотрим следующий запрос:

```
mysql> SELECT
    -> state AS State,
    -> AVG((TO_DAYS(death)-TO_DAYS(birth))/365) AS Age
    -> FROM president WHERE death IS NOT NULL
    -> GROUP BY state ORDER BY Age;
+-------+-----------+
| State | Age       |
+-------+-----------+
| KY    | 56.208219 |
| VT    | 58.852055 |
| NC    | 60.141096 |
| OH    | 62.866145 |
| NH    | 64.917808 |
| NY    | 69.342466 |
| NJ    | 71.315068 |
| TX    | 71.476712 |
| MA    | 72.642009 |
| VA    | 72.822945 |
| PA    | 77.158904 |
| SC    | 78.284932 |
| CA    | 81.336986 |
| MO    | 88.693151 |
| IA    | 90.254795 |
+-------+-----------+
```

Этот запрос производит выборку умерших президентов, группирует их по дате рождения, определяет их возраст на момент смерти, вычисляет средний возраст по штату, а затем сортирует полученные результаты по среднему возрасту. Другими словами, запрос определяет средний возраст по штатам на момент смерти для всех президентов США, которых уже нет в живых.

Что же он показывает? Он показывает только то, что вы умеете писать запросы. И ничего больше. Не все операции, которые можно осуществлять с базами данных, имеют смысл. Однако люди иногда впадают в состояние эйфории, когда вдруг понимают, что они могут "вытворять" с базами данных. В результате этого в последние годы особенно бросается в глаза бешеный рост экзотерической (и совершенно бесцельной) спортивной статистики. Спортивные статистики с помощью своих баз данных вытягивают на свет все что нужно и не нужно знать о спортивных клубах. Неужели кого-то действительно может заинтересовать рекорд по перехватам мяча правого полузащитника в момент, когда его клуб вел с разницей в два мяча?

Выборка данных из нескольких таблиц

Все предыдущие запросы, рассмотренные нами, были получены в результате манипуляций с одной таблицей. А теперь мы приблизились к очень интересному

моменту нашего повествования. Я уже упоминал, что основное преимущество реляционных СУБД заключается в возможности ссылаться на различные объекты базы данных, что позволяет совмещать информацию и отвечать на вопросы, ответы на которые на основании данных из одной таблицы просто невозможны. В этом разделе рассказывается, как писать запросы, которые позволят это сделать.

При выборке информации из нескольких таблиц пользователь выполняет операцию, которая называется *объединение* (*join*). Такой термин здесь применяется из-за того, что результат такого запроса получен в результате объединения информации, взятой из одной таблицы, с информацией, взятой из другой таблицы.

Рассмотрим пример. Ранее в разделе "Таблицы проекта "Учет успеваемости" запрос такого рода был представлен без объяснений. Теперь пришло время и для них. В действительности это объединение является трехуровневым объединением. Таким образом, разработаем его в два этапа.

На первом этапе создадим запрос выборки результатов тестов для определенной даты:

```
mysql> SELECT student_id, date, score, type
    -> FROM event, score
    -> WHERE date = '2002-09-23'
    -> AND event.event_id = score.event_id;
+------------+------------+-------+------+
| student_id | date       | score | type |
+------------+------------+-------+------+
|          1 | 2002-09-23 |    15 | Q    |
|          2 | 2002-09-23 |    12 | Q    |
|          3 | 2002-09-23 |    11 | Q    |
|          5 | 2002-09-23 |    13 | Q    |
|          6 | 2002-09-23 |    18 | Q    |
...
```

Запрос выполняется так: сначала делается выборка записи события по заданной дате, а идентификатор события используется для поиска результатов по тому же идентификатору события. В результате выполнения запроса отображаются идентификатор учащегося, результат, дата и тип события всех записей с совпадающими записями событий.

Этот запрос отличается от всех уже записанных нами по двум важным аспектам:

- поиск производится из нескольких таблиц, и в предложении FROM перечислены две таблицы:

  ```
  FROM event, score
  ```

- в предложении WHERE определено, что таблицы event и score объединяются по значениям столбца event_id:

  ```
  WHERE ... event.event_id = score.event_id
  ```

Обратите внимание на синтаксис обращения к столбцу вида *tbl_name.col_name*. В нем обязательно указано имя таблицы. (Столбец event_id присутствует в обеих таблицах, поэтому здесь упоминание имени столбца без имени таблицы приводит к неоднозначности.)

Другие столбцы, участвующие в запросе (date, score, type), можно указывать без имени таблицы. Для этого запроса они уникальны, и их упоминание не вызывает неоднозначности. Однако я предпочитаю указывать имена таблиц в за-

просах. Это делает его более прозрачным. В полностью квалифицированном запросе мы получим:

```
SELECT score.student_id, event.date, score.score, event.type
FROM event, score
WHERE event.date = '2002-09-23'
AND event.event_id = score.event_id;
```

Первый запрос использует таблицу event для отображения даты в столбце event ID, а затем, чтобы найти соответствующие баллы в таблице scores, использует идентификационный номер. Вывод запроса содержит значения таблицы student_id. Однако очевидно, что имена были бы выразительнее. При помощи таблицы student мы сможем выводить идентификационные номера учеников, что представляет собой второй шаг. Это можно осуществить с помощью столбца student_id, присутствующего в таблицах score и student. Получим запрос следующего вида:

```
mysql> SELECT student.name, event.date, score.score, event.type
    -> FROM event, score, student
    -> WHERE event.date = '2002-09-23'
    -> AND event.event_id = score.event_id
    -> AND score.student_id = student.student_id;
+-----------+------------+-------+------+
| name      | date       | score | type |
+-----------+------------+-------+------+
| Megan     | 2002-09-23 |    15 | Q    |
| Joseph    | 2002-09-23 |    12 | Q    |
| Kyle      | 2002-09-23 |    11 | Q    |
| Abby      | 2002-09-23 |    13 | Q    |
| Nathan    | 2002-09-23 |    18 | Q    |
...
```

Этот запрос отличается от предыдущего следующим.

- В предложение FROM добавляется таблица student. Она необходима в дополнение к таблицам event и score.
- Теперь столбец student_id без упоминания имени таблицы приводит к неоднозначности. Необходима нотация score.student_id или student.student_id, чтобы конкретизировать столбец. (Это справедливо, даже если вы имеете привычку не именовать таблицы в запросах-объединениях.)
- В предложение WHERE добавляется условие соответствия записей таблиц score и student по значениям столбцов student_id:

  ```
  WHERE ... score.student_id = student.student_id;
  ```

- Запрос выводит имя учащегося, а не его идентификатор. (Конечно, при желании можно выводить оба поля сразу.)

С помощью этого запроса можно запросить любую дату и получить результаты по ней. Совсем не обязательно знать идентификационные номера учащихся или экзаменов. СУБД MySQL автоматически берет на себя всю заботу об определении соответствующих значений и их использовании для поиска соответствия между строками таблиц.

Еще одной задачей проекта учета успеваемости является учет посещаемости. Отсутствие учащихся на занятиях регистрируется по идентификатору учащихся и дате,

хранящимся в таблице absence. Для того чтобы получить имена учащихся (а не только их идентификаторы), нам необходимо объединить таблицы absence и student. Это можно сделать по значению student_id. Вот запрос, выводящий идентификатор учащихся и их имена вместе с суммой занятий, пропущенных учащимися:

```
mysql> SELECT student.student_id, student.name,
    -> COUNT(absence.date) as absences
    -> FROM student, absence
    -> WHERE student.student_id = absence.student_id
    -> GROUP BY student.student_id;
+------------+-------+----------+
| student_id | name  | absences |
+------------+-------+----------+
|          3 | Kyle  |        1 |
|          5 | Abby  |        1 |
|         10 | Peter |        2 |
|         17 | Will  |        1 |
|         20 | Avery |        1 |
+------------+-------+----------+
```

Совет

Несмотря на то что здесь в предложении GROUP BY указано имя таблицы, это совсем не обязательно. Предложение GROUP BY имеет отношение к столбцам, указанным в списке выборки (в первых двух строках запроса). А там указан только один столбец student_id, поэтому СУБД MySQL "знает", какая таблица имеется в виду.

Этот запрос выдает отличный результат тогда, когда мы стремимся узнать имена учащихся, которые имеют пропуски занятий. Но если такой список предоставить педагогическому совету, то может последовать совершенно естественный вопрос: "А как же остальные учащиеся? Нам необходимо оценить результаты и оставшихся учеников". Но это совсем другой вопрос. Это значит, что необходимо получить количество пропущенных занятий даже для учащихся, которые пропусков не имеют. Этот запрос отличается от предыдущего.

Для того чтобы на него ответить, воспользуемся конструкцией LEFT JOIN в предложении WHERE. Она указывает СУБД MySQL сделать выборку всех строк из таблицы, указанной в предложении слева (слева от слов LEFT JOIN). Назвав сначала таблицу student, мы получим перечень всех учащихся, даже тех, кто не представлен в таблице absence. Теперь запрос будет выглядеть следующим образом:

```
mysql> SELECT student.student_id, student.name,
    -> COUNT(absence.date) as absences
    -> FROM student LEFT JOIN absence
    -> ON student.student_id = absence.student_id
    -> GROUP BY student.student_id;
+------------+-----------+----------+
| student_id | name      | absences |
+------------+-----------+----------+
|          1 | Megan     |        0 |
|          2 | Joseph    |        0 |
|          3 | Kyle      |        1 |
|          4 | Katie     |        0 |
|          5 | Abby      |        1 |
|          6 | Nathan    |        0 |
|          7 | Liesl     |        0 |
...
```

Выше, в разделе "Получение итоговых результатов", был показан пример запроса, который выдавал цифровые данные на основе информации, содержащейся в таблице score. Результат этого запроса содержал идентификатор события, но не мог содержать дату получения результата теста или его тип. Теперь мы уже знаем, каким образом можно объединить таблицы score и event, чтобы получить даты и типы результатов.

```
mysql> SELECT
    -> event.date,event.type,
    -> MIN(score.score) AS minimum,
    -> MAX(score.score) AS maximum,
    -> MAX(score.score)-MIN(score.score)+1 AS range,
    -> SUM(score.score) AS total,
    -> AVG(score.score) AS average,
    -> COUNT(score.score) AS count
    -> FROM score, event
    -> WHERE score.event_id = event.event_id
    -> GROUP BY event.date;
+------------+------+---------+---------+-------+-------+---------+-------+
| date       | type | minimum | maximum | range | total | average | count |
+------------+------+---------+---------+-------+-------+---------+-------+
| 2002-09-03 | Q    |       9 |      20 |    12 |   439 | 15.1379 |    29 |
| 2002-09-06 | Q    |       8 |      19 |    12 |   425 | 14.1667 |    30 |
| 2002-09-09 | T    |      60 |      97 |    38 |  2425 | 78.2258 |    31 |
| 2002-09-16 | Q    |       7 |      20 |    14 |   379 | 14.0370 |    27 |
| 2002-09-23 | Q    |       8 |      20 |    13 |   383 | 14.1852 |    27 |
| 2002-10-01 | T    |      62 |     100 |    39 |  2325 | 80.1724 |    29 |
+------------+------+---------+---------+-------+-------+---------+-------+
```

Для получения суммарных значений на основании данных из столбцов различных таблиц можно также воспользоваться агрегатными функциями COUNT() и AVG(). Следующий запрос определяет число результатов и среднеарифметическое для всех комбинаций дат событий и пола учащихся:

```
mysql> SELECT event.date, student.sex,
    -> COUNT(score.score) AS count, AVG(score.score) AS average
    -> FROM event, score, student
    -> WHERE event.event_id = score.event_id
    -> AND score.student_id = student.student_id
    -> GROUP BY event.date, student.sex;
+------------+-----+-------+---------+
| date       | sex | count | average |
+------------+-----+-------+---------+
| 2002-09-03 | F   |    14 | 14.6429 |
| 2002-09-03 | M   |    15 | 15.6000 |
| 2002-09-06 | F   |    14 | 14.7143 |
| 2002-09-06 | M   |    16 | 13.6875 |
| 2002-09-09 | F   |    15 | 77.4000 |
| 2002-09-09 | M   |    16 | 79.0000 |
| 2002-09-16 | F   |    13 | 15.3077 |
| 2002-09-16 | M   |    14 | 12.8571 |
| 2002-09-23 | F   |    12 | 14.0833 |
| 2002-09-23 | M   |    15 | 14.2667 |
| 2002-10-01 | F   |    14 | 77.7857 |
| 2002-10-01 | M   |    15 | 82.4000 |
+------------+-----+-------+---------+
```

Аналогичный запрос можно сделать для проекта учета успеваемости. Например, для вычисления суммарного результата каждого учащегося в конце периода обучения:

```
SELECT student.student_id, student.name,
SUM(score.score) AS total, COUNT(score.score) AS n
FROM event, score, student
WHERE event.event_id = score.event_id
AND score.student_id = student.student_id
GROUP BY score.student_id
ORDER BY total;
```

Объединять можно не только различные таблицы, но и таблицу саму с собой. Например, можно определить, есть ли президенты, которые родились в одном и том же месте:

```
mysql> SELECT p1.last_name, p1.first_name, p1.city, p1.state
    -> FROM president AS p1, president AS p2
    -> WHERE p1.city = p2.city AND p1.state = p2.state
    -> AND (p1.last_name != p2.last_name OR p1.first_name !=
p2.first_name)
    -> ORDER BY state, city, last_name;
+-----------+------------+-----------+-------+
| last_name | first_name | city      | state |
+-----------+------------+-----------+-------+
| Adams     | John Quincy | Braintree | MA    |
| Adams     | John       | Braintree | MA    |
+-----------+------------+-----------+-------+
```

Этот запрос имеет две особенности.

- Требуется ссылка на два экземпляра одной и той же таблицы. Для этого создаются два экземпляра одной и той же таблицы с различными псевдонимами (p1, p2).
- Запись каждого президента соответствует самой себе. Но они не нужны в отклике запроса. Вторая строка предложения WHERE позволяет избежать этого.

Аналогичный запрос делает выборку президентов, родившихся в один и тот же день. При этом нельзя сравнивать даты рождения непосредственно из таблицы. Для сравнения дня и месяца рождения воспользуемся функциями MONTH() и DAYOFMONTH().

```
mysql> SELECT p1.last_name, p1.first_name, p1.birth
    -> FROM president AS p1, president AS p2
    -> WHERE MONTH(p1.birth) = MONTH(p2.birth)
    -> AND DAYOFMONTH(p1.birth) = DAYOFMONTH(p2.birth)
    -> AND (p1.last_name != p2.last_name OR p1.first_name !=
p2.first_name)
    -> ORDER BY p1.last_name;
+-----------+------------+------------+
| last_name | first_name | birth      |
+-----------+------------+------------+
| Harding   | Warren G.  | 1865-11-02 |
| Polk      | James K.   | 1795-11-02 |
+-----------+------------+------------+
```

Запрос можно упростить, если вместо комбинации функций MONTH() и DAYOFMONTH() воспользоваться функцией DAYOFYEAR(), но это даст некорректные результаты при сравнении дат високосных и невисокосных годов.

Вот таким образом объединение позволяет манипулировать информацией из таблиц, имеющих некую смысловую взаимосвязь, но логику этих связей знает только сам создатель запроса. СУБД MySQL не "знает", есть ли вообще что-то общее между объединенными таблицами. Например, можно объединить таблицы event и president для того, чтобы определить, проводились ли какие-либо викторины или тесты на тему дней рождения президентов.

```
mysql> SELECT president.last_name, president.first_name,
    -> president.birth, event.type
    -> FROM president, event
    -> WHERE MONTH(president.birth) = MONTH(event.date)
    -> AND DAYOFMONTH(president.birth) = DAYOFMONTH(event.date);
+-----------+------------+------------+------+
| last_name | first_name | birth      | type |
+-----------+------------+------------+------+
| Carter    | James E.   | 1924-10-01 | T    |
+-----------+------------+------------+------+
```

Оказывается, такое возможно. Ну и что? А то, что СУБД MySQL может выдавать результаты независимо от того, имеют они смысл или абсолютно бессмысленны. Результаты запроса необязательно имеют пользу и какой-то смысл только потому, что они получены с помощью компьютера. К счастью, человек все еще должен думать и понимать, что он делает.

Удаление и модификация существующих записей

Иногда требуется избавиться от определенных записей или изменить их содержимое. Для этой цели предназначены операторы DELETE и UPDATE.

Оператор DELETE имеет следующий синтаксис:

```
DELETE FROM tbl_name
WHERE which records to delete;
```

Предложение WHERE определяет, какая именно запись удаляется. Она может опускаться, но в этом случае будут удалены все записи в таблице! Это значит, что чем проще оператор DELETE, тем он "опаснее":

```
DELETE FROM tbl_name;
```

Этот запрос стирает все содержимое таблицы. *Будьте бдительны!*

Чтобы при удалении определенных записей указать их, воспользуйтесь предложением WHERE. Совсем так, как в операторе SELECT. Например, для того чтобы удалить из таблицы president всех президентов, родившихся в штате Огайо, выведите следующую команду:

```
mysql> DELETE FROM president WHERE state='OH';
Query OK, 7 rows affected (0.00 sec)
```

Перед использованием предложения WHERE в операторе DELETE советую проверить его работу на операторе SELECT. Не мешает убедиться, какие записи будут удалены оператором DELETE. Предположим, что нужно удалить запись о президенте Теодоре Рузвельте. Выполнит ли следующий запрос эту задачу?

```
DELETE FROM president WHERE last_name='Roosevelt';
```

Да, он выполнит в том плане, что удалит запись, которую вы задумали удалить. Нет, поскольку он также удалит запись о Франклине Рузвельте. Безопаснее предварительно использовать предложение WHERE с оператором SELECT:

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name='Roosevelt';
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Roosevelt | Theodore    |
| Roosevelt | Franklin D. |
+-----------+-------------+
```

Этот результат показывает, что требуется более серьезная детализация запроса:

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name='Roosevelt' AND first_name='Theodore';
+-----------+------------+
| last_name | first_name |
+-----------+------------+
| Roosevelt | Theodore   |
+-----------+------------+
```

Теперь нам известно правильное предложение WHERE. Таким образом, правильный оператор будет иметь вид

```
mysql> DELETE FROM president
    -> WHERE last_name='Roosevelt' AND first_name='Theodore';
```

Правда, операция удаления записи требует много предварительной работы? Семь раз отмерь, один раз отрежь! (На этом не стоит экономить. Сэкономить можно на копировании, вставке или методах редактирования строк. Более подробно об этом читайте в разделе "Как работать с mysql".)

Для модификации уже существующих записей можно воспользоваться оператором UPDATE, который имеет следующий синтаксис:

```
UPDATE tbl_name
SET which columns to change
WHERE which records to update;
```

Здесь предложение WHERE ведет себя так же, как и предложение WHERE для оператора DELETE. Оно может опускаться, но в этом случае *изменения будут сделаны во всех записях таблицы*! Вот запрос, который изменит имена всех учащихся из таблицы student на имя "George":

```
mysql> UPDATE student SET name='George';
```

Очевидно, что необходимо быть предельно осторожными с такими запросами.

К записям, которые подвергаются модификации, необходимо подходить более детально. Предположим, что недавно в "Историческую Лигу" вступил новый член. При этом были заполнены только отдельные столбцы:

```
mysql> INSERT INTO member (last_name,first_name)
    -> VALUES('York','Jerome');
```

После этого стало понятно, что была пропущена дата истечения срока членства. Это легко исправить с помощью оператора UPDATE, который содержит соответствующее предложение WHERE, чтобы определить, какую запись надо изменить:

```
mysql> UPDATE member
    -> SET expiration='2001-7-20'
    -> WHERE last_name='York' AND first_name='Jerome';
```

Можно модифицировать несколько столбцов одновременно. Этот запрос изменит адреса электронной и обычной почты одновременно:

```
mysql> UPDATE member
    -> SET email='jeromey@aol.com',street='123 Elm St',city='Anytown',
    -> state='NY',zip='01003'
    -> WHERE last_name='York' AND first_name='Jerome';
```

Можно присвоить значениям столбцов пустое значение (если столбец имеет такую возможность). Допустим, Джером решил заплатить большой членский взнос, что позволяет ему стать пожизненным членом. Это можно зафиксировать, установив дату истечения срока полномочий в NULL (никогда не наступает):

```
mysql> UPDATE member
    -> SET expiration=NULL
    -> WHERE last_name='York' AND first_name='Jerome';
```

Для оператора UPDATE, так же как и для оператора DELETE, верно утверждение, что нужно проверять правильность предложения WHERE с помощью оператора SELECT. Если ваш критерий выборки слишком широк, то модифицируется слишком много записей, если слишком узок — слишком мало записей.

Тем читателям, которые в процессе ознакомления с этими разделами выполняли все описанные шаги на тестовой базе данных sampdb, перед тем, как перейти к следующему разделу, рекомендуем отменить все изменения, сделанные в таблицах базы данных. Проще всего это сделать, перезагрузив все таблицы в соответствии с рекомендациями, данными в разделе "Дополнение таблиц".

Как работать с mysql

В этом разделе описываются приемы эффективной работы с клиентской программой mysql. Показано, как можно просто подключиться к серверу и вводить запросы, копируя их из одного черновика, а также как изменить приглашение, если вам не нравится стандартное.

Упрощение процесса подключения

Очевидно, что при вызове mysql необходимо указать такие параметры подключения, как имя узла, имя пользователя или пароль. Эта процедура требует длительного ввода и очень быстро утомляет. Ниже перечислено несколько способов, упрощающих эту процедуру.

- Храните параметры подключения в конфигурационном файле.
- Воспользуйтесь возможностью повторения команд из журнала регистрации.
- С помощью псевдонима или сценария оболочки задайте ярлык командной строки mysql.

Использование конфигурационных файлов

Начиная с версии 3.22 появилась возможность сохранять параметры соединения в конфигурационном файле. В этом случае нет необходимости вводить

параметры каждый раз при запуске mysql. Эти параметры используются также и в других клиентских приложениях, таких как, например, mysqlimport

Конфигурационный файл упростит процесс ввода команд не только для клиента mysql, но и для других программ.

В ОС UNIX создадим файл ~/.my.cnf (т.е. файл под именем my.cnf в вашем корневом каталоге). В ОС Windows создадим файл с названием my.cnf в корневой папке диска C или файл my.ini в папке System (C:\my.cnf или %SYSTEM%\my.ini). Конфигурационный файл является текстовым файлом, поэтому вы можете создавать его и редактировать в любом текстовом редакторе. Его содержание должно быть похоже на следующее:

```
[client]
host=server_host
user=your_name
password=your_pass
```

Строка [client] идентифицирует начало группы параметров данного клиента. Каждая последующая строка считывается СУБД MySQL как значение параметров клиентской программы. Завершается информация концом файла или началом группы параметров другого клиента. Здесь нужно заменить *server_host*, *your_name* и *your_pass* именем вашего сервера, вашим именем и паролем. Например, мой .my.cnf выглядит следующим образом:

```
[client]
host=cobra.snake.net
user=sampadm
password=secret
```

Здесь является обязательной только строка [client]. Строки, определяющие значения параметров, могут быть опущены; можно указывать только те, которые необходимы. Например, если ваше имя пользователя в базе данных совпадает с именем пользователя ОС UNIX, то его можно не указывать и не включать в строку user.

После создания файла my.cnf рекомендуется ограничить к нему доступ. Делается это следующим образом:

```
% chmod 600 .my.cnf
% chmod u=rw,go-rwx .my.cnf
```

Более подробную информацию о конфигурационных файлах можно найти в приложении Д, "Программы MySQL".

Использование журнала регистрации команд

Такие оболочки, как csh, tcsh и bash, запоминают отработанные команды в журналах регистрации. Это позволяет вызывать команды непосредственно из журнала регистрации без их повторного ввода. Например, повторить уже один раз введенную команду mysql можно следующим образом:

```
% !my
```

Символ "!" сообщает оболочке, что необходимо найти последнюю команду, которая начинается с "my", и повторить ее так, как если бы ее ввели с клавиатуры. Некоторые оболочки позволяют просматривать историю вверх и вниз с помощью клавиш <↑> и <↓> (или комбинаций клавиш <Ctrl+P> и <Ctrl+N>). Таким образом выбирается нужная команда, после чего для ее активизации следует нажать клавишу

<Enter>. Оболочки tcsh и bash имеют такую возможность. Для того чтобы узнать больше о возможностях оболочки, просмотрите документацию и по ней.

Псевдонимы и сценарии

Присвоение псевдонима позволит вызвать большую командную строку с помощью небольшой команды. Например, создадим в оболочках csh и tcsh псевдоним sampdb, используя следующую командную строку:

```
alias sampdb 'mysql -h cobra.snake.net -p -u sampadm sampdb'
```

Оболочка bash имеет другой синтаксис для команды создания псевдонима:

```
alias sampdb='mysql -h cobra.snake.net -p -u sampadm sampdb'
```

Создание псевдонима уравняет в функциональности такие две команды:

```
% sampdb
% mysql -h cobra.snake.net -p -u sampadm sampdb
```

Совершенно очевидно, что первую командную строку значительно проще напечатать, чем вторую. Для того чтобы псевдоним активизировался при каждой регистрации, добавьте команду alias в ваш стартовый файл (например, .cshrc для оболочки csh или .bash_profile для оболочки bash).

В системе Windows можно создать ярлык, который указывает на программу mysql, а затем, чтобы указать параметры соединения, отредактировать свойства ярлыка.

Еще одной формой ярлыка является сценарий, который будет выполняться программой mysql. В ОС UNIX файл сценария, являющийся эквивалентом псевдонима sampdb, выглядит следующим образом:

```
#! /bin/sh
exec mysql -h cobra.snake.net -p -u sampadm sampdb
```

Для того чтобы этот сценарий, назовем его sampdb, заработал, необходимо сделать его исполнимым (используя команду chmod +x sampdb). Теперь можно, напечатав sampdb, запустить клиентское приложение mysql.

В ОС Windows для этой цели используются пакетные файлы. Назовем такой файл sampdb.bat и введем туда такую строку:

```
mysql -h cobra.snake.net -p -u sampadm sampdb
```

Этот файл можно запустить из консоли DOS или двойным щелчком на соответствующей пиктограмме.

Чтобы подключиться к нескольким узлам и иметь доступ к нескольким базам данных, можно создать сразу несколько псевдонимов или сценариев, каждый из которых будет активизировать mysql с различными параметрами.

Упрощение процесса создания запросов

Программа mysql – это очень удобный инструмент взаимодействия с базой данных, но ее интерфейс больше всего подходит для создания коротких запросов в одну строку. Программа mysql сама по себе ничего не "знает" о длинных запросах, занимающих несколько строк. Такие запросы очень трудно создавать. Очень огорчительно после длительного ввода запроса обнаружить, что он содержит синтаксическую ошибку и его надо переделать.

Вот несколько приемов, которые позволят избежать ненужного ввода и перепечатывания.

- Для ввода строк в `mysql` пользуйтесь текстовым редактором.
- Не забывайте о возможностях копирования и вставки.
- Работайте с `mysql` в пакетном режиме.

Для ввода строк в `mysql` пользуйтесь текстовым редактором

Программа `mysql` имеет встроенную библиотеку для редактирования. В процессе ввода командных строк ими можно манипулировать, вызывать уже введенные строки, повторять их и модифицировать. Это очень удобно. Если при вводе команды вы допустили ошибку, можно вернуться назад и в переделах строки внести исправление. Аналогичным образом можно повторить запрос, который оказался ошибочным. (Эта задача существенно упрощается, если запрос был введен одной строкой.)

В табл. 1.4 приведены наиболее часто используемые комбинации клавиш для работы с текстовым редактором `mysql`. Но кроме этих существует еще множество команд. С ними вы можете ознакомиться в разделе команд редактирования в справочнике `bash`, который доступен на Web-узле GNU проекта (`http://www.gnu.org/manual/`).

Таблица 1.4. Команды редактора программы `mysql`

Комбинации клавиш	Назначение
<↑> или <Ctrl+P>	Отобразить предыдущую строку
<↓> или <Ctrl+N>	Отобразить следующую строку
<←> или <Ctrl+B>	Передвинуть курсор влево (назад)
<→> или <Ctrl+F>	Передвинуть курсор вправо (вперед)
<Escape>, <Ctrl+ B>	Передвинуть курсор назад на одно слово
<Escape>, <Ctrl+ F>	Передвинуть курсор вперед на одно слово
<Ctrl+ A>	Передвинуть курсор в начало строки
<Ctrl+ E>	Передвинуть курсор в конец строки
<Ctrl+ D>	Удалить символ, находящийся под курсором
<Delete>	Удалить символ, находящийся слева от курсора
<Escape D>	Удалить слово
<Escape>, <Backspace>	Удалить слово, находящееся слева от курсора
<Ctrl+K>	Удалить все символы от курсора до конца строки
<Ctrl+_>	Отменить последнее действие (может быть повторено)

Вот пример, который отлично иллюстрирует простоту редактирования вводимых строк. Предположим, что была введена следующая ошибочная командная строка:

```
mysql> SHOW COLUMNS FROM persident;
```

Допустим, вы заметили ошибку в слове president, до того как нажали клавишу <Enter>. С помощью комбинации клавиш <Ctrl+B> или клавиши <←> переместите

курсор до символа "s". Затем дважды нажмите клавишу <Delete>, чтобы удалить "er", и введите "e". Нажмите клавишу <Enter>. Теперь допустим, что ошибка была обнаружена после нажатия клавиши <Enter>. В этом случае после диагностического сообщения нажмите клавишу <↑> или комбинацию клавиш <Ctrl+P>. После того как ошибочная строка будет выведена на экран, внесите изменения в строку вышеуказанным способом.

Возможность редактирования вводимой командной строки отсутствует в обычной версии `mysql` для ОС Windows. (В системе Windows NT `mysql` поддерживается переход между командами с помощью стрелок, но не более того.) Чтобы иметь возможность редактировать команды, используйте программу `mysqlc`, которая идентична программе `mysql`, но работает с библиотеками `Cygnus`, поддерживающими редактирование строк.

Чтобы проверить, правильно ли установлена `mysqlc`, обратитесь к приложению Д, "Программы MySQL".

Пользуйтесь возможностями копирования и вставки

Работая в оконной среде, можно сохранять и воспроизводить нужные запросы или их отдельные части с помощью обычных операций копирования и вставки. Вот примерная последовательность действий.

1. Вызовите `mysql` в окне терминала или в консоли DOS.
2. Откройте в окне текстового редактора файл, содержащий запросы. (Например, `BBEdit` — в операционной системе Mac OS и `vi` — в окне `xterm` под X Window в ОС UNIX).
3. Для того чтобы вызвать запрос, хранящийся в файле, выделите и скопируйте его. Затем перейдите в окно `Telnet` или консоль DOS и вставьте запрос в `mysql`.

Как видите, процедура достаточно громоздкая в печатном виде, но это самый легкий путь быстрого ввода запросов без изнурительного перепечатывания.

Этот метод также позволяет не только редактировать запросы в окне текстового редактора, но там же создавать новые запросы, просто копируя и вставляя фрагменты старых запросов. Предположим, вы часто выбираете строки из определенной таблицы, но при этом их необходимо сортировать различными способами. Для этого можно хранить разные предложения `ORDER BY` и по необходимости просто их копировать.

Операцией копирования и вставки можно пользоваться и в обратном направлении (из окна терминала — в файл запросов). В процессе работы с `mysql` все команды сохраняются в файле `.mysql_history`, находящемся в корневом каталоге текущего клиента. Любой ранее введенный запрос можно повторить из файла `.mysql_history`, скопировав его из этого файла в файл запроса.

Работайте с `mysql` в пакетном режиме

Совсем не обязательно работать с программой `mysql` в интерактивном режиме. Программа `mysql` имеет возможность считывать командные строки из пакетного файла в неинтерактивном (пакетном) режиме. Это удобно для работы с за-

просами, которые нужно запускать с определенной периодичностью, потому что затруднительно вводить командную строку всякий раз, когда требуется запустить этот запрос. Значительно проще записать их в файл один раз, а потом периодически обращаться к нему.

Предположим, что существует запрос выборки членов "Исторической Лиги" по научным интересам. Например, для выборки членов, интересующихся временами "Великой Депрессии", необходимо написать следующий запрос:

```
SELECT last_name, first_name, email, interests FROM member
WHERE interests LIKE '%depression%'
ORDER BY last_name, first_name;
```

Сохраним этот запрос в файле `interests.sql` и попробуем запустить `mysql` из командной строки:

```
% mysql sampdb < interests.sql
```

По умолчанию `mysql` при работе в пакетном режиме осуществляет вывод без разделения столбцов табуляцией. Для получения вывода с разделением столбцов табуляцией необходимо указать ключ `-t`:

```
% mysql -t sampdb < interests.sql
```

Для сохранения результата работы запроса перенаправьте вывод в файл:

```
% mysql -t sampdb < interests.sql > output_file
```

Этот же запрос можно использовать для выборки членов "Исторической Лиги", интересующихся периодом правления президента Джефферсона. Замените в предложении `WHERE depression` на `Jefferson`. Затем запустите `mysql` повторно. Такой метод хорош и при редком обращении к запросу. При более частом обращении подойдет другой метод. Для того чтобы запрос стал более гибким, создайте его в виде параметризированного сценария. Параметр будет передавать переменное значение для предложения `WHERE`. Посмотрим, как данный метод работает.

```
#! /bin/sh
# interests.sh - поиск членов "Лиги" с определенными интересами
if [ $# -ne 1 ]; then echo 'Введите одно ключевое слово'; exit; fi
mysql -t sampdb <<QUERY_INPUT
SELECT last_name, first_name, email, interests FROM member
WHERE interests LIKE '%$1%'
ORDER BY last_name, first_name;
QUERY_INPUT
```

Третья строка проверяет, что в командной строке было введено одно ключевое слово. Выводится короткая строка, в противном случае работа завершается. Все, что находится между первым `<<QUERY_INPUT` и последним `QUERY_INPUT`, является вводом в `mysql`. Оболочка заменяет ссылку `$1` на значение, введенное в командной строке. (В сценариях оболочки `$1`, `$2` и т.д. обозначают аргументы команды.)

Перед тем как запустить запрос, его необходимо сделать исполнимым:

```
% chmod +x interests.sh
```

Теперь необходимость в редактировании запроса отпала. То, что вам требуется от запроса, можно определить командной строкой:

```
% interests.sh depression
% interests.sh Jefferson
```

Сценарий `interests.sh` вы найдете в папке `misc` дистрибутива `sampdb`. Его эквивалент, пакетный файл Windows `interests.bat`, находится там же.

Изменение приглашения `mysql`

В MySQL версии 4.0.2 вы можете изменить приглашение, если вам не нравится стандартное. Например, чтобы вписать имя текущей базы данных в приглашение, напишите команду PROMPT, а потом выберите разные базы данных, чтобы увидеть, насколько приглашение соответствует выбору:

```
% mysql
mysql> PROMPT \d>\_
PROMPT set to '\d>\_'
(none)> USE sampdb;
Database changed
sampdb> USE test;
Database changed
test>
```

За ключевым словом PROMPT следует желаемая строка запроса. Последовательности, которые начинаются с обратной косой черты, — это специальные параметры приглашения. Последовательности \d и _ объявляют текущее имя базы данных и пробел. Полный перечень параметров приведен в приложении Д, "Программы MySQL".

Что дальше?

Теперь нам кое-что известно о работе СУБД MySQL. Мы знаем, как создать базу данных и ее таблицы, умеем заносить данные в эти таблицы, производить различные выборки, модификации и удаления. Но это только начало. Это можно заметить по состоянию базы данных с примером. Мы только что создали базу данных и ее таблицы, занесли туда начальные данные, а также рассмотрели методы написания запросов, без которых невозможно обойтись, работая с базой данных.

К примеру, сейчас у нас есть на вооружении достаточно удобные способы ввода запросов, которые необходимы для ввода оценок в проекте учета успеваемости или добавления записей о новых членах "Исторической Лиги". Но, к сожалению, нет простых методов редактирования существующих записей. Неизвестно также, как печатать или отображать в интерактивном режиме список членов "Исторической Лиги". Эти и другие проблемы будут рассмотрены в последующих главах, в частности в главах 7 и 8.

То, к изучению какой главы вы перейдете сейчас, зависит от того, чем вы интересуетесь. Если вы хотите продолжить работу, начатую в проекте "Историческая Лига", или интересуетесь учетом успеваемости, то рекомендую перейти к части II, в которой описывается программирование в СУБД MySQL. Часть III сможет удовлетворить тех, кто интересуется проблемами администрирования. Но я рекомендую сначала набраться опыта в работе с

СУБД MySQL, прочитав до конца главы части I. В них описывается, как обрабатываются данные из баз данных, глубже проработан синтаксис и использование операторов SQL, приведены методы оптимизации запросов. Хорошие знания по этой теме станут фундаментом для дальнейшего использования СУБД MySQL независимо от контекста ее применения – работаете ли вы с `mysql`, пишете ли свои собственные программы или выполняете функции администратора.

2

Работа с данными в MySQL

В этой главе...

Типы данных СУБД MySQL
Типы столбцов СУБД MySQL
Обработка последовательностей
Поддержка кодировок
Выбор типа столбца
Вычисление выражений и преобразование типов

Практически все операции, которые производятся в СУБД MySQL, осуществляются над данными. Как бы там ни было, по определению, основное назначение СУБД заключается в том, чтобы управлять данными. Даже самая простая команда `SELECT 1` предполагает вычисление выражения и получение числового результата.

Все данные в MySQL характеризуются своим типом. Так, `37.4` — число (тип number), а `'abc'` — строка (тип string). Типы данных определяются на этапе создания таблицы с помощью оператора `CREATE TABLE`.

```
CREATE TABLE mytbl
(
  int_col INT,           /* столбец целого типа */
  str_col CHAR(20),      /* столбец строкового типа */
  date_col DATE,         /* столбец типа DATE */
);
```

Иногда типы данных задаются явным способом, как это происходит при ссылке на данные типа литералов при передаче или приеме данных функциями.

```
INSERT INTO mytbl (int_col, str_col, date_col)
VALUES(14,CONCAT('a', 'b'),20020115);
```

Здесь оператор `INSERT` выполняет следующие операции.

- Присваивает значение 14 целому столбцу `int_col`.
- Передает строковые данные `'a'` и `'b'` функции `CONCAT()`, которая возвращает значение `'ab'`. Потом это значение присваивается столбцу `str_col`.
- Присваивает целое значение `20020115` столбцу `date_col`. Как видно, при этом присвоении имеет место несоответствие типов. В таком случае СУБД MySQL автоматически преобразует целое число `20020115` в дату `'2002-01-15'`. Здесь видно, как автоматически преобразуются типы данных в СУБД MySQL.

Для эффективного использования СУБД MySQL необходимо глубокое понимание ее процессов обработки данных. В этой главе описаны типы данных, с которыми работает СУБД MySQL, и обсуждаются проблемы, возникающие при использовании этих типов данных.

- Все типы данных, которые может обрабатывать СУБД MySQL, включая пустое значение (`NULL`).
- Специальные типы данных СУБД MySQL и свойства, которыми они обладают. Некоторые типы данных СУБД MySQL, как, например, тип `BLOB`, являются общими для всех существующих СУБД. Другие, такие как параметры `AUTO_INCREMENT` и данные календарного типа `TIMESTAMP`, имеют некоторые особенности, которые необходимо хорошо понимать. Это поможет избежать непредвиденных результатов.
- Поддержка СУБД MySQL различных кодировок.
- Правильный выбор типов данных в таблицах. Необходимо хорошо понимать сам процесс выбора правильного типа данных на этапе создания таблиц.
- Правила вычисления выражений, принятые в СУБД MySQL. СУБД MySQL предлагает широкий набор операторов и функций, которые можно использовать в процессе поиска и отображения данных, а также манипулирования ими. Правила вычисления выражений включают в себя правила преобразования типов данных, когда данные одного типа можно использовать при контекстном поиске данных другого типа.

  Важно понимать, как происходит преобразование типов данных; некоторые преобразования бессмысленны, и в результате мы получаем абсолютно бессмысленные значения. Присвоение строки `'13'` числовому столбцу даст в результате значение `13`, а присвоение строки `'abc'` в этом же самом числовом столбце даст в результате значение `0`, так как строка `'abc'` никак не похожа на число. Возможны варианты и похуже: операция может нанести непоправимый вред, например удалить или модифицировать все строки таблицы, тогда когда вы собирались преобразовать только несколько строк.

Дополнительная информация о типах данных СУБД MySQL, операторах и функциях содержится в приложении Б, "Типы данных", и в приложении В, "Операторы и функции".

В примерах, приведенных в этой главе, довольно широко используется оператор `CREATE TABLE`. Этот оператор уже достаточно знаком вам, так как он использовался нами в обучающем разделе главы 1, "Знакомство с СУБД MySQL и

SQL". Полный синтаксис этой команды описан в приложении Г, "Синтаксис SQL". В нескольких примерах для внесения изменения в структуру таблиц используется команда ALTER TABLE. Этот оператор также обсуждается в приложении Г, "Синтаксис SQL", и главе 3, "Синтаксис и использование языка SQL".

СУБД MySQL поддерживает работу с таблицами нескольких типов, которые характеризуются различными свойствами. В отдельных случаях способ использования ими колонки определенного типа определяется типом таблицы. В этой главе типы таблиц определяются эпизодически, а более детальное описание имеющихся типов таблиц и их характеристики можно найти в главе 3, "Синтаксис и использование языка SQL".

Типы данных СУБД MySQL

В СУБД MySQL существует несколько типов данных. Это общие категории, по которым можно классифицировать данные.

Цифровые данные

Числами будем считать значения наподобие 48 или 193,62. СУБД MySQL различает целые числа (без дробной части) и вещественные числа (с дробной частью). Целые представляются в десятичном или шестнадцатеричном формате.

Целое число состоит из последовательности цифр, не содержащей десятичной точки. В цифровом контексте целое число можно представить шестнадцатеричной константой и обрабатывать ее как 64-битовое целое. Синтаксис задания шестнадцатеричных значений приводится ниже, в разделе "Строковые (символьные) данные", так как по умолчанию они рассматриваются как строки.

Вещественные числа состоят из последовательности цифр, разделенных точкой. Одна из этих последовательностей (но не все сразу) может быть пустой.

СУБД MySQL может работать и с научной нотацией чисел. Этот формат выглядит следующим образом: число в целом или вещественном формате, затем символ экспоненты "e" или "E", знак ("+" или "–") и целая экспоненциальная часть. 1,34E+12 или 43,27e–1 — это числа в правильной научной нотации. Однако 1,34E12 является *неправильной* научной нотацией, так как перед экспонентой отсутствует символ знака. Шестнадцатеричные числа не могут быть представлены в научной нотации: символ "e", предваряющий экспоненту, здесь является полноправным числом, и его появление в выражении приводит к неоднозначности.

Для обозначения отрицательных значений числа предваряются знаком "минус" (-).

Строковые (символьные) данные

Строками являются такие значения, как `'Madison, Wisconsin'` или `'patient shows improvement'`. Для обозначения строки она берется в одинарные или двойные кавычки. В качестве стандарта в ANSI SQL определяются одинарные кавычки, поэтому операторы, в которых они используются, будут более совместимы при переносе в другие СУБД.

Для обозначения специальных символов применяются управляющие последовательности (табл. 2.1). Последовательность начинается с символа обратной косой черты (`'\'`) для временного отключения от обычных правил интерпретации символов. Обратите внимание, что байт NUL — это не то же самое, что значение NULL. NUL — это байт с нулевым значением, а NULL символизирует отсутствие какого-либо значения вообще.

Таблица 2.1. Управляющие последовательности

Последовательность	Значение
\0	NUL (ASCII 0)
\'	Одинарная кавычка
\"	Двойная кавычка
\b	<Backspace>
\n	Новая строка
\r	Возврат каретки
\t	Табуляция
\\	Обратная косая черта
\z	<Ctrl+Z> (символ EOF в ОС Windows)

Итак, достаточно традиционная проблема включения кавычек в строку может быть решена следующим образом.

- Повторение символа кавычки, если это одинаковые кавычки.

```
'I can' 't'
"He said, ""I told you so."""
```

- Текст заключается в кавычки, отличные от кавычек, используемых в самом тексте. В этом случае повторять кавычки не нужно.

```
"I can't"
'He said, "I told you so."'
```

- Перед символом кавычки ставится обратная косая черта. В этом случае тип кавычек, заключающих строку, роли не играет.

```
'I can\'t'
"I can\'t"
"He said, \"I told you so.\""
'He said, \"I told you so.\"'
```

Для представления строковых переменных тоже может использоваться шестнадцатеричный формат. Существуют два различных синтаксиса представления таких констант. Первый состоит из `'0x'` с последующими одной и более шестнадцатеричными цифрами (от `'0'` до `'9'` и от `'a'` до `'f'`). Например, `'0x0a'` равно десятичному числу 10, а `'0x0ffff'` равно 65535. Нецифровые шестнадцатеричные коды могут быть представлены как в нижнем, так и в верхнем регистре. Но ведущая последовательность символов `'0x'` должна быть представлена только в нижнем регистре. Таким образом, значения `'0x0a'` и `'0x0A'` — правильные, а `'0X0a'` и `'0X0A'` — нет. В строковом контексте пара шестнадцатеричных цифр интерпретируется как коды ASCII, преобразованные в символы. В цифровом кон-

тексте шестнадцатеричная константа интерпретируется как число. В следующем примере проиллюстрированы оба применения:

```
mysql> SELECT 0x616263, 0x616263+0;
+----------+------------+
| 0x616263 | 0x616263+0 |
+----------+------------+
| abc      | 6382179    |
+----------+------------+
```

Начиная с версии 4.0 СУБД MySQL строковые данные также могут быть представлены с помощью записи `X'val'`, где *val* состоит из пары шестнадцатеричных цифр. Как и при записи `'0x'`, такие значения интерпретируются как строки, но могут быть использованы в цифровом контексте:

```
mysql> SELECT X'616263', X'616263'+0;
+-----------+-------------+
| X'616263' | X'616263'+0 |
+-----------+-------------+
| abc       | 6382179     |
+-----------+-------------+
```

В отличие от записи `'0x'`, запись `'X'` не чувствительна к регистру:

```
mysql> SELECT X'61', x'61';
+-------+-------+
| X'61' | x'61' |
+-------+-------+
| a     | a     |
+-------+-------+
```

Начиная с версии СУБД MySQL 4.1 и выше появилась возможность задания кодировки строковых значений. Более детально с проблемами, касающимися кодировок, можно познакомиться в разделе "Поддержка кодировок".

Дата и время (календарные данные)

Дата и время – это значения типа "2002-06-17" или "12:30:43". СУБД MySQL воспринимает также комбинированные значения типа дата/время, такие как "2002-06-17 12:30:43". Обратите внимание, что СУБД MySQL отображает даты в формате год-месяц-день. Этот факт удивляет новичков, начинающих работать с СУБД MySQL, несмотря на то, что этот формат является стандартом ANSI SQL. Отображать дату можно в любом формате. Для этих целей существует функция `DATE_FORMAT()`, но по умолчанию принят формат с предшествующим годом. Поэтому при вводе даты сначала вводится год.

Пустое значение (NULL)

`NULL` – это значение, находящееся вне всяких типов. Обычно оно используется для определения значений "нет значения", "неизвестное значение", "вне диапазона", "ничто из вышеперечисленного" и т.д. Пустые значения можно добавлять в таблицы, осуществлять поиск пустых значений по таблицам, проверять, является ли значение пустым. Арифметические операции над пустыми значениями невозможны. (Такое значение возвращает `NULL`.)

Типы столбцов СУБД MySQL

Таблица базы данных состоит из одного и более столбцов. При создании таблицы командой CREATE TABLE объявляются типы столбцов. Тип столбца – более детализированная сущность, чем тип данных, который укладывается в такие общие категории, как "число" и "строка". Тип столбца точно определяет конкретную сущность значений, которые может содержать данный столбец таблицы (например, SMALLINT или VARCHAR(32)).

Типы столбцов СУБД MySQL являются инструментом, с помощью которого проектировщик определяет данные, которые будет содержать создаваемая таблица. Например, если у вас есть числовые данные, то их можно хранить в столбцах строкового или числового типа, но СУБД MySQL будет рассматривать эти значения по-разному, в зависимости от того, как вы их храните. Тип столбцов имеет несколько характеристик в зависимости от следующих особенностей.

- Значения какого типа хранятся в них.
- Каков размер значений, и имеют ли эти значения фиксированную длину (для всех значений этого типа потребуется один и тот же объем памяти) или переменную длину (объем занимаемого пространства зависит от сохраняемого значения).
- Как значения данного типа сравниваются и сортируются.
- Позволяет ли этот тип хранить пустые значения.
- Может ли этот тип индексироваться.

Рассмотрим вкратце типы столбцов СУБД MySQL, а потом обсудим их подробнее.

Краткий обзор типов столбцов

СУБД MySQL имеет все типы столбцов для значений всех основных категорий типов данных за исключением пустых значений. Пустые значения распространяются на все типы данных, и то, могут или нет они принимать пустое значение, определяется атрибутом типа при объявлении типа столбца.

Целые и вещественные типы перечислены в табл. 2.2. Целые значения могут иметь знак (signed) или быть беззнаковыми (unsigned). Специальный параметр AUTO_INCREMENT позволяет автоматически генерировать значения столбцов. Этот параметр можно использовать в приложениях, требующих создания уникальной последовательности номеров.

Таблица 2.2. Числовые типы столбцов

Тип	Значение
TINYINT	Очень малое целое
SMALLINT	Малое целое
MEDIUMINT	Целое среднего размера
INT	Целое стандартного размера
BIGINT	Большое целое

Тип	Значение
FLOAT	Вещественное число одинарной точности
DOUBLE	Вещественное число двойной точности
DECIMAL	Вещественное число, представляемое строкой

Строковые типы СУБД MySQL приведены в табл. 2.3. Строка может содержать любое значение или произвольные двоичные данные, такие как изображения и звуки. Над строками можно выполнять операции сравнения. Строки можно проверять на соответствие образцу. СУБД MySQL позволяет производить проверку соответствия для любого типа данных, но чаще всего эта операция выполняется над строками.

Таблица 2.3. Строковые типы столбцов

Тип	Значение
CHAR	Строка фиксированной длины
VARCHAR	Строка переменной длины
TINYBLOB	Строка с большим двоичным объектом сверхмалого размера
BLOB	Строка с большим двоичным объектом малого размера
MEDIUMBLOB	Строка с большим двоичным объектом среднего размера
LONGBLOB	Строка с большим двоичным объектом большого размера
TINYTEXT	Строка с текстом сверхмалого размера
TEXT	Строка с текстом малого размера
MEDIUMTEXT	Строка с текстом среднего размера
LONGTEXT	Строка с текстом большого размера
ENUM	Перечисление; столбцам присваиваются элементы перечисления
SET	Множество; столбцам могут быть присвоены элементы множеств

Типы календарных данных, имеющиеся в СУБД MySQL, показаны в табл. 2.4. Здесь CC, YY, MM, DD, hh, mm и ss соответствуют веку, годам, месяцам, дням, часам, минутам и секундам. Для хранения календарных данных СУБД MySQL имеет типы для дат (со временем или без него), времени и временных отметок (специальный тип данных, предназначенный для отслеживания момента, когда было внесено последнее изменение записи в таблице). Если вам нужно хранить информацию только о годе, то СУБД MySQL и для этого случая имеет специальный тип данных, предназначенный для хранения годичных данных.

Таблица 2.4. Столбцы календарного типа

Тип	Значение
DATE	Дата в формате "CCYY-MM-DD"
TIME	Время в формате "hh:mm:ss"
DATETIME	Дата и время в формате "CCYY-MM-DD hh:mm:ss "
TIMESTAMP	Значение временной отметки в формате "CCYYMMDDhhmmss"
YEAR	Значение года в формате "CCYY"

Создание таблиц

Процедура создания таблиц заключается в запуске команды CREATE TABLE. В этой команде определяются имена столбцов, составляющих таблицу. Каждый столбец имеет свое имя и тип, с типом могут указываться различные атрибуты. Создадим таблицу mytbl, имеющую в своем составе столбцы f, c и i.

```
CREATE TABLE mytbl
(
   f FLOAT(10,4),
   c CHAR(15) NOT NULL DEFAULT "none",
   i TYNYINT UNSIGNED NULL
);
```

Синтаксис объявления столбца имеет вид

col_name col_type [col_attributes] [general_attributes]

Имя столбца задается в col_name, который всегда располагается в начале. Правила присвоения имен столбцам приведены в разделе "Правила присвоения имен в СУБД MySQL" главы 3, "Синтаксис и использование языка SQL". Вкратце можно сказать, что имя может содержать до 64 символов. Имя может состоять из совокупности алфавитно-цифровых символов в кодировке, принятой по умолчанию на сервере, символа подчеркивания и символа доллара ("_" и "$"). Имена функций (например, слова POS, MIN) не являются зарезервированными словами, и их применение в качестве имен допустимо. Но такие ключевые слова, как SELECT, DELETE и CREATE, являются зарезервированными, и их нельзя использовать в качестве имен столбцов, однако начиная с СУБД MySQL 3.23.6 имя может начинаться с любого допустимого символа, включая цифру. Имя может включать и другие символы или использовать зарезервированные слова, ограниченные символами "обратная одинарная кавычка" (``). Имя столбца может начинаться с любого допустимого символа, включая и цифру. Однако заключенное в обратные одинарные кавычки имя не может состоять полностью из цифр, так как это уже будет больше похоже на числовое значение.

Тип столбца *col_type* определяет тип значений, которые может принимать столбец. Спецификация типа также может определять максимальную длину значений, хранимых в столбце. Для некоторых типов длина определяется явным образом с указанием числа. В других случаях длина определяется именем типа. Например, CHAR(10) определяет длину столбца в 10 символов явным способом, а значения TINYBLOB всегда имеет длину 255 символов. Некоторые типы спецификаций позволяют определить максимальную длину отображения (сколько символов используется для отображения значений столбца). Типы вещественных столбцов позволяют указывать количество цифр в дробной части. Это позволяет задавать точность значений столбца.

После типа столбца определяются как необязательные параметры, зависящие от типа столбца, так и более общие параметры. Параметры действуют как модификаторы типов и вызывают изменение работы со столбцом.

- Параметры могут зависеть от типа столбца. Например, параметр UNSIGNED допустим только для целого типа, а параметр BINARY — только для типов CHAR и VARCHAR.

- Параметры общего типа присваиваются столбцу любого типа за несколькими исключениями. Можно присваивать параметры NULL и NOT NULL для того, чтобы определить, может ли столбец содержать пустые значения. Можно присваивать параметр DEFAULT *def_value* для того, чтобы показать, что при создании новой строки по умолчанию столбцу присваивается значение *def_value*, если при этом не указывается явное значение. Значение *def_value* должно быть константой, но не может быть выражением или ссылкой на другой столбец. Значение по умолчанию не определяется для столбцов типа BLOB и TEXT.

При указании нескольких параметров существуют определенные ограничения на порядок их следования. В целом можно сказать, что правильно указать такие типозависимые параметры, как UNSIGNED или ZEROFILL, перед такими общими параметрами, как NULL или NOT NULL.

Ниже в этой главе обсуждаются синтаксис объявления типов столбцов и свойства, характеризующие их, такие как их диапазон и требования к памяти. Спецификации типов приводятся в синтаксисе оператора CREATE TABLE. Необязательная информация выделяется квадратными скобками ([]). Например, синтаксис MEDIUMINT[(M)] означает, что максимальная ширина отображения, обозначенная как [(M)], может не указываться. Однако отсутствие квадратных скобок в нотации VARCHAR(M) свидетельствует о том, что объявление длины (M) для этого типа обязательно.

Цифровые типы столбцов

Цифровые типы данных в СУБД MySQL можно разделить на две основные категории.

- **Целочисленные типы**. Это числа, не имеющие дробной части (1, 43, –3, 0 или –798432). Целые столбцы можно использовать для хранения целых величин, таких как вес (до фунта), высота (до дюйма), число звезд Галактики, количество жильцов в доме или количество бактерий в чашке Петри.
- **Вещественные типы**. Это числа, имеющие дробную часть (3,14159; –,00273; –4,78; 39,3E+4). Столбцы вещественного типа можно использовать для хранения величин, имеющих дробную часть, или таких, значения которых предельно малы или, наоборот, велики. Вот пример некоторых данных, которые можно представить в вещественных величинах: средняя урожайность, расстояние, затраты и зарплата, уровни занятости или биржевые котировки. Этим столбцам можно присваивать целые значения, которые можно рассматривать как частный случай вещественного числа с дробной частью, равной нулю.

Целые типы – самые простые. Вещественные типы значительно сложнее, в частности, из-за того, что по мере развития СУБД MySQL их поведение в определенной мере претерпевает изменения. Вещественные значения могут присваиваться целым столбцам, но при этом они будут округляться до ближайшего целого. И наоборот, целые значения могут присваиваться вещественным столбцам. В этом случае их можно рассматривать как вещественные значения с дробной частью, равной нулю. В табл. 2.5 приведены наименования и диапазоны всех число-

вых типов данных, а в табл. 2.6 представлены требования к объему памяти, необходимой для хранения таких типов данных.

Таблица 2.5. Диапазоны значений цифровых данных

Тип	Диапазон
`TINYINT[(M)]`	Знаковые значения: от -128 до 127 (от -2^7 до 2^7-1)
	Беззнаковые значения: от 0 до 255 (от 0 до 2^8-1)
`SMALLINT(M)]`	Знаковые значения: от -32768 до 32767 (от -2^{15} до $2^{15}-1$)
	Беззнаковые значения: от 0 до 65535 (от 0 до $2^{16}-1$)
`MEDIUMINT[(M)]`	Знаковые значения: от -8388608 до 8388607 (от -2^{23} до $2^{23}-1$)
	Беззнаковые значения: от 0 до 16777215 (от 0 до $2^{24}-1$)
`INT[(M)]`	Знаковые значения: от -2147683648 до 2147483647 (от -2^{31} до $2^{31}-1$)
	Беззнаковые значения: от 0 до 4294967295 (от 0 до $2^{32}-1$)
`BIGINT[(M)]`	Знаковые значения: от -9223372036854775808 до 9223372036854775807 (от -2^{63} до $2^{63}-1$)
	Беззнаковые значения: от 0 до 18446744073709551615 (от 0 до 2^{64})
`FLOAT[(M,D)]`	Минимальное ненулевое значение: +- 1,175494351E-38
	Максимальное ненулевое значение: +- 3,402823466E+38
`DOUBLE[(M,D)]`	Минимальное ненулевое значение: +- 2,2250738585072014E-308
	Максимальное ненулевое значение: +- 1,7976931348623157E+308
`DECIMAL(M,D)`	Переменное; диапазон зависит от параметров *M* и *D*

Таблица 2.6. Требования к объемам памяти различных типов цифровых данных

Тип	Объем памяти
`TINYINT[(M)]`	1 байт
`SMALLINT[(M)]`	2 байта
`MEDIUMINT[(M)]`	3 байта
`INT[(M)]`	4 байта
`BIGINT[(M)]`	8 байт
`FLOAT[(M,D)]`	4 байта
`DOUBLE[(M,D)]`	8 байт
`DECIMAL(M,D)`	*M* байт (MySQL версии ниже 3.23) и *M*+2 байт (MySQL версии 3.23 и выше)

Целые типы столбцов

СУБД MySQL имеет пять целых типов: `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT` и `BIGINT`. Тип `INTEGER` имеет синоним `INT`. Различие между этими типами заключается в диапазоне величин, которые они представляют, и в объемах памяти, которые для них требуются. (Чем больше диапазон значений у типа данных, тем больше для него требуется памяти.) Целочисленные столбцы могут быть объявлены как `UNSIGNED`. Это исключит появление в значениях таких столбцов отрицательных величин. При этом диапазон значений сдвигается, и начальным

значением для такого типа будет значение 0. Объем памяти, необходимый для них, также становится другим. Естественно, что типы, имеющие больший диапазон величин, требуют больше памяти.

При объявлении целого типа задается длина отображения M. Диапазон этого параметра составляет от 1 до 255. Он определяет количество символов, предназначенных для отображения значений столбца. Например, столбец, объявленный как `MEDIUMINT(4)`, объявляет целый столбец типа `MEDIUMINT`, который будет отображаться четырьмя цифрами. Кроме того, длина может присваиваться по умолчанию. При объявлении целого столбца без длины присваивается значение по умолчанию. Это будет длина, максимальная для этого типа данных.

Необходимо заметить, что отображаемые значения не усекаются до значения M. Если длина отображаемого значения больше M символов, будет отображено все значение, а не его часть, усеченная до M.

Размер отображения M для целого столбца имеет отношение только к числу символов, которые используются для отображения значений, хранящихся в столбцах. Но он совершенно *ничего не значит* с точки зрения объема памяти, необходимого для хранения данных. Например, для хранения значений типа `BIGINT` требуется 8 байт памяти. И совершенно невозможно уменьшить необходимую память вдвое, объявив тип `BIGINT(4)`. Значение M никак не влияет на диапазон допустимых значений. Объявив столбец как `INT(3)`, вы не можете ограничить его максимальное значение значением 999.

Следующий оператор демонстрирует значения по умолчанию M и D для целочисленных типов данных:

```
CREATE TABLE mytbl
(
  itiny TINYINT,
  itiny_u TINYINT UNSIGNED,
  ismall SMALLINT,
  small_u SMALLINT UNSIGNED,
  imedium MEDIUMINT,
  imedium_u MEDIUMINT UNSIGNED,
  ireg INT,
  ireg_u INT UNSIGNED,
  ibig BIGINT,
  ibig_u BIGINT UNSIGNED,
);
```

Команда `DESCRIBE mytbl` возвратит следующий результат[1]:

```
+-----------+------------------------+
| Field     | Type                   |
+-----------+------------------------+
| itiny     | tinyint(4)             |
| itiny_u   | tinyint(3) unsigned    |
| ismall    | smallint(6)            |
| small_u   | smallint(5) unsigned   |
| imedium   | mediumint(9)           |
| imedium_u | mediumint(8) unsigned  |
| ireg      | int(11)                |
```

[1] Распечатка этого запроса возвращает для типа `BIGINT` значение 21 (а не 20) для версии MySQL ниже 3.23.

```
| ireg_u     | int(10) unsigned      |
| ibig       | bigint(20)            |
| ibig_u     | bigint(20) unsigned   |
+------------+-----------------------+
```

Вещественные типы столбцов

СУБД MySQL имеет три вещественных типа: FLOAT, DOUBLE, DECIMAL.

Эти типы имеют соответствующие синонимы: для типа DOUBLE синонимами являются типы DOUBLE PRECISION и REAL и NUMERIC для DECIMAL. Диапазоны значений для этих типов отличаются от диапазонов значений целочисленных типов в том смысле, что для них можно представить не только максимальное значение, но и минимальное ненулевое значение. Минимальные значения обеспечивают меру точности данного типа, что особенно важно при хранении научных данных. (И это, конечно, с учетом соответствующих отрицательных минимальных и максимальных значений.)

Вещественный тип может объявляться как UNSIGNED (но это несправедливо для типов FLOAT и DOUBLE до версии СУБД MySQL 4.0.2). В отличие от целочисленных типов, объявление вещественного типа как UNSIGNED не сдвигает диапазон типа вверх, а просто отсекает отрицательную часть. Точность данных определяется минимальными значениями, допустимыми для данного типа, что очень важно для научных приложений. (Между отрицательным минимумом и положительным максимумом есть определенное соответствие.)

Для вещественных данных задается максимальный размер отображения M символов (включая D символов его дробной части). Значение параметра M лежит в пределах от 1 до 255. Значение другого параметра — D — лежит в пределах от 0 до 30. Оно не должно превышать M-2. (Читателям, знакомым с концепцией ODBC, можно сказать, что в терминах ODBC M отражает "точность", а D — "масштаб".)

Для типов DOUBLE и FLOAT эти параметры можно не упоминать. Если они опущены, эти типы будут интерпретированы следующим образом.

- До версии 3.23.6 типы FLOAT и DOUBLE по умолчанию принимаются как FLOAT(10,2) и DOUBLE(16,4) где сохраненные данные округляются до 2 и 4 цифр после запятой соответственно.
- Для версии 3.23.6 и выше типы FLOAT и DOUBLE сохраняются с максимальной точностью, которой можно добиться с помощью данного оборудования.

Для типа DECIMAL значения M и D могут быть обязательными, но это полностью зависит от используемой версии СУБД MySQL.

- До версии 3.23.6 необходимо указание атрибутов M и D для столбцов типа DECIMAL.
- Для версии 3.23.6 и выше при отсутствии атрибута D его значение устанавливается равным 0. При отсутствии атрибута M его значение устанавливается по умолчанию равным 10. Другими словами, справедливо следующее равенство:

```
DECIMAL = DECIMAL(10) = DECIMAL(10,0)
DECIMAL(n) = DECIMAL(n,0)
```

Для совместимости с ODBC (Open Bata Base Connectivity) разрешен синтаксис LOAT(p). Однако предугадать точное поведение столбцов, объявленных с применением такого синтаксиса, достаточно сложно.

- До версии 3.23 допустимыми значениями атрибута p были значения 4 и 8, которые отражали объем памяти в байтах, необходимый для хранения одного значения. Типы FLOAT(4) и FLOAT(8) обрабатывались как типы FLOAT(10,2) и DOUBLE(16,4) с сохраненными значениями, округленными до 2 и 4 знаков после запятой соответственно.
- Для версий 3.23.0–3.23.5 допустимыми значениями атрибута p также являются значения 4 и 8, которые отражают количество байтов, но теперь FLOAT(4) и FLOAT(8) рассматриваются как столбцы с одинарной и двойной точностью соответственно, сохраненные с полной точностью, которую позволяет данное оборудование.
- Для версий начиная с 3.23.6 допустимое значение p может лежать в диапазоне от 0 до 53 и отражает минимальное количество битов, необходимых для сохранения значений. При значениях атрибута p от 0 до 24 столбец считается объявленным с одинарной точностью. При значениях p от 25 до 53 столбец считается объявленным с двойной точностью.

Еще больше удручает то, что СУБД MySQL до сих пор допускает в качестве синонимов типы FLOAT4 и FLOAT8, но являются ли они синонимами, зависит от спользуемой версии СУБД MySQL.

- До версии 3.23.6 типы FLOAT4 и FLOAT8 были равнозначны типам FLOAT(10,2) и DOUBLE(16,4).
- После версии 3.23.6 типы FLOAT4 и FLOAT8 стали равнозначны типам FLOAT и DOUBLE.

При более тщательном сравнении типов FLOAT4 и FLOAT8 с типами FLOAT(4) FLOAT(8) обратите внимание, что типы FLOAT4 и FLOAT8 не *совсем аналогичны* ипам FLOAT(4) и FLOAT(8), хотя на первый взгляд кажется, что это не так.

Проверка того, как СУБД MySQL обрабатывает спецификацию типа

Когда нет полной уверенности в том, как используемая версия обрабатывает заданную спецификацию вещественного столбца, можно прибегнуть к следующей процедуре. Необходимо создать таблицу, содержащую столбец такого типа, относительно которого существуют определенные сомнения, а после этого воспользоваться оператором DESCRIBE и посмотреть, как СУБД MySQL сообщает о полученном типе. Например, при создании в СУБД MySQL 3.23.0 столбца с типом FLOAT4 вы увидите следующий отчет:

```
mysql> CREATE TABLE t (f FLOAT4);
mysql> DESCRIBE t;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| f     | float(10,2) | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
```

В СУБД MySQL 3.23.6 вместо этого получим такую распечатку:

```
mysql> CREATE TABLE t (f FLOAT4);
mysql> DESCRIBE t;
+-------+-------+------+-----+---------+-------+
```

```
| Field | Type  | Null | Key | Default | Extra |
+-------+-------+------+-----+---------+-------+
| f     | float | YES  |     | NULL    |       |
+-------+-------+------+-----+---------+-------+
```

Отсутствие в последнем случае индикатора (M,D) свидетельствует о том, что значения сохраняются с максимально возможной точностью.

Этот метод действительно помогает определить, как СУБД MySQL обрабатывает любые определения столбцов, но он оказался наиболее показательным при работе с вещественными типами данных.

Выбор цифровых типов столбцов

При выборе цифрового типа, принимая во внимание диапазон возможных значений столбца, обязательно выбирайте наименьший тип. Выбор больших типов приводит к неэффективному увеличению размеров таблиц. В случае целых величин наиболее подходящим типом, например для хранения возраста человека или количества детей, будет тип TINYINT. Столбцы типа MEDIUMINT могут хранить значения порядка миллионов для многих типов данных, но это происходит за счет дополнительного расхода оперативной памяти. Тип BIGINT имеет самый большой диапазон значений, но для его хранения требуется вдвое больше памяти, чем для хранения данных в ближайшем меньшем целом типе INT. Поэтому он должен использоваться только тогда, когда это действительно необходимо. Вещественные данные типа DOUBLE требуют пространства вдвое большего, чем данные типа FLOAT. Поэтому, если не нужны данные исключительно высокой точности или предельно широкого диапазона принимаемых значений, пользуйтесь типом FLOAT. Это поможет вдвое сократить расход памяти.

Очевидно, что любой тип цифровых данных имеет определенный допустимый диапазон значений. При вводе значения, выходящего за пределы, СУБД MySQL усекает значение до ближайшего значения, лежащего внутри диапазона, и возвращает полученный результат. Усечения не происходит при выполнении операции выборки.

Здесь нужно понимать, что усечение значения возникает из-за нарушения диапазона типа столбца, а не из-за нарушения размера отображаемых значений. Так, столбец типа SMALLINT(3) имеет длину отображаемых значений, равную 3, а диапазон значений – от -32768 до 32676. Значение 12345 выходит за пределы длины отображения, но оно попадает в диапазон значений столбца этого типа. Поэтому такое число будет сохранено без усечения. Значение 99999 выходит за пределы допустимого диапазона, поэтому при сохранении оно будет усечено до 32767.

Что касается столбцов с вещественными значениями, то они округляются до десятых, указанных в их спецификации. Присвоение значения 1,23456 столбцу типа FLOAT(8,1) даст значение 1,2. При записи аналогичного значения в столбце типа FLOAT(8,4) получим 1,2346. Это значит, что при объявлении размерности столбцов всегда необходимо принимать во внимание размерность значений, которые будут храниться в них. Поэтому, если необходимо обеспечить точность до тысячных, не следует объявлять только две цифры после запятой.

Тип DECIMAL является вещественным типом, но он отличается от типов FLOAT и DOUBLE тем, что значения в нем сохраняются как строки и имеют фиксированное количество знаков после запятой. И эта разница проявляется в том, что при использовании типа данных DECIMAL возникновение ошибки округления

невозможно. Это делает данный тип исключительно полезным для использования в денежных расчетах. Недостатки применения этого типа заключаются в том, что он не настолько эффективен, как вещественные типы, где значения хранятся в формате, который процессор обрабатывает непосредственно.

Максимальный диапазон значений типа DECIMAL аналогичен значениям типа DOUBLE, но диапазон эффективных значений определяется величинами M и D. Увеличивая значение M при постоянном D, мы увеличиваем диапазон значений. Это хорошо проиллюстрировано в табл. 2.7. Однако, увеличивая значение D при постоянном M, мы уменьшаем диапазон значений (увеличивая при этом точность). Эта ситуация моделируется в табл. 2.8.

Таблица 2.7. Зависимость диапазона значений типа DECIMAL (M, D) от значения атрибута M

Спецификация типа	Диапазон (для версии ниже 3.23)	Диапазон (для версии 3.23 и выше)
DECIMAL(4,1)	-9,9 до 99,9	-999,9 до 9999,9
DECIMAL(5,1)	-99,9 до 999,9	-9999,9 до 99999,9
DECIMAL(6,1)	-999,9 до 9999,9	-99999,9 до 999999,9

Таблица 2.8. Зависимость диапазона значений типа DECIMAL (M, D) от значения атрибута D

Спецификация типа	Диапазон (для версии ниже 3.23)	Диапазон (для версии 3.23 и выше)
DECIMAL(4,0)	-999 до 9999	-9999 до 999999
DECIMAL(4,1)	-9,9 до 99,9	-999,9 до 9999,9
DECIMAL(4,2)	-,99 до 9,99	-99,99 до 999,99

Диапазон типа DECIMAL зависит от используемой версии СУБД MySQL. В версии 3.23 значения типа DECIMAL обрабатываются в соответствии со спецификацией ANSI, в которой констатируется, что столбцы типа DECIMAL(M,D) должны отображать любое значение, содержащее M цифр и D цифр после запятой. Таким образом, тип DECIMAL(4,2) имеет диапазон от -99.99 до 99.99, так как они занимают все отведенные 5 символов. Поскольку требуется место для хранения символов знака и десятичной точки, дополнительно потребуются еще два байта. Таким образом, для хранения значений столбцов, объявленных как DECIMAL(M,D), для СУБД MySQL 3.23 требуется M+2 байтов. Для самого "широкого" значения (-99.99) типа DECIMAL(4,2) потребуется 6 байтов. На положительном конце этого диапазона значений байт для хранения знака не нужен, поэтому его можно использовать для расширения диапазона сверх того, что гарантирует спецификация ANSI. Другими словами, максимальное значение для типа DECIMAL(4,2), которое может храниться в шести байтах, составляет 999,99.

Существуют два специфических условия, позволяющих уменьшить потребление памяти типом DECIMAL до значений, меньших M+2.

- Если значение атрибута D равно 0, значения типа DECIMAL не имеют дробной части и для хранения десятичной точки дополнительный байт не нужен. Это уменьшает расход памяти еще на один байт.
- Если столбец типа DECIMAL имеет атрибут UNSIGNED, отпадает необходимость хранить знак, что также позволяет уменьшить требующуюся память на один байт.

До версии СУБД MySQL 3.23 значения типа DECIMAL представлялись разными способами. Столбец типа DECIMAL(M,D) хранит значения в M байтах, в которые включаются символ знака и десятичная точка (по необходимости). Таким образом, для типа DECIMAL(4,2) диапазон составляет от –0,99 до 9,99, который охватывает все вероятные 4-символьные значения. При D, равном 0, необходимость сохранять десятичную точку отпадает, и байт, который обычно используется для этой цели, можно отвести под сохранение еще одной цифры. При этом преследуется цель расширить диапазон значений столбца на порядок. (Этим и можно объяснить то, что диапазон для версий до 3.23 (см. табл. 2.8) сдвигается с коэффициентом 10 для DECIMAL(4,2) по сравнению с DECIMAL(4,1), но с коэффициентом 100 по сравнению DECIMAL(4,0) и DECIMAL(4,1). Честное слово, вы вряд ли это заметили!)

Атрибуты цифровых типов столбцов

Цифровые типы данных можно объявить с атрибутом ZEROFILL. Он указывает на то, что значения столбца при их отображении дополняются ведущими нулями. Это может пригодиться в том случае, когда надо убедиться в том, что столбцы заполняются правильно. Точнее можно выразиться, что "задано минимальное количество цифр", так как значения, длина которых превышает длину отображения, будут отображаться полностью без усечения. Это можно продемонстрировать с помощью следующих операторов:

```
mysql> DROP TABLE IF EXISTS mytbl;
mysql> CREATE TABLE mytbl(my_zerofill INT(5) ZEROFILL);
mysql> INSERT INTO mytbl VALUES(1), (100), (1000), (1000000);
mysql> SELECT my_zerofill FROM mytbl;
+-------------+
| my_zerofill |
+-------------+
|       00001 |
|       00100 |
|       10000 |
|     1000000 |
+-------------+
```

Обратите внимание на то, что если длина значения превышает длину, объявленную при создании таблицы, отображается все значение.

Атрибут UNSIGNED запрещает присвоение отрицательных значений. Чаще всего он используется с целочисленными типами. Присвоение этого типа столбцу не изменяет диапазон типа данных, он только сдвигает его вверх. Рассмотрим следующую спецификацию:

```
CREATE TABLE mytbl
(
   itiny TINYINT,
   itiny_u TINYINT UNSIGNED
);
```

Здесь itiny и itiny_u являются столбцами типа TINYINT с диапазоном в 256 принимаемых значений, но если область значений столбца itiny лежит между -128 и 127, то область значений столбца itiny_u лежит между 0 и 256.

Атрибут UNSIGNED имеет смысл для столбцов, которые никогда не будут содержать отрицательных значений. В качестве таковых можно назвать количество

населения или показатель посещаемости. В противном случае можно будет воспользоваться только половиной доступного диапазона. Создавая столбец с атрибутом UNSIGNED, пользователь удваивает диапазон значений. Задание атрибута UNSIGNED при использовании столбца для хранения порядковых номеров позволяет удвоить диапазон возможных значений.

Атрибут UNSIGNED можно также задавать для столбцов с вещественными значениями, но при этом эффект будет немного другим. Диапазон значений не сдвигается вверх, верхний предел остается неизменным, а нижний предел становится равным нулю. Необходимо соблюдать осторожность и не использовать атрибут UNSIGNED с такими вещественными типами, как FLOAT или DOUBLE, до версии 4.0.2. В ранних версиях СУБД MySQL не распознает ошибки в объявлении этих типов с таким параметром, но результаты могут быть непредсказуемыми. (Этот запрет не распространяется на тип DECIMAL.)

Другой атрибут, AUTO_INCREMENT, может использоваться только со столбцами целого типа. Этот атрибут указывается при необходимости автоматического создания уникальных, последовательных кодов. Ввод пустого значения в столбец, имеющий атрибут AUTO_INCREMENT, позволяет генерировать следующее по порядку значение. Обычно начальное значение равно 1 с автоматическим приращением 1. Последовательность значений может быть нарушена вследствие удаления строк из таблицы. Но это зависит от типа таблицы, которая задает возможность повторного использования.

Таблица может иметь не больше одного столбца с атрибутом AUTO_INCREMENT. Столбец, имеющий атрибут AUTO_INCREMENT, должен быть объявлен как NOT NULL и проиндексирован как PRIMARY KEY или UNIQUE. Кроме того, так как порядковые номера всегда положительны, их можно одновременно объявлять с атрибутом UNSIGNED. Например, столбец можно объявить с атрибутом AUTO_INCREMENT в таких случаях:

```
CREATE TABLE ai (i INT AUTO_INCREMENT NOT NULL PRIMARY KEY);
CREATE TABLE ai (i INT AUTO_INCREMENT NOT NULL PRIMARY KEY(i));
CREATE TABLE ai (i INT AUTO_INCREMENT NOT NULL UNIQUE(i));
```

Можно объявить явным образом столбец, имеющий атрибут AUTO_INCREMENT, как NOT NULL. Однако начиная с версии 3.23 и выше СУБД MySQL автоматически объявляет как NOT NULL столбцы, имеющие параметр AUTO_INCREMENT. Поведение столбцов AUTO_INCREMENT обсуждается ниже, в разделе "Обработка последовательностей".

Кроме вышеописанных параметров, являющихся специфическими для цифровых столбцов, можно задать и общие параметры — NOT NULL и NULL. По умолчанию цифровые столбцы объявляются с параметром NULL. Кроме того, можно указать значения, принимаемые столбцом по умолчанию. Для этого служит параметр DEFAULT. Без указания значения по умолчанию цифровым данным по умолчанию присваивается значение NULL (для тех столбцов, для которых это допустимо), а остальным — значение 0.

Пример создания таблицы с тремя столбцами типа INT со значениями по умолчанию -1, 1 и NULL:

```
CREATE TABLE t
(
  i1 INT DEFAULT -1,
```

```
  i2 INT DEFAULT 1,
  i3 INT DEFAULT NULL,
);
```

Строковые типы столбцов

В своем арсенале СУБД MySQL имеет несколько типов символьных данных, которые позволяют хранить данные следующего вида:

```
'N. Bertram, et al.'
'Pencils (no. 2 lead)'
'123 Elm St. '
'Monograph Series IX'
```

В действительности строки являются "общим" типом данных в том смысле, что в качестве символьных данных можно хранить любые данные. Это могут быть двоичные данные (изображения или звуки), архивный файл, созданный с помощью архивирующей утилиты `gzip`, и т.д.

Данные по строковым типам, максимальные размеры и требования к памяти приведены в табл. 2.9. Объем памяти, необходимый для хранения данных в столбцах переменной длины, варьируется от строки к строке и зависит от длины значений, реально хранимых в этом столбце. Эта длина обозначается в таблице буквой *L*.

Байты, приплюсованные к *L*, — это байты, необходимые для хранения длины значения. СУБД MySQL позволяет хранить символьные значения переменной длины. Для этого сохраняется как само значение, так и его длина. Эти дополнительные данные можно обрабатывать как беззнаковые целые. Обратите внимание на связь между максимальной длиной типов переменной длины, количеством дополнительных байт, имеющихся для этого типа, и диапазоном беззнакового целого, который использует то же количество байт. Например, значения типа `MEDIUMBLOB` могут иметь длину до $2^{24}-1$ байт и требуют 3 байта для хранения результата. Трехбайтовый целочисленный тип `MEDIUMINT` имеет максимальное значение до $2^{24}-1$. И это не просто совпадение.

Таблица 2.9. Строковые типы столбцов

Тип спецификации	Максимальный размер	Требуемая память
`CHAR(`*M*`)`	*M* байт	*M* байт
`VARCHAR(`*M*`)`	*M* байт	*L*+1 байт
`TINYBLOB, TINYTEXT`	2^8-1 байт	*L*+1 байт
`BLOB, TEXT`	$2^{16}-1$ байт	*L*+2 байт
`MEDIUMBLOB, MEDIUMTEXT`	$2^{24}-1$ байт	*L*+3 байт
`LONGBLOB, LONGTEXT`	$2^{32}-1$ байт	*L*+4 байт
`ENUM('value1', 'value2',...)`	65535 элементов	1 или 2 байт
`SET('value1', 'value2',...)`	64 элемента	1, 2, 3, 4 или 8 байт

Объявление столбцов для типов `ENUM` и `SET` содержит перечень допустимых значений. Попытка сохранения другого значения завершается преобразованием его в пустую строку (`''`). Необходимо учитывать одну особенность строковых типов данных — усечение слишком длинных значений. Однако в СУБД MySQL существуют

самые различные типы строковых данных: от самых коротких до самых длинных. Причем самые вместительные могут хранить до 4 Гбайт информации. Так что с уверенностью можно сказать, что всегда можно подобрать достаточно длинный тип данных, чтобы избежать потери информации[2].

Как будет показано в разделе "Типы ENUM и SET", эти типы хранят свои значения в цифровом виде. Значения других строковых типов хранятся в виде последовательностей байтов и обрабатываются как байты или символы, в зависимости от того, хранит ли этот тип двоичные или недвоичные строки.

- Двоичная строка обрабатывается как простая последовательность байтов без учета кодировки. Столбцы типа BLOB хранят совершенно такие же двоичные значения, как и столбцы типов CHAR и VARCHAR, объявленные с параметром BINARY.
- Недвоичная строка обрабатывается как последовательность символов и интерпретируется в соответствии со свойствами определенной кодировки. Столбцы TEXT хранят недвоичные строки, как и столбцы CHAR и VARCHAR, при объявлении без параметра BINARY. Для однобайтовых кодировок каждый символ занимает один байт. Для многобайтовых кодировок под один символ может потребоваться больше одного байта. В СУБД MySQL версии 4.1 и выше кодировки могут присваиваться отдельным столбцам. До СУБД MySQL 4.1 для интерпретации символьных строк использовалась кодировка, принятая на данном сервере по умолчанию.

Применение кодировки позволяет сравнивать и сортировать недвоичные строки в соответствии с сортировочной последовательностью данной кодировки. Сравните, двоичные строки не имеют кодировки и, следовательно, сортировочной последовательности. Это приводит к определенным различиям при интерпретации двоичных и недвоичных строк.

- Сравнение двоичных строк производится по байтам, на основании числового значения каждого байта. Единственное затруднение здесь заключается в том, что двоичные значения чувствительны к регистру, так как варианты одной буквы в верхнем и нижнем регистрах имеют различные числовые коды.
- Сравнение недвоичных строк производится посимвольно с учетом сортировочной последовательности определенной кодировки. В большинстве кодировок верхний и нижний регистры заданной буквы имеют одно и то же сортировочное значение, что означает, что сравнение недвоичных строк в этом случае не зависит от регистра. Например, в кодировке latin1 сравнение букв "Е" и "Й" дает равенство.

Существует несколько кодировок, в которых верхний и нижний регистры имеют различные сортировочные последовательности для верхнего и нижнего регистров одной буквы и делающих различие между символами с одинарными кавычками и без них: cp1521csas, cp1527t1vcsas, latin1csas, maccecsas и macromancsas. Заметим, что названия этих кодировок завершаются суффиксом

---

[2] Из-за ограничений, накладываемых максимальным размером пакета коммуникационного протокола, эффективный размер столбца имеет определенный предел. Он равен 16 Мбайт в СУБД MySQL до версии 4 и 1 Гбайт – в последующих версиях.

csas, что означает "case sensitive, accent sensitive". Это настолько особенный случай, что всякий раз, когда идет обработка недвоичных строк, которые не чувствительны к регистру, помните, что есть исключение из правил – эти кодировки.

Различие между символами и байтами хорошо заметно при рассмотрении длины строк, содержащих многобайтовые символы. Например, в СУБД MySQL 4.1 и выше для генерирования строки в любой имеющейся кодировке можно использовать функцию CONVERT(). Следующий оператор создает строку @s с использованием кодировки ucs2, в которой для преобразования символов используются два байта:

```
mysql> SET @s = CONVERT('ABC' USING ucs2);
```

Какова же длина строки @s? Она может быть различной. Измерив ее с помощью функции CHAR_LENGTH(), которая учитывает многобайтовую природу символов, вы получите длину в символах. Измерив ее с помощью функции LENGTH(), которая не учитывает многобайтовую природу символов, вы получите длину в байтах:

```
mysql> SELECT CHAR_LENGTH(@s), LENGTH(@s);
+------------------+------------+
| CHAR_LENGTH(@s)  | LENGTH(@s) |
+------------------+------------+
|                3 |          6 |
+------------------+------------+
```

Двоичная строка обрабатывается как простая последовательность отдельных байтов. Следовательно, независимо от измерения ее в символах или байтах, длина всегда будет одинаковой:

```
mysql> SET @s = BINARY CONVERT('ABC' USING ucs2);
mysql> SELECT CHAR_LENGTH(BINARY @s), LENGTH(BINARY @s);
+------------------------+--------------------+
| CHAR_LENGTH(BINARY @s) | LENGTH(BINARY @s)  |
+------------------------+--------------------+
|                      6 |                  6 |
+------------------------+--------------------+
```

Различие между длинами в символах или байтах довольно существенно сказывается при интерпретации значения типов строковых столбцов. Например, объявление столбца как VARCHAR(20) не означает "максимально 20 символов", а означает "столько символов, сколько поместится в 20 байтах". Для однобайтовой кодировки оба определения равнозначны, так как количество символов совпадает с количеством байтов. Но для многобайтовых кодировок количество символов будет заведомо меньше 20.

Типы столбцов CHAR и VARCHAR

Типы CHAR и VARCHAR используются чаще всего. Единственное различие между ними заключается в том, что первый является типом с фиксированной длиной, а второй – с переменной. Все значения типа CHAR(M) занимают по M байт каждое, более короткие значения дополняются пробелами справа. (Однако во время выборки они отсекаются.) Значения типа VARCHAR(M) занимают столько байт, сколько им необходимо, плюс один байт для хранения длины строки. При сохранении значений типа VARCHAR хвостовые пробелы отсекаются. В этом заключается отличие от SQL-формата ANSI для значений типа VARCHAR. (Тип VAR-

CHAR, для которого отсечение хвостовых пробелов не предусмотрено, будет добавлен в будущих версиях СУБД MySQL.)

Столбцы типа CHAR и VARCHAR объявляются со значением M в диапазоне от 1 до 255. Значение M может быть опущено, и в этом случае принимается значение по умолчанию. Начиная с версии 2.23 можно объявлять столбцы типа CHAR(0). Это очень удобно для резервирования места в таблице в случае, если известно, что столбец должен быть, но пока неизвестно, какова должна быть его длина. Позже можно воспользоваться оператором ALTER TABLE для того, чтобы его расширить. Столбец типа CHAR(0) можно использовать для значений, которые могут принимать два значения ON/OFF, если он будет иметь пустое значение. Этими двумя значениями будут значения NULL и пустая строка. Этот тип столбца занимает очень мало места – только 1 бит. Начиная с СУБД MySQL 4.0.2 появилась возможность задавать тип VARCHAR(0), который обрабатывается совсем не так, как CHAR(0).

При выборе между типами столбцов CHAR и VARCHAR необходимо придерживаться двух основополагающих принципов.

- При равных длинах значений тип VARCHAR потребует больше памяти, так как для него требуется дополнительный байт для хранения длины значения. С другой стороны, если значения имеют переменную длину, столбцы типа VARCHAR требуют меньше места. Столбец CHAR(*n*) всегда будет занимать *n* байт, даже если это пустое значение.
- Тип CHAR будет предпочтительней типа VARCHAR в том случае, если длина значений столбца изменяется незначительно и используются таблицы типов MyISAM или ISAM. Причина этого заключается в том, что в таблицах этих типов строки фиксированной длины обрабатываются эффективней строк переменной длины.

За небольшим исключением, в одной таблице нельзя смешивать столбцы типа CHAR и VARCHAR. В зависимости от обстоятельств СУБД MySQL может даже самостоятельно менять типы столбцов. (Это то, что другие базы данных делать не могут.) Ниже перечислены общие принципы работы со столбцами этих типов.

- Таблицы со строками фиксированной длины обрабатываются эффективнее, чем таблицы со строками переменной длины.
- Строки таблицы считаются строками фиксированной длины только в том случае, если все столбцы таблицы имеют фиксированную длину.
- Из-за потери преимущества производительности, проявляющегося при использовании строк фиксированной длины, любой другой столбец фиксированной длины может быть также преобразован в свой эквивалент переменной длины, если это приведет к экономии пространства.

Это значит, что если в таблице имеются столбцы типа VARCHAR, BLOB или TEXT, в ней нельзя использовать столбцы типа CHAR, так как СУБД MySQL просто преобразует их тип в тип VARCHAR. Предположим, что вы создали следующую таблицу:

```
CREATE TABLE my_tbl
(
  c1 CHAR(10),
```

```
   c2 VARCHAR(10)
);
```

Команда DESCRIBE my_tbl даст такой результат:

```
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| c1    | varchar(10) | YES  |     | NULL    |       |
| c2    | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

Обратите внимание на то, что наличие столбца типа VARCHAR вынуждает СУБД MySQL преобразовать тип столбца c1 тоже в тип VARCHAR. При попытке преобразовать тип столбца c1 в тип CHAR с помощью команды ALTER TABLE ничего не получится. Единственным выходом будет преобразование всех столбцов VARCHAR в CHAR одновременно.

```
mysql> ALTER TABLE mytbl MODIFY c1 CHAR(10), MODIFY c2 CHAR(10);
mysql> DESCRIBE mytbl;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| c1    | char(10) | YES  |     | NULL    |       |
| c2    | char(10) | YES  |     | NULL    |       |
+-------+----------+------+-----+---------+-------+
```

Типы BLOB и TEXT тоже являются типами переменной длины, но они не имеют аналогичного типа с фиксированной длиной. Таким образом, типы CHAR, BLOB и TEXT несовместимы и не могут соседствовать в одной таблице. Столбец типа CHAR будет автоматически преобразован в столбец типа VARCHAR.

Исключением, при котором преобразования из типа CHAR в тип VARCHAR не происходит, является случай, когда длина столбца CHAR меньше 4 символов. Например, СУБД MySQL не будет преобразовывать тип столбца CHAR в тип VARCHAR в таблице:

```
CREATE TABLE mytbl
(
   c1 CHAR(2),
   c2 VARCHAR(10))
);
```

Это можно увидеть по результатам работы команды DESCRIBE:

```
mysql> DESCRIBE mytbl;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| c1    | char(2)     | YES  |     | NULL    |       |
| c2    | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

Причина, по которой столбцы короче четырех символов не преобразуются в тип VARCHAR, заключается в том, что в этом случае преобразование даст увеличение необходимой памяти за счет байта, необходимого для хранения длины. В действительности, когда все столбцы короткие, СУБД MySQL автоматически преобразует столбцы типа VARCHAR в столбцы типа CHAR, так как это не приводит

к увеличению потребления памяти. Такое преобразование произойдет с таблицей, созданной следующим оператором CREATE TABLE:

```
CREATE TABLE mytbl
(
  c0 VARCHAR(0),
  c1 VARCHAR(1),
  c2 VARCHAR(2),
  c3 VARCHAR(3)
);
```

Факт изменения типов столбцов можно проверить, используя команду DESCRIBE mytbl.

```
mysql> DESCRIBE mytbl;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| c0    | char(0) | YES  |     | NULL    |       |
| c1    | char(1) | YES  |     | NULL    |       |
| c2    | char(2) | YES  |     | NULL    |       |
| c3    | char(3) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
```

Типы столбцов BLOB и TEXT

Тип BLOB, или двоичный большой объект, — это своеобразный накопитель, который может содержать все, что вы хотите туда поместить. Его можно сделать настолько большим, насколько это возможно. В СУБД MySQL тип BLOB представляет собой семейство типов (TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB), которые различаются только размером памяти, отводимой под них (см. табл. 2.9). Столбцы BLOB полезны при хранении данных, объем которых может сильно возрасти или размер которых сильно варьируется от строки к строке. В качестве примеров таких данных можно привести электронные документы, изображения и звуки, сложные данные и новости. Аналогично, тип TEXT открывает собой целое семейство типов (TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT). Типы BLOB и TEXT во всем аналогичны и отличаются только тем, что при выполнении операций над столбцами типа TEXT учитывается кодировка (для СУБД MySQL 4.1 и выше кодировка может присваиваться непосредственно столбцу TEXT; до версии 4.1 в расчет принималась только стандартная кодировка сервера). Это приводит к таким же результатам, как и в случае с двоичными и недвоичными строками, описанными ранее. Например, при выполнении операций сравнения и сортировки тип BLOB реагирует на регистр данных, а тип TEXT — нет.

Столбцы BLOB и TEXT в зависимости от типа таблицы могут индексироваться.

- Таблицы типа MyISAM и BDB поддерживают индексирование столбцов типов BLOB и TEXT (для СУБД MySQL 3.23.2 и выше). Однако для индекса необходимо задать размер префикса. Это позволяет избежать создания индексных записей больших размеров, что сводит на нет все преимущества, которые можно получить от применения индекса. Исключением является то, что префикс не создается для индексов типа FULLTEXT, так как поиск по индексу такого типа осуществляется на основании всего содержимого индексированных столбцов.

- В таблицах типа ISAM, HEAP и InnoDB индексы по столбцам типа BLOB и TEXT не поддерживаются.

Столбцы типа BLOB и TEXT требуют дополнительного обслуживания по ряду причин.

- Из-за большого разброса размеров столбцов типа BLOB и TEXT таблицы, которые их содержат, подвергаются частым изменениям и обычно сильно фрагментированы. Снижение этого фактора и поддержание хороших показателей производительности может потребовать периодического запуска операции оптимизации с помощью оператора OPTIMIZE TABLE (подробнее об этом рассказывается в главе 4, "Оптимизация запросов").
- При наличии очень больших значений может потребоваться настройка значения параметра max_allowed_packet для увеличения размера передаваемого пакета (подробнее об этом читайте в главе 11, "Общее администрирование MySQL"). Кроме того, может потребоваться увеличение размера пакета и на стороне клиента. (В приложении Д, "Программы MySQL", описано, как это можно осуществить с помощью программы mysql.)

Типы ENUM и SET

Типы ENUM и SET — это особые строковые типы, значения которых выбираются из фиксированного (заданного предварительно) списка допустимых строковых значений. Главная разница между ними заключается в том, что значение типа ENUM должно содержать точно одно значение из указанного множества, столбцы SET могут содержать любой или все элементы множества сразу. Другими словами, тип ENUM содержит взаимоисключающие значения, а тип SET позволяет выбрать несколько значений одновременно.

Тип ENUM используется для перечисления. Столбцу такого типа присваивается одно значение из множества, указанного во время создания таблицы. Перечисление может содержать до 65536 элементов. Перечисления обычно используются для отображения значений, представляющих категории. Например, значения, хранящиеся в столбце, объявленном как ENUM('N', 'Y'), могут принимать только значение 'N' или 'Y'. Тип ENUM можно использовать также для хранения таких значений, как ответы на вопросы из анкеты, размер или цвет товара.

```
employees ENUM('less than 100', '100-500', '501-1500',
               'more than 1500')
color ENUM('red', 'green', 'blue', 'black')
size ENUM('S', 'M', 'L', 'XL', 'XXL')
```

Этим типом можно воспользоваться при обработке выборок из Web-страниц, когда посетитель страницы делает выбор с помощью радиокнопок, расположенных на странице. Так, для выбора типа заказываемой пиццы тип ENUM можно задать следующим образом:

```
crust ENUM('thin', 'regular', 'pan style', 'deep dish')
```

В случае, когда категории перечисления представляют числа, очень важно, чтобы при создании перечисления категории были выбраны верно. Так, при сохранении лабораторных показателей содержания белых телец в крови значения типа ENUM группируются по следующим категориям:

```
wbc ENUM('0 - 100', '101 - 300', '> 300')
```

Когда проверяемый результат попадает в один из перечисловых диапазонов, он запоминается как категория, в которую он попадает. С помощью типа ENUM очень трудно подобрать точное значение для целочисленного счетчика. (Для этого лучше всего использовать отдельный целочисленный столбец.)

Тип SET аналогичен типу ENUM в том смысле, что при объявлении этих типов вы определяете список элементов перечисляемого множества. На этом совпадения заканчиваются. В отличие от типа ENUM, столбец типа SET может принимать любое количество значений из списка. Множество может содержать до 64 элементов. Тип SET можно применить для хранения фиксированного набора не взаимоисключающих значений, как это имеет место для типа ENUM. Например, тип SET можно использовать для хранения вариантов комплектации автомобилей:

```
SET('luggage rack', 'cruise control', 'air conditioning', 'sun roof')
```

Тип SET может содержать варианты, которые действительно заказываются покупателями.

```
SET('cruise control, sun roof')
SET('luggage rack, air conditioning')
SET('luggage rack, cruise control, air conditioning')
SET('air conditioning')
SET('')
```

Пустая строка означает, что покупатель не заказал никаких вариантов. Это тоже допустимое значение.

Значения располагаются в одну строку. Если значение состоит из элементов множества, эти элементы разделяются в строке запятыми. Это значит, что в качестве элемента множества нельзя использовать значение "запятая".

В качестве другого примера использования столбцов типа SET можно привести хранение диагнозов пациентов или результат выборки из Web-страницы. В случае диагноза пациентов элементами множества может быть стандартный перечень вопросов, необходимых для выяснения клинической картины. Пациент ответит на все вопросы или только на часть из них. Для заказа пиццы в режиме "on-line" это может быть выбор требуемых ингредиентов. При этом может быть выбрано сразу несколько ингредиентов.

Правила объявления списков значений для типов SET и ENUM можно определить так.

- Допустимые значения определяются списком.
- Регистр значений задается в объявлении столбца. Поэтому значения в столбце, объявленном как ENUM('N', 'Y'), сохраненные как 'n', 'y', при выборке будут отображаться как 'N', 'Y'. Это не повлияет на результаты при сравнении и сортировке, так как типы SET и ENUM не чувствительны к регистру.
- Порядок значений в объявлении типа ENUM определяет порядок сортировки. Порядок значений в объявлении SET также определяется порядком сортировки. Однако здесь имеет место более сложная зависимость, так как элементы списка могут содержать множественный набор элементов.
- Порядок следования значений в объявлении SET определяет порядок, в котором появляются подстроки при отображении множественных значений.

Так как типы SET и ENUM при объявлении столбцов задаются строками, то они классифицируются как строковые. Но внутренне элементы множеств сохраняются как числа, и их можно обрабатывать как таковые. Это означает, что значения типов SET и ENUM можно обрабатывать как цифровые. Очевидно, что эти типы эффективнее других строковых типов, так как их можно обрабатывать цифровыми, а не строковыми операциями. Это также означает, что типы SET и ENUM можно использовать как в строковом, так и в цифровом контексте.

Элементы типа ENUM нумеруются последовательно, начиная с 1. (0 используется для обозначения ошибочного элемента, который представляется в строковом виде пустой строкой.) Число номеров перечисления определяет объем, который занимает столбец типа ENUM. Один байт может хранить 256 значений, два байта — 65536. (Сравните это с диапазонами значений одно- и двухбайтовых целочисленных типов TINYINT UNSIGNED и SMALLINT UNSIGNED.) Напрашивается вывод, что максимальным количеством элементов перечисления является 65536 (включая ошибочный элемент), а объем требуемой памяти зависит от того, содержит список больше 256 элементов или нет. При этом максимум определяется как 65535 элементов (а не 65536), так как один элемент по умолчанию резервируется для хранения ошибочного элемента. Он нужен для того, чтобы присваивать ошибочное значение всякий раз, когда присваивается недопустимое значение.

Ниже приведен пример, который вы сможете воспроизвести с помощью клиентской программы mysql. Из него видно, что выборку значений типа ENUM можно осуществлять как в числовом, так и строковом виде (он отражает сортировку по цифровому порядку, а также то, что пустому значению номер при сортировке не присваивается).

```
mysql> CREATE TABLE e_table (e ENUM('jane', 'fred', 'will', 'marcia'));
mysql> INSERT INTO e_table;
    -> VALUES('jane'), ('fred'), ('will'), ('marcia'), (''), (NULL);
mysql> SELECT e, e+0, e+1, e+3 FROM e_table;
+--------+------+------+------+
| e      | e+0  | e+1  | e+3  |
+--------+------+------+------+
| jane   |    1 |    2 |    3 |
| fred   |    2 |    3 |    6 |
| will   |    3 |    4 |    9 |
| marcia |    4 |    5 |   12 |
|        |    0 |    1 |    0 |
| NULL   | NULL | NULL | NULL |
+--------+------+------+------+
```

Элементы типа ENUM можно сравнивать по имени или по числовому значению.

```
mysql> SELECT e FROM WHERE e = 'will';
+--------+
| e      |
+--------+
| will   |
+--------+
mysql> SELECT e FROM WHERE e = 3;
+--------+
| e      |
+--------+
| will   |
+--------+
```

Указание пустой строки в качестве элемента перечисления вполне допустимо. Аналогично любому другому элементу, перечисленному в объявлении, ему будет присвоено ненулевое значение. Однако нужно помнить, что использование пустой строки может привести к неопределенности данных, потому что пустая строка может использоваться для элемента с цифровым значением, равным 0. На данном примере видно назначение ошибочного значения 'x'. Это можно увидеть только в цифровом представлении.

```
mysql> CREATE TABLE t (e ENUM('a', '', 'b'));
mysql> INSERT INTO t VALUES('a'), (''), ('b'), ('x');
mysql> SELECT e, e+0 FROM t;
+---+-----+
| e | e+0 |
+---+-----+
| a |   1 |
|   |   2 |
| b |   3 |
|   |   0 |
+---+-----+
```

Цифровое представление столбцов типа SET немного отличается от цифрового представления столбцов типа ENUM. Здесь элементы множества не нумеруются последовательно. Вместо этого каждый элемент соответствует своему биту в значении SET. Первый элемент множества соответствует нулевому биту, второй — первому и т.д. Цифровое значение SET, равное 0, соответствует пустой строке. Элементы множества обрабатываются как биты. Таким образом, в одном байте можно сохранить восемь значений. Отсюда объем требуемой памяти определяется количеством элементов множества до 64 элементов. Значения SET занимают 1, 2, 3, 4 или 8 байт соответственно для множеств от 1 до 8 элементов, от 9 до 16 элементов, от 17 до 24 элементов, от 25 до 32 элементов и от 33 до 64 элементов.

Хранение значений типа SET в виде наборов битов позволяет значению иметь множественный набор элементов. Любая комбинация битов может быть преобразована в значение. Таким образом, значение может принимать любую комбинацию строк.

Вот пример связи между строковыми и цифровыми формами столбцов типа SET. Здесь цифровое значение отображается как в десятичной, так и двоичной форме.

```
mysql> CREATE TABLE s_table (a SET('jane', 'fred', 'will', 'marcia'));
mysql> INSERT INTO s_table
    -> VALUES('jane'), ('fred'), ('will'), ('marcia'), (''), (NULL);
mysql> SELECT s, s+0 BIN(S+0) FROM s_table;
+--------+------+----------+
| s      | s+0  | BIN(s+0) |
+--------+------+----------+
| jane   |    1 | 1        |
| fred   |    2 | 10       |
| will   |    4 | 100      |
| marcia |    8 | 1000     |
|        |    0 | 0        |
| NULL   | NULL | NULL     |
+--------+------+----------+
```

При присвоении столбцу SET значения, содержащего подстроки, не перечисленные как элементы множества, эта строка будет опущена, а столбцу присвоено

значение, состоящее из оставшихся подстрок. Порядок следования подстрок при присвоении значений типа SET не обязательно должен соответствовать порядку следования подстрок в объявлении столбца. Однако потом при выборке значения элементы будут перечислены в том порядке, в котором они заданы при объявлении. Предположим, что столбец SET предназначен для хранения наименований мебельных изделий.

```
SET('table', 'lamp', 'chair')
```

При этом при присвоении столбцу значения `'chair, couch, table'` произойдет следующее. Во-первых, значение `'couch'` будет опущено, так как оно не является элементом множества. Во-вторых, при последующей выборке значений пользователь получит `'chair, table'`. Это происходит из-за того, что СУБД MySQL закрепляет за каждой подстрокой свой бит, а значение `'couch'` не имеет такого соответствия и поэтому будет проигнорировано. При выборке СУБД MySQL, сканируя биты по порядку, конструирует строковое значение из цифрового значения и сортирует строку в том порядке, в котором она была объявлена. Это означает, что даже если элемент упоминается в строке несколько раз, при выборке он будет показан только раз. При выборке значения из столбца типа SET, которому были присвоены значения `'lamp, lamp, lamp'`, будет получено только значение `'lamp'`.

То, что СУБД MySQL пересортирует элементы в значениях SET, означает, что при выборке с помощью строк элементы необходимо указывать в нужном порядке. Значение, добавленное как `'chair, table'`, не будет найдено по выборке `'chair, table'`. Его необходимо искать только как `'table, chair'`.

Сортировка и индексация столбцов типа SET и ENUM производится в соответствии с внутренними (цифровыми) значениями столбца. Этот пример на первый взгляд может показаться ошибочным из-за того, что он не отсортирован в алфавитно-цифровом порядке:

```
mysql> SELECT s FROM e_table ORDER BY e;
+---------+
| e       |
+---------+
| NULL    |
|         |
| jane    |
| fred    |
| will    |
| marcia  |
+---------+
```

Сортировка пустого значения NULL зависит от того, как используемая вами версия СУБД MySQL сортирует пустые значения. (Обратитесь к разделу "Сортировка результатов запроса" в главе 1, "Операторы и функции".)

Для фиксированного набора значений и при необходимости сортировки в определенном порядке можно воспользоваться сортировочным порядком ENUM. При создании таблицы создайте столбец типа ENUM и перечислите значения в том порядке, в котором они должны быть отсортированы. Предположим, что в вашем распоряжении есть таблица, содержащая список членов спортивной организации, например футбольного клуба, и перед вами стоит задача отсортировать персонал в соответст-

вии со служебным положением, занимаемым в клубе. Объявите столбец с типом ENUM и перечислите элементы в порядке, в котором вы хотите их видеть. После сортировки по этому столбцу вы автоматически получите данные в нужном порядке.

В случаях, когда необходимо сортировать значения в привычном лексографическом порядке, столбец можно преобразовать с помощью функции CONCAT() в строку отличного ENUM от типа. Получим сортировку:

```
mysql> SELECT CONCAT(e) AS e_str FROM e_table ORDER BY e_str;
+---------+
| e_str   |
+---------+
| NULL    |
|         |
| fred    |
| jane    |
| marcia  |
| will    |
+---------+
```

Функция CONCAT() не изменяет отображаемые значения, но проявляет побочный эффект в этом запросе преобразования столбца типа ENUM в столбец строкового типа, который нарушает сортировочные свойства типа ENUM.

Атрибуты строковых столбцов

Атрибут BINARY используется только с типами CHAR и VARCHAR. Он позволяет оперировать столбцом как двоичной строкой (т.е. как строкой байтов, а не символьной строкой). Использование этого атрибута делает столбцы чувствительными к регистру.

В СУБД MySQL версии 4.1 и выше для столбцов типа CHAR, VARCHAR и TEXT можно задать атрибут CHARACTER SET *charset*, где *charset* должен быть допустимым именем кодировки. Кодировка в различных столбцах одной таблицы может быть различной. Например, следующая таблица содержит столбцы с кодировками latin1_de (немецкий язык), utf8 (Unicode) и sjis (японский язык):

```
CREATE TABLE mytbl
(
  c1 CHAR(10) CHARACTER SET latin1_de,
  c2 VARCHAR(40) CHARACTER SET utf8,
  t MEDIUMTEXT CHARACTER SET sjis
);
```

В версиях СУБД MySQL, в которых отдельным столбцам может присваиваться отдельная кодировка, оператор DESCRIBE выдаст следующий результат:

```
mysql> DESCRIBE mytbl;
+-------+-----------------------------------+------+-----+---------+---+
| Field | Type                              | Null | Key | Default |
Extra |
+-------+-----------------------------------+------+-----+---------+---+
| c1    | varchar(10) character set latin1_de | YES |     | NULL    |   |
| c2    | varchar(40) character set utf8      | YES |     | NULL    |   |
| t     | mediumtext character set sjis       | YES |     | NULL    |   |
+-------+-----------------------------------+------+-----+---------+---+
```

Двоичные строки не могут иметь кодировки, поэтому атрибут CHARACTER SET не применим к столбцам типов CHAR BINARY, VARCHAR BINARY или к любому из типов BLOB. Атрибут CHARACTER SET не применим также к столбцам типа ENUM или SET, так как они имеют внутреннее цифровое представление.[3]

Начиная с СУБД MySQL 4.1 любой недвоичный символьный столбец имеет определенную кодировку, которая присваивается, даже если в объявлении столбца она не определена явным образом. Кодировка может быть присвоена на уровне столбцов, таблиц, баз данных или сервера. Поэтому при создании символьного столбца СУБД MySQL определяет его кодировку, перебирая правила в следующем порядке.

1. Прежде всего используется кодировка, указанная в объявлении столбца.
2. В противном случае используется кодировка, заданная при объявлении таблицы, отличная от DEFAULT.
3. В противном случае используется кодировка, заданная при создании базы данных, отличная от DEFAULT.
4. В противном случае используется кодировка, принятая по умолчанию для сервера.

Другими словами, СУБД MySQL просматривает снизу вверх все уровни, на которых может быть задана кодировка, до тех пор, пока не будет найдена кодировка, заданная явным образом, которая и будет использоваться для интерпретации столбца. На уровне сервера обязательно имеется кодировка, принятая по умолчанию. Таким образом, процесс поиска обязательно завершится на уровне сервера, даже если ни на одном из низших уровней кодировка явным образом не объявлена.

Предположим, что на сервере установлена кодировка greek, а для активной в данный момент базы данных — DEFAULT. И, наконец, оператор CREATE TABLE не задает никакой кодировки вообще ни на уровне столбцов, ни на уровне таблицы.

```
CREATE TABLE t (c CHAR(10));
```

База данных также не имеет заданной явным образом кодировки, таким образом, СУБД MySQL ищет кодировку до самого серверного уровня, чтобы найти заданную там кодировку (greek), с которой будет работать столбец c. Это можно проверить с помощью команды DESCRIBE:

```
mysql> DESCRIBE t;
+-------+-----------------------------+------+-----+---------+-------+
| Field | Type                        | Null | Key | Default | Extra |
+-------+-----------------------------+------+-----+---------+-------+
| c     | char(10) character set greek | YES  |     | NULL    |       |
+-------+-----------------------------+------+-----+---------+-------+
```

Следующий оператор задает кодировку на табличном уровне; таким образом, в процессе поиска СУБД MySQL останавливает поиск на этом уровне, и для столбца c будет принята кодировка czech:

```
CREATE TABLE t (c CHAR(10)) CHARACTER SET czech;
```

---

[3] В настоящее время строковые значения, относящиеся к значениям ENUM и SET, интерпретируются в соответствии с кодировкой, принятой на сервере по умолчанию. Сейчас ведутся разработки в СУБД MySQL 4.1, чтобы позволить таким столбцам ассоциироваться с указанной кодировкой или объявляться с атрибутом BINARY — новая возможность, которая уже может быть реализована в новой версии СУБД MySQL в то время, когда эта книга дойдет до своего читателя.

Это снова можно проверить с помощью команды DESCRIBE:

```
mysql> DESCRIBE t;
+-------+-------------------------------+------+-----+---------+-------+
| Field | Type                          | Null | Key | Default | Extra |
+-------+-------------------------------+------+-----+---------+-------+
| c     | char(10) character set czech | YES  |     | NULL    |       |
+-------+-------------------------------+------+-----+---------+-------+
```

Принципы присвоения кодировки детально будут объяснены позже, в разделе "Поддержка кодировок".

Столбец строкового типа может быть указан с общим атрибутом NULL или NOT NULL. Значение по умолчанию — NULL. Однако объявление столбца как NOT NULL ни в коем случае не может предотвратить ввода пустой строки. Здесь нельзя путать отсутствующее значение с пустой строкой. Очевидно, что нельзя предотвратить ввод пустой строки, объявив ее с параметром NOT NULL. Предотвратить ввод пустых строк можно только на прикладном уровне.

Для всех типов столбцов, кроме BLOB и TEXT, можно задать атрибут DEFAULT. Таким образом, объявляется значение по умолчанию. Если значение по умолчанию не определено, то столбцам присваиваются следующие значения. Пустое значение — для столбцов, объявленных с атрибутом NULL, пустая строка — для столбцов, которые объявлены с атрибутом NOT NULL. Исключение составляют столбцы, объявленные с типом ENUM, где в качестве значения по умолчанию назначается первый элемент перечисления. (Значением по умолчанию типа SET, в случае когда столбец не может содержать пустые значения (NULL), будет пустое множество, которое можно рассматривать в качестве эквивалента пустой строки.)

Календарные типы столбцов

СУБД MySQL имеет несколько типов столбцов, предназначенных для хранения календарных данных: DATE, DATETIME, TIME, TIMESTAMP и YEAR (табл. 2.10). Требования к памяти представлены в табл. 2.11.

Таблица 2.10. Календарные типы столбцов

Спецификация типа	Диапазон
DATA	от '1000-01-01' до '9999-12-31н
TIME	от '-838:59:59' до '838:59:59'
DATATIME	от '1000-01-01 00:00:00' до '9999-12-31 00:00:00'
TIMESTAMP[(M)]	от 19700101000000 до неопределенной даты в 2037 году
YEAR[(M)]	от 1901 до 2155 для YEAR(4) и от 1970 до 2069 для YEAR(2)

Все типы имеют так называемое нулевое значение, которое записывается тогда, когда вводится недопустимое значение. Нулевые значения календарных типов приведены в табл. 2.12. Это значение также является значением по умолчанию для столбцов, объявленных как NOT NULL.

Таблица 2.11. Объем памяти, требующийся для столбцов календарного типа

Спецификация типа	Объем памяти
DATA	3 байта (4 байта для версий до 3.22)
TIME	3 байта
DATATIME	8 байт
TIMESTAMP	4 байта
YEAR	1 байт

Таблица 2.12. Нулевые значения календарных типов данных

Спецификация типа	Нулевое значение
DATA	'0000-00-00'
TIME	'00:00:00'
DATATIME	'0000-00-00 00:00:00'
TIMESTAMP	00000000000000
YEAR	0000

В соответствии со спецификацией ANSI SQL и ISO 8601, даты представляются в формате "год-месяц-день". Например, дата 3 декабря 2004 года представляется в формате '2004-12-03'. СУБД MySQL делает некоторые "послабления" при вводе дат, позволяя вводить данные с незначительными отклонениями от стандарта. Так, она автоматически преобразует представление года двумя цифрами в представление года четырьмя цифрами; допускается также ввод чисел и месяцев меньше 10 без ведущего нуля, однако сначала нужно указать год. Более привычные форматы, такие как '12/3/99' или '3/12/99', могут интерпретироваться неверно. Правила интерпретации дат в СУБД MySQL обсуждаются подробнее ниже, в разделе "Работа со столбцами календарного типа".

Временные значения возвращаются в местном времени, установленном на сервере СУБД MySQL. Адаптация клиента к местному времени не производится.

Типы DATE, TIME и DATETIME

Столбцы типа DATE, TIME, DATETIME используются для хранения дат и времени, а также для дат и времени одновременно. Их форматы — 'CCYY-MM-DD', 'hh:mm:ss' и 'CCYY-MM-DD hh:mm:ss' соответственно, где CC, YY, MM, DD, hh, mm и ss соответствуют веку, годам, месяцам, дням, часам, минутам и секундам. В столбцах типа DATETIME требуется указать как часть с датой, так и часть, в которой указывается время. При присвоении столбцу типа DATETIME значения из столбца DATE временной части будет добавлено значение '00:00:00'. (И наоборот, при присвоении столбцу DATE значения, заданного типом DATETIME, СУБД MySQL отбрасывает часть, задающую время.)

СУБД MySQL обрабатывает временные данные типа DATETIME, TIME по-разному. В столбцах типа DATETIME СУБД MySQL хранит информацию о времени в формате времени суток, в столбцах типа TIME — как прошедшее время (вот почему диапазон значений для этого типа так велик и допускаются отрицательные значения).

При вводе значений типа TIME вы должны быть уверены в том, что используете только "короткое" значение (неполное). Например, можно проверить, что при

вводе значений '30' и '12:30' в столбец типа TIME одно значение будет интерпретировано справа налево, в другое — слева направо. В результате будут сохранены значения '00:00:30' и '12:30:00' (т.е. для хранения времени "12 часов 30 минут" его необходимо вводить как "12:30:00").

Тип TIMESTAMP

Столбцы типа TIMESTAMP предназначены для хранения в формате CCYYMMDDhhmmss даты начиная с 19700101000000 и заканчивая неопределенной датой в 2037 году. Этот диапазон привязан к таймеру ОС UNIX, в котором отсчет начинается с первого дня 1970 года, который является "нулевой датой", или "началом эпохи" ОС UNIX. Начало 1970 года опеределяет начало диапазона значений, принимаемых данными типа TIMESTAMP. Верхнее значение диапазона соответствует четырехбайтовому предельному значению, которое может содержать значения, отражающие какую-то неопределенную дату в 2037 году[4].

Столбец типа TIMESTAMP (временная отметка) имеет такое название потому, что он записывается в момент создания или модификации записи.

- При вводе пустого значения в столбец TIMESTAMP он автоматически получит значение текущей даты и времени.
- Это же происходит при создании или модификации строки, но без присвоения какого-либо явного значения данному столбцу. Однако подобным образом будут обрабатываться данные только первого столбца данного типа.
- Значение любого столбца типа TIMESTAMP можно модифицировать, установив текущую временную отметку и присвоив ему пустое значение; с другой стороны, установку временных отметок можно отменить, задав в качестве значения столбца явным образом дату и время.

При объявлении столбца TIMESTAMP можно определить максимальную длину выводимого значения *M*. Допустимые форматы отображения представлены в табл. 2.13. Если *M* в объявлении отсутствует, равно 0 или больше 14, столбцу будет присвоено значение TIMESTAMP(14). Все нечетные значения *M*, лежащие в диапазоне от 1 до 13, будут приведены к ближайшему большему четному числу.

Таблица 2.13. Форматы отображения типа TIMESTAMP

Спецификация типа	Формат отображения
TIMESTAMP(14)	*CCYYMMDDhhmmss*
TIMESTAMP(12)	*YYMMDDhhmmss*
TIMESTAMP(10)	*YYMMDDhhmm*
TIMESTAMP(8)	*CCYYMMDD*
TIMESTAMP(6)	*YYMMDD*
TIMESTAMP(4)	*YYMM*
TIMESTAMP(2)	*YY*

[4] Верхнее значение диапазона типа данных TIMESTAMP возрастает по мере роста верхнего предела диапазона временных значений в ОС UNIX. Все это сделано на уровне системных библиотек, поэтому СУБД MySQL сразу же воспользуется нововведениями.

Формат отображения столбцов типа TIMESTAMP не имеет ничего общего с объемом памяти, который требуется для запоминания, и внутренним представлением значений. Значения этого типа данных хранятся в 4 байтах и используются в вычислениях с полной точностью 14 знаков независимо от ширины отображения. Для того чтобы это продемонстрировать, создадим таблицу, затем добавим несколько строк и выберем их.

```
mysql> CREATE TABLE mytbl (ts TIMESTAMP(8), i INT);
mysql> INSERT INTO mytbl VALUES(20020801120000,3);
mysql> INSERT INTO mytbl VALUES(20020801120000,2);
mysql> INSERT INTO mytbl VALUES(20020801120000,1);
mysql> INSERT INTO mytbl VALUES(20020801120000,0);
mysql> SELECT * FROM mytbl ORDER BY ts, i;
+----------+-----+
| ts       | i   |
+----------+-----+
| 20020801 |   3 |
| 20020801 |   2 |
| 20020801 |   1 |
| 20020801 |   0 |
+----------+-----+
```

По полученному результату видно, что строки отсортированы в неверном порядке — значения первого столбца одинаковы, и может показаться, что строки отсортированы в соответствии с порядком следования столбцов второго столбца. Этот абсолютно неверный результат получен благодаря тому, что сортировка значений типа TIMESTAMP в СУБД MySQL делается по полному значению, состоящему из 14 знаков.

СУБД MySQL не имеет типа столбца, который может принимать значение текущей даты и текущего времени в момент создания записи. Это можно сделать двумя способами.

- В столбце типа TIMESTAMP. При создании записи для инициализации текущей даты и времени присвойте столбцу пустое значение.

  ```
  INSERT INTO tbl_name (ts_col,...) VALUES (NULL,...);
  ```

  Если впоследствии потребуется модифицировать запись, присвойте явным образом столбцу значение, которое он уже имеет. Это подавит механизм автоматического присвоения текущей даты и времени и предотвратит автоматическую модификацию столбца.

  ```
  UPDATE tbl_name SET ts_col = ts_col WHERE...;
  ```

- В столбце типа DATETIME. При создании записи для инициализации текущей даты и времени присвойте столбцу значение NOW().

  ```
  INSERT INTO tbl_name (dt_col,...) VALUES (NOW(),...);
  ```

  Если в дальнейшем возникнет необходимость модифицировать запись, не трогайте эти столбцы.

  ```
  UPDATE tbl_name SET /* здесь можно задавать все, кроме столбца dt_col
  / WHERE...;
  ```

По мере необходимости можно использовать столбцы TIMESTAMP как для хранения даты создания, так и для хранения даты последней модификации. Для этого необходимо создать два столбца: один типа TIMESTAMP для хранения времени соз-

дания записи, а другой — аналогичного типа для хранения времени модификации записи. При этом столбец, хранящий дату модификации записи, должен быть первым в таблице, а столбец, хранящий дату создания записи, — вторым. Первый столбец будет принимать значение текущей даты всякий раз, когда запись создается или изменяется, а второму будет присваиваться значение NOW() в момент создания записи. Таким образом, можно ввести данные по добавлению записей в таблицы, которые не будут изменяться при внесении изменений в таблицу.

Тип YEAR

Тип YEAR является экономичным способом хранения данных по году. Объявление столбца типа YEAR предусматривает включение спецификации ширины отображения M, которая может равняться 4 или 2. Если M в объявлении типа YEAR отсутствует, принимается значение по умолчанию, 4. Диапазон типа YEAR(4) начинается 1901 годом и заканчивается 2155 годом. Диапазон типа YEAR(2) начинается 1970 годом и заканчивается 2069 годом, при этом отображаются только две последние цифры. Этот тип удобен, когда требуется хранить только ту часть даты, которая обозначает год. Таким образом можно хранить год рождения, год избрания в президенты и т.д. Это значит, что всякий раз, когда не требуется полная дата, тип YEAR является более эффективным, чем другие типы данных.

Тип TINYINT требует столько же памяти, сколько и YEAR (один байт), но имеет другой диапазон принимаемых значений. Для того чтобы получить аналогичный диапазон значений, необходимо прибегнуть к помощи типа SMALLINT, который занимает вдвое больше места. Таким образом, тип YEAR эффективнее использует память. В числе других преимуществ типа YEAR перед целочисленными типами можно назвать и то, что СУБД MySQL всегда преобразует двухцифровое значение в четырехцифровое. Так, годы 97 и 14 будут преобразованы в 1997 и 2014. Здесь уместно напомнить, что добавление значения 00 приведет к записи значения 0000, а не 2000. А строковое значение '00' будет корректно преобразовано в 2000.

Атрибуты столбцов календарного типа

Эти типы данных не имеют каких-либо специальных атрибутов. Среди общих атрибутов, применяющихся с этими типами данных, можно назвать такие, как NULL и NOT NULL. Атрибут NULL присваивается по умолчанию. Кроме того, для указания значения по умолчанию можно прибегнуть к помощи атрибута DEFAULT. Если это значение не указано, значение по умолчанию принимается автоматически. Так, столбцы, которые могут принимать пустое значение, по умолчанию принимают значение NULL, в противном случае — 0.

Следует заметить, что так как значения по умолчанию должны быть константами, для задания значения "текущей даты и времени" нельзя использовать значение функции NOW() в качестве значения по умолчанию для столбца типа DATETIME. Для того чтобы получить этот результат при создании столбца такого типа, необходимо явным образом указать значение NOW() или воспользоваться столбцом типа TIMESTAMP (в предположении, что специальные свойства типа TIMESTAMP подходят для ваших целей).

Работа со столбцами календарного типа

СУБД MySQL интерпретирует дату и время в различных форматах. Допустимые форматы для дат и для времени приведены в табл. 2.14.

Таблица 2.14. Форматы ввода данных календарного типа

Тип	Допустимый формат
`DATETIME, TIMESTAMP`	*'CCYY-MM-DD hh:mm:ss'*
	'YY-MM-DD hh:mm:ss'
	'CCYYMMDDhhmmss'
	'YYMMDDhhmm'
	CCYYMMDDhhmmss
	YYMMDDhhmm
`DATE`	*'CCYY-MM-DD'*
	'YY-MM-DD'
	'CCYYMMDD'
	'YYMMDD'
	CCYYMMDD
	YYMMDD
`TIME`	*'hh:mm:ss'*
	'hhmmss'
	hhmmss
`YEAR`	*'CCYY'*
	'YY'
	CCYY
	YY

Форматы, не имеющие части, описывающей столетие (CC), интерпретируются в соответствии с правилом, детально описанным в разделе "Интерпретация неоднозначных значений года". Для строковых типов, которые имеют символы-разделители, в качестве таковых совсем не обязательно использовать знаки '-' для дат или ':' для времени. В качестве разделителя может использоваться любой знак пунктуации. Интерпретация данных зависит от их контекста, а не от разделителя. Например, несмотря на то, что часы, минуты и секунды обычно разделяются символом ':', СУБД MySQL не станет интерпретировать значение, содержащее ':', как время в контексте, когда ожидается дата. Кроме того, в строковых форматах, которые содержат разделители, совсем не обязательно указывать две цифры для описания месяца, дня, часа, минуты или секунды, значение которых меньше десяти. Обратите внимание, что все эти строки идентичны:

```
'2012-02-03 05:04:09'
'2012-2-03 05:04:09'
'2012-2-3 05:04:09'
'2012-2-3 5:04:09'
'2012-2-3 5:4:09'
'2012-2-3 5:4:9'
```

Совет

В зависимости от типа (строковые данные или цифровые) значения с ведущими нулями могут интерпретироваться по-разному. С одной стороны, строка "001231" отображается как строка из шести цифр, "2000-12-31" — как дата типа DATE, "2000-12-31 00:00:00" — как дата типа DATETIME. С другой стороны, при интерпретации числового значения "001231" получим 1231, что сделает дальнейшую интерпретацию этого числа довольно проблематичной. Поэтому лучше пользоваться строковыми данными или полностью квалифицировать значения, если используются цифровые типы данных (т.е. 20001231 для DATE или 200012310000 для DATETIME).

В общем случае можно достаточно свободно обмениваться данными между типами данных DATE, DATETIME и TIMESTAMP, но при этом следует помнить следующее.

- При присвоении значения DATETIME, TIMESTAMP столбцу типа DATE временные показатели будут утеряны.
- При присвоении значения DATE столбцам типа DATETIME, TIMESTAMP временной части будет присвоено нулевое значение ('00:00:00').
- Типы имеют различные диапазоны значений. В частности, тип TIMESTAMP имеет более ограниченный диапазон (от 1970 до 2037). Так, присвоение даты раньше 1970 года из столбца типа DATETIME столбцу типа TIMESTAMP приведет к ошибочным результатам. Аналогичная ситуация возможна с датами в будущем.

СУБД MySQL предлагает много функций, работающих с календарными данными. Более подробную информацию о календарных функциях можно найти в приложении В, "Операторы и функции".

Интерпретация неоднозначных значений года

Все типы данных, содержащие информацию о годе (DATE, DATETIME, TIMESTAMP, YEAR), могут преобразовывать значения, содержащие две цифры, в значения, состоящие из четырех цифр. Это преобразование подчиняется следующим правилам.

- Значения 00–69 преобразуются в 2000–2069.
- Значения 70–99 преобразуются в 1970–1999.

Проще всего ознакомиться с этими правилами на практике, задав столбцу типа YEAR несколько различных значений, содержащих две цифры, а затем сделать выборку. В итоге можно получить довольно интересные результаты:

```
mysql> CREATE TABLE y_table (y YEAR)
mysql> INSERT INTO y_table VALUES(68), (69), (99), (00)
mysql> SELECT * FROM y_table
+------+
| y    |
+------+
| 2068 |
| 2069 |
| 1999 |
| 0000 |
+------+
```

Совет

Обратите внимание на то, что значение 00 было преобразовано в 0000, а не в 2000. Это произошло потому, что значение 0 является допустимым значением для данных типа YEAR. Чтобы получить значение 2000, добавьте строку "0" или "00". Для того чтобы быть уверенным, что это строковое, а не числовое значение, добавьте его с помощью функции CONCAT(), которая возвращает строковые данные независимо от типа данных параметра.

В любом случае следует помнить, что преобразование двухцифрового представления года в четырехцифровое может стать источником ошибки. Никогда нельзя быть уверенным в дате, когда столетие не определено. Когда нет полной уверенности в том, что преобразование дат СУБД MySQL дает нужное значение, решение очевидно: пользуйтесь однозначными данными (представление года четырьмя цифрами).

Решена ли проблема 2000 года для СУБД MySQL

Сама по себе СУБД MySQL защищена от проблемы 2000 года, так как здесь обеспечено внутреннее хранение дат в представлении года четырьмя цифрами, однако ответственность за правильное хранение значений прежде всего лежит на пользователе СУБД MySQL. Сама проблема интерпретации двухцифрового года коренится не в свойствах и характеристиках СУБД MySQL, а в естественном человеческом желании не тратить усилия и ввести неоднозначные данные. Если у вас появилось желание рискнуть, можете сделать это. Рискуете вы сами, а правила интерпретации данных адекватны во многих ситуациях. Но помните, что есть ситуации, когда действительно необходимо вводить четыре цифры. Например, при вводе дат рождения и смерти президентов США в таблицу president, в которой хранится список президентов США до 1700-х годов включительно, уместны только четырехцифровые значения. Значения, представленные в этой таблице, охватывают несколько столетий, поэтому предоставление СУБД MySQL возможности угадывать век по двухцифровому представлению года будет ошибочным решением.

Обработка последовательностей

Во многих случаях для идентификации данных требуется уникальное число. Необходимость использования уникальных значений возникает во многих случаях: это могут быть членские номера, номера лотов, идентификаторы потребителей, отчеты о возникающих ошибках и т.д.

В СУБД MySQL механизм такой нумерации осуществляется с помощью атрибута AUTO_INCREMENT. Он позволяет генерировать последовательные номера автоматически. Однако столбцы AUTO_INCREMENT обрабатываются по-разному в зависимости от применяемого типа таблицы, поддерживаемого СУБД MySQL. В этом разделе описывается поведение атрибута AUTO_INCREMENT. Надеюсь, это позволит избежать неожиданных ошибок. Кроме того, будет показано, как функциональность атрибута AUTO_INCREMENT можно эмулировать без атрибута AUTO_INCREMENT.

Для различных версий СУБД MySQL (вплоть до 3.23) единственным доступным механизмом был ISAM. После этого появились дополнительные типы: сначала MyISAM и HEAP, затем BDB и InnoDB. Ниже мы рассмотрим, как каждый тип ведет себя относительно столбцов, объявленных с атрибутом AUTO_INCREMENT. (Для получения более полной информации о характеристиках заголовков таблиц СУБД MySQL обратитесь к главе 3, "Синтаксис и использование языка SQL".)

Атрибут AUTO_INCREMENT для таблиц ISAM

Столбцы с атрибутом AUTO_INCREMENT в таблицах ISAM ведут себя следующим образом.

- Добавление пустого значения в столбец AUTO_INCREMENT приводит к добавлению в данном столбце следующего по счету номера в новой строке. Эта последовательность чисел начинается с 1, где первая запись имеет значение, равное 1, следующая — 2, затем — 3 и т.д. В общем, каждое новое значение будет на единицу больше, чем хранящееся в этом столбце максимальное значение.
- Добавление значения 0 в столбец AUTO_INCREMENT аналогично добавлению пустого значения. Но поскольку это не является гарантированным результатом, лучше все-таки вводить значение NULL.
- Добавление строки без определенного значения в столбце AUTO_INCREMENT также аналогично добавлению пустого значения.
- При добавлении новой записи с указанным значением в столбце с атрибутом AUTO_INCREMENT явным образом произойдет одно из двух. Если запись с таким значением уже существует в таблице, возникает ошибочная ситуация, так как значения в столбце AUTO_INCREMENT должны быть уникальны. Если строка с таким значением отсутствует, новое значение станет новым максимумом последовательности. Таким образом, дается скачкообразное приращение счетчика уникальных записей.

  Такой "скачок" счетчика приведет к возникновению пропусков в непрерывной последовательности чисел. Из этой ситуации также можно извлечь преимущество. Предположим, что была создана таблица типа ISAM со столбцом AUTO_INCREMENT, но нумерация должна начинаться с 1000. Для этого создадим первую запись со значением 1000, заданным явно, или первую пустую запись со значением в этом столбце 999. Во втором случае первая реальная запись получит значение 1000. После этого пустую запись будет можно удалить.

  (Зачем это может понадобиться? Одной из веских причин может быть необходимость создания последовательности чисел, все числа которой имеют одинаковое количество цифр. Допустим, что количество заказчиков никогда не будет больше одного миллиона, в этом случае при создании идентификационных номеров покупателей последовательность чисел может начинаться с 1000000. Это позволит добавить более миллиона записей, без изменения количества цифр в идентификационном номере заказчика. Другая причина, из-за которой не следует начинать список с 1, может лежать совершенно не в технической плоскости. Например, при присвоении членских номеров может оказаться уместным начинать последовательность с чисел больше 1, чтобы избежать политических трений по поводу присвоения номера 1. Да, и такое случается!)
- После удаления записи, содержащей максимальное значение в столбце AUTO_INCREMENT, это значение будет использовано повторно для генерирования нового значения. Это последствие применения принципа, констатирующего, что для таблиц типа ISAM каждое новое автоматически созданное значение должно быть на единицу больше максимального значения, сохраненного в столбце. При удалении всех записей из таблицы нумерация опять будет начата с 1.

- При попытке установки в столбце AUTO_INCREMENT с помощью оператора UPDATE значения, которое уже существует в другой записи, возникнет ошибка "дублирование ключа" ("duplicate key"). Изменение значения столбца до значения, превышающего существующее значение столбца, приведет к тому, что последовательность будет продолжена значением, следующим после нового значения.
- При попытке установки значения в столбце AUTO_INCREMENT с помощью оператора REPLACE значение не будет изменено. При использовании оператора REPLACE со значением NULL для модификации записи на основании какого-то другого индекса PRIMARY KEY или UNIQUE столбец, объявленный с атрибутом AUTO_INCREMENT, будет обновлен с новым порядковым номером.
- Наибольшее максимальное автоматически сгенерированное значение можно просмотреть с помощью функции LAST_INSERT_ID(). Она позволяет обратиться к текущему значению AUTO_INCREMENT, не зная, каково оно в реальности. Значение, возвращаемое функцией LAST_INSERT_ID(), привязано к значению AUTO_INCREMENT, полученному во время последнего сеанса, и не связано с другими сеансами, работавшими со столбцами AUTO_INCREMENT. Функция LAST_INSERT_ID() возвращает 0, если во время текущей сессии не было сгенерировано ни одного значения AUTO_INCREMENT.

Механизм поведения атрибута AUTO_INCREMENT для таблиц ISAM формирует базис для понимания поведения последовательностей для таблиц других типов. Поведение таблиц этих типов во многом аналогично вышеописанному, и это необходимо иметь в виду по мере дальнейшего ознакомления с материалом настоящей книги.

Атрибут AUTO_INCREMENT для таблиц MyISAM

Таблицы MyISAM обладают наибольшей гибкостью при обработке последовательностей. Формат хранения таблиц MyISAM имеет усовершенствование, которое направлено на устранение некоторых узких мест формата ISAM.

- В таблицах типа ISAM удаляемые значения используются повторно. При удалении записи с максимальным порядковым номером новая запись получит то же последовательное значение, что и удаленная запись. Это приводит к созданию последовательностей, не обеспечивающих строгую монотонность, так необходимую для обеспечения гарантии уникальности номеров. В таблицах формата MyISAM величины в автоматически генерируемых последовательностях строго возрастающие и повторно не используются. Если максимальное значение равно 143 и эта запись будет удалена, СУБД MySQL создаст следующую запись со значением 144.
- Последовательности формата ISAM всегда начинаются с 1, если специально для запуска не начать последовательность с большего значения, воспользовавшись методом создания ложных записей, который упоминался раньше. Формат MyISAM в операторе CREATE TABLE позволяет задавать начальное значение явным образом атрибутом AUTO_INCREMENT = *n*. Следующий пример демонстрирует создание таблицы в формате MyISAM со столбцом

seq, снабженным атрибутом AUTO_INCREMENT, имеющим начальное значение 1000000:

```
CREATE TABLE mytbl
(
    seq INT UNSIGNED AUTO_INCREMENT NOT NULL,
    PRIMARY KEY (seq)
) TYPE = MYISAM AUTO_INCREMENT = 1000000;
```

В таблице может объявляться только один столбец, снабженный атрибутом AUTO_INCREMENT. Поэтому никогда не возникнет неоднозначности относительно того, к какому столбцу имеет отношение завершающий атрибут AUTO_INCREMENT = *n*.

- Текущий счетчик последовательности можно изменить с помощью оператора ALTER TABLE. Если текущее значение счетчика равно 1000, следующий оператор сделает так, что текущее значение счетчика станет равным 2000.

```
ALTER TABLE mytbl AUTO_INCREMENT = 2000;
```

При желании значения, удаленные из последовательности, можно использовать повторно. Этот оператор установит значение счетчика в минимальное значение:

```
ALTER TABLE mytbl AUTO_INCREMENT = 1;
```

В дополнение к преодолению узких мест в обработке последовательностей, присущих формату ISAM, обработчик таблиц СУБД MySQL 3.23.5 поддерживает использование составных (многостолбцовых) индексов для создания нескольких независимых последовательностей в пределах одной таблицы. Для того чтобы воспользоваться этой возможностью, создайте с помощью атрибута PRIMARY KEY или UNIQUE многостолбцовый индекс, в котором в качестве последнего столбца задан символ с атрибутом AUTO_INCREMENT. Для каждого ключа в самом левом столбце или столбцах индекса столбец AUTO_INCREMENT будет создавать отдельную последовательность значений.

Например, таблица bugs предназначена для ведения отчетов об ошибках нескольких проектов разработки программных продуктов:

```
CREATE TABLE bugs
(
  proj_name VARCHAR(20) NOT NULL,
  bug_id INT UNSIGNED AUTO_INCREMENT NOT NULL,
  description VARCHAR(100),
  PRIMARY KEY (proj_name, bug_id)
) TYPE = MYISAM;
```

Здесь столбец proj_name содержит наименование проекта, а столбец description — описание ошибки. Столбец bug_id объявлен с параметром AUTO_INCREMENT. Создавая индекс, который привязывает его к столбцу proj_name, вы можете создать независимую последовательность чисел для каждого проекта. Предположим, что следующие добавляемые в таблицу записи регистрируют три ошибки для проекта SuperBrowser и две для SpamSquisher:

```
mysql> INSERT INTO bugs (proj_name,description)
    -> VALUES('SuperBrowser', 'crashes when displaying complex tables');
mysql> INSERT INTO bugs (proj_name,description)
    -> VALUES('SuperBrowser', 'image scaling does not work');
mysql> INSERT INTO bugs (proj_name,description)
```

```
    -> VALUES('SpamSquisher', 'fails to block known blacklisted domains');
mysql> INSERT INTO bugs (proj_name,description)
    -> VALUES('SpamSquisher', 'fails to respect whitelist addresses');
mysql> INSERT INTO bugs (proj_name,description)
    -> VALUES('SuperBrowser', 'background patterns not displayed');
```

В результате получим следующее содержимое таблицы:

```
mysql> SELECT * FROM bugs ORDER BY proj_name, bug_id;
+-------------+--------+-----------------------------------------+
| proj_name   | bug_id | description                             |
+-------------+--------+-----------------------------------------+
| SpamSquisher |      1 | fails to block known blacklisted domains |
| SpamSquisher |      2 | fails to respect whitelist addresses    |
| SuperBrowser |      1 | crashes when displaying complex ables   |
| SuperBrowser |      2 | image scaling does not work             |
| SuperBrowser |      3 | background patterns not displayed       |
+-------------+--------+-----------------------------------------+
```

Обратите внимание на то, что порядок следования записей между проектами не имеет никакого значения. Значения порядковых номеров в столбце bug_id присваиваются отдельно для каждого проекта. При использовании составного индекса для создания нескольких последовательностей максимальные удаленные значения *будут* использованы повторно. Это контрастирует с обычным поведением формата MyISAM, в котором значения не используются повторно.

Атрибут AUTO_INCREMENT для таблиц HEAP

Таблицы типа HEAP не поддерживают механизм AUTO_INCREMENT до СУБД MySQL версии 4.1. Начиная с версии 4.1 объявление столбцов с этим атрибутом уже разрешено, и они ведут себя следующим образом.

- В операторе CREATE TABLE можно явным образом задать начальное значение параметром AUTO_INCREMENT = *n*. Потом его можно изменить, используя этот параметр с оператором ALTER TABLE.
- Удаленные значения повторно не используются.
- Для создания множественных независимых последовательностей составные индексы не используются.

Атрибут AUTO_INCREMENT для таблиц BDB

Механизм BDB обрабатывает столбцы AUTO_INCREMENT следующим образом.

- В операторе CREATE TABLE можно явным образом задать начальное значение атрибутом AUTO_INCREMENT = *n*. Потом его можно изменить, используя этот параметр с оператором ALTER TABLE.
- Удаленные значения повторно не используются.
- Для создания множественных независимых последовательностей могут использоваться составные индексы.

Атрибут AUTO_INCREMENT для таблиц InnoDB

Механизм InnoDB обрабатывает столбцы AUTO_INCREMENT следующим образом.

- В операторе CREATE TABLE можно явным образом задать начальное значение атрибутом AUTO_INCREMENT = *n*. Потом его можно изменить, используя этот параметр с оператором ALTER TABLE.
- Удаленные значения повторно не используются.
- Для создания множественных независимых последовательностей составные индексы не используются.

Вопросы, которые необходимо учитывать при использовании атрибутов AUTO_INCREMENT

Для того чтобы избежать неожиданностей при использовании столбцов AUTO_INCREMENT, необходимо учитывать следующее.

- AUTO_INCREMENT — это не тип столбца, а атрибут типа столбца. Более того, AUTO_INCREMENT — это атрибут, предназначенный для использования только с целочисленными типами. В версиях СУБД MySQL до 3.23 это ограничение не соблюдается, и они могут объявить такой тип как CHAR с атрибутом AUTO_INCREMENT. Однако корректно будут работать как столбцы AUTO_INCREMENT только столбцы целого типа.
- Одним из главных предназначений механизма AUTO_INCREMENT является генерирование последовательности положительных целых величин, поэтому столбцы AUTO_INCREMENT могут объявляться с параметром UNSIGNED. Еще одно преимущество такого объявления заключается в том, что оно позволяет получить вдвое больше последовательных номеров.

  При определенных обстоятельствах с помощью атрибута AUTO_INCREMENT можно генерировать последовательность отрицательных величин. Но такое применение атрибута AUTO_INCREMENT не санкционировано, и результаты могут быть самые непредвиденные. Мои собственные эксперименты с различными версиями СУБД MySQL продемонстрировали непостоянные результаты. Таким образом, даже получив желаемые результаты для одной версии СУБД MySQL, при переходе на другую версию можно получить совершенно другие результаты. (Другими словами, попытка использовать атрибут AUTO_INCREMENT в каких-то других целях, отличных от создания последовательности положительных целых значений, может привести к совершенно непредвиденным результатам. Вас предупредили!)
- Не заблуждайтесь относительно того, что прибавление атрибута AUTO_INCREMENT в объявление столбца является чудесным способом получения неограниченной последовательности чисел. Это не так, последовательности AUTO_INCREMENT всегда ограничены диапазоном типа столбца. Например, при использовании столбца TINYINT максимальным номером последовательности будет 127. Достигнув этого предела, ваше приложение будет выдавать ошибки "дублирование ключа" ("duplicate key"). При использовании типа TINYINT UNSIGNED будет получен другой предел, равный 255.

- Полное обновление содержимого таблицы может обновить последовательность и снова начать ее с 1, даже для типов таблиц, которые не используют повторно значения AUTO_INCREMENT. Так произойдет при использовании любого из следующих операторов:

  ```
  DELETE FROM tbl_name;
  TRUNCATE TABLE tbl_name;
  ```

  Сброс последовательности происходит в результате того, что СУБД MySQL оптимизирует операцию удаления всей таблицы. Она отбрасывает записи и индексы и вновь воссоздает таблицу, а не удаляет строки построчно, поэтому все последовательные числа будут безвозвратно утеряны. Если возникнет необходимость удалить все записи с сохранением последовательной информации, такую оптимизацию можно отключить, воспользовавшись оператором DELETE с выражением WHERE, которое всегда равно значению "истина":

  ```
  DELETE FROM tbl_name WHERE 1;
  ```

  Это выражение вынудит СУБД MySQL оценивать условие для каждой записи и удалять их, таким образом, индивидуально.

Отключение повторного использования значений в последовательности

Что можно предпринять для того, чтобы обеспечить строгое возрастание порядковых чисел для типов таблиц, которые используют значения повторно? Одним из решений будет использование отдельной таблицы, предназначенной только для создания значений типа AUTO_INCREMENT, записи из которой никогда не будут удаляться. Таким образом, значения из этой таблицы никогда не будут использоваться повторно. Чтобы создать новую запись в основной таблице, сначала вставьте значение NULL в таблицу порядковых номеров. Затем вставьте запись в вашу основную таблицу, получив значение с помощью LAST_INSERT_ID() для столбца, который будет содержать порядковый номер:

```
INSERT INTO ai_tbl SET ai_col = NULL;
INSERT INTO main_tbl SET id=LAST_INSERT_ID() ... ;
```

Добавление столбца с порядковым номером в таблицу

Предположим, что вы создаете таблицу и вводите в нее информацию:

```
mysql> CREATE TABLE t (c CHAR(10));
mysql> INSERT INTO t VALUES('a'),('b'),('c');
mysql> SELECT * FROM t;
+------+
| c    |
+------+
| a    |
| b    |
| c    |
+------+
```

Затем в эту таблицу понадобилось добавить столбец, содержащий порядковый номер. Для этого воспользуйтесь оператором ALTER TABLE со столбцом

AUTO_INCREMENT, используя тот же синтаксис, который использовался в операторе CREATE TABLE:

```
mysql> ALTER TABLE t ADD i INT AUTO_INCREMENT NOT NULL PRIMARY KEY;
mysql> SELECT * FROM t;
+------+---+
| c    | i |
+------+---+
| a    | 1 |
| b    | 2 |
| c    | 3 |
+------+---+
```

Обратите внимание на то, как СУБД MySQL автоматически присвоила значения столбцу AUTO_INCREMENT. Самому так делать не надо.

Перенумерация существующего столбца

Если таблица уже имеет столбец, объявленный с параметром AUTO_INCREMENT, и вы хотите перенумеровать ее для того, чтобы свести к минимуму пробелы в нумерации, которые могли получиться после удаления строк, просто удалите столбец с последующим его воссозданием.

Когда СУБД MySQL добавляет столбец, она присваивает новые порядковые номера автоматически, как было показано в предыдущем примере. Предположим, что таблица t имеет следующий вид, где столбец i объявлен с параметром AUTO_INCREMENT:

```
mysql> CREATE TABLE t (c CHAR(10), i INT NOT NULL AUTO_INCREMENT PRIMARY
KEY);
mysql> INSERT INTO t ©
    ->
VALUES('a'),('b'),('c'),('d'),('e'),('f'),('g'),('h'),('i'),('j'),('k');
mysql> DELETE FROM t WHERE c IN('a','d','f','g','j');
mysql> SELECT * FROM t;
+------+----+
| c    | i  |
+------+----+
| b    |  2 |
| c    |  3 |
| e    |  5 |
| h    |  8 |
| i    |  9 |
| k    | 11 |
+------+----+
```

Оператор ALTER TABLE удаляет столбец, а затем добавляет его снова:

```
mysql> ALTER TABLE t
    -> DROP i,
    -> ADD i INT UNSIGNED AUTO_INCREMENT NOT NULL,
    -> AUTO_INCREMENT = 1;
mysql> SELECT * FROM t;
+------+---+
| c    | i |
+------+---+
| b    | 1 |
| c    | 2 |
```

```
| e    | 3 |
| h    | 4 |
| i    | 5 |
| k    | 6 |
+------+---+
```

Выражение AUTO_INCREMENT = 1 сбрасывает существующую последовательность и восстанавливает последовательность с 1.

Для таблиц формата MyISAM (или HEAP, начиная с версии MySQL 4.1) в качестве начального значения можно использовать значение, отличное от 1. Для других типов таблиц необходимо опустить выражение AUTO_INCREMENT, так как в этих типах задавать начальное значение последовательности таким образом нельзя. Последовательность все равно начнется с 1.

Заметим, что хотя вносить изменения в последовательности несложно, существует не столь уж много достаточно веских причин делать это. СУБД MySQL не "интересует", есть ли пропуск в последовательности, а пользователь не получает никаких выгод от пересортировки.

Генерирование последовательностей без атрибута AUTO_INCREMENT

Есть и другой способ генерации последовательностей — без применения атрибута AUTO_INCREMENT. Для этого используется вариант функции LAST_INSERT_ID() с аргументом. (Эта модификация функции была добавлена в СУБД MySQL версии 3.22.9.) Параметр expr в выражении LAST_INSERT_ID(expr) можно рассматривать как счетчик, созданный механизмом AUTO_INCREMENT, т.е. после добавления или модификации столбца с помощью функции LAST_INSERT_ID(expr) последующее обращение к функции LAST_INSERT_ID() вернет значение expr. Такой механизм позволит создать счетчик и затем использовать его в последующих операторах внутри одного сеанса работы с базой данных.

Одним из применений этой стратегии является создание таблицы с одной строкой, значение которой наращивается по мере необходимости. Создадим следующую таблицу:

```
CREATE TABLE seq_table (seq INT UNSIGNED NOT NULL);
INSERT INTO seq_table VALUES(0);
```

С помощью этих операторов создается и инициализируется таблица seq_table. В результате таблица содержит одну строку со значением seq, равным 0. Следующий номер генерируется так:

```
UPDATE seq_table SET seq = LAST_INSERT_ID(seq+1);
SELECT LAST_INSERT_ID();
```

Этот оператор UPDATE выбирает текущее значение столбца seq и увеличивает его на 1 для генерирования следующего значения последовательности. Генерирование нового значения с помощью функции LAST_INSERT_ID(seq+1) аналогично генерированию последовательностей механизмом AUTO_INCREMENT. Причем этот механизм закрепляется за конкретным клиентом.

Этот метод можно также использовать для создания отрицательных приращений или приращений больше 1. Например, первый оператор генерирует числовую последовательность с приращением, равным 100, второй — с приращением -1:

```
UPDATE seq_table SET seq = LAST_INSERT_ID(seq+100);
```

Повторение такого оператора генерирует числовую последовательность с приращением –1:

```
UPDATE seq_table SET seq = LAST_INSERT_ID(seq-1);
```

Этот метод можно развить и задать последовательность, начинающуюся с произвольного значения, присвоив столбцу `seq` соответствующее начальное значение.

Выше обсуждалась процедура установки счетчика с помощью таблицы с одной строкой. Это подходит для одного счетчика, но если вам потребуется несколько счетчиков, создание по одной таблице на счетчик приведет к излишнему дублированию таблиц. Предположим, например, что вам необходимо создать на нескольких страницах что-то вроде "доступ к этой странице осуществлялся *n* раз". Очевидно, что создание отдельных таблиц со счетчиками для каждой страницы будет невыгодно.

Одним из способов избежать создания нескольких таблиц со счетчиками является создание одной таблицы с несколькими столбцами. Один столбец хранит значение счетчика, а другой — имя, которое уникально идентифицирует каждый счетчик. Можно также использовать функцию `LAST_INSERT_ID()` и с помощью имени счетчика решить, к какому столбцу оно будет иметь отношение. Таблица имеет следующий вид:

```
CREATE TABLE counter
(
  name VARCHAR(255) BINARY NOT NULL,
  PRIMARY KEY (name),
  value INT UNSIGNED
);
```

Столбец `name` имеет строковый тип, что позволяет дать счетчику любое имя и объявить этот столбец с параметром `PRIMARY KEY`. Это позволяет избежать дублирования имен. При этом предполагается, что приложения, которые работают с этой таблицей, "знают" имена, которые они используют.

Для Web-счетчиков уникальность имен счетчиков обеспечивается за счет использования в качестве счетчика имен пути к странице в пределах дерева документов. Атрибут `BINARY` делает процедуру обработки имен пути чувствительной к регистру. (Его не следует использовать, если имена путей у системы не чувствительны к регистру.)

Для того чтобы воспользоваться таблицей `counter`, вставьте строку, соответствующую каждой странице, для которой требуется счетчик. Например, для установки значения нового счетчика для начальной страницы узла необходимо выполнить следующее:

```
INSERT INTO counter (name,value) VALUES('index.html',0);
```

Произойдет инициализация счетчика 'index.html' с присвоением ему значения 0. Для генерирования следующего порядкового значения для страницы воспользуйтесь именем его пути, чтобы просмотреть правильное значение счетчика и увеличить его с помощью функции `LAST_INSERT_ID(expr)`. После этого можно осуществить поиск значения с помощью функции `LAST_INSERT_ID()`:

```
UPDATE counter SET value = LAST_INSERT_ID(value+1) WHERE name =
'index.html';
SELECT LAST_INSERT_ID();
```

Увеличить счетчик можно и без использования функции LAST_INSERT_ID():

```
UPDATE counter SET value = value+1 WHERE name = 'index.html';
SELECT value FROM counter WHERE name = 'index.html';
```

Однако такой прием может корректно не сработать, если в промежуток времени между вашими командами UPDATE и SELECT кто-то другой не увеличит этот счетчик. Эту проблему можно решить, воспользовавшись механизмом транзакций или заключив оба оператора в скобки LOCK TABLES и UNLOCK TABLES для блокировки других пользователей во время использования счетчика вами. Очевидно, что метод использования функции LAST_INSERT_ID() сделает это же значительно проще. Поскольку значение зависит от клиента, вы всегда будете иметь значение, вставленное вами, а не полученное другим клиентом, при этом не потребуется усложнять код, применяя транзакции или блокировки, чтобы заблокировать других клиентов.

Поддержка кодировок

Значения данных, базирующиеся на заданном наборе символов, интерпретируются с учетом заданной кодировки, определяющей допустимые используемые символы. Кодировки также имеют определенный порядок сортировки, который влияет на различные типы операций, выполняемые над символическими значениями:

- операции сравнения: <, <=, =, <>, >= и >;
- операции сортировки: ORDER BY, MIN(), MAX();
- операции группировки: GROUP BY, DISTINCT.

Кодировка также влияет на разнообразные аспекты работы сервера, такие как диапазон символов, которые могут быть использованы в именах баз данных, таблиц и столбцов, так как имена обычно составляются из алфавитно-цифровых символов, входящих в стандартную кодировку сервера (подробно об этом — в разделе "Правила присвоения имен в СУБД MySQL" в главе 3, "Синтаксис и использование языка SQL").

Уровень доступной поддержки кодировки зависит от используемой версии СУБД MySQL. До версии 4.1 сервер может работать только с одной кодировкой. Начиная с СУБД MySQL 4.1 сервер может поддерживать несколько кодировок одновременно, и кодировка может задаваться на самых различных уровнях: на уровне сервера, баз данных, таблиц, столбцов или строковых констант. Теперь, например, возможно, чтобы в таблице с кодировкой latin1, принятой по умолчанию, были представлены столбцы, работающие с кодировкой на иврите и греческом языке. Предусмотрены также механизмы определения работающих кодировок или преобразования из одной кодировки в другую.

В этом разделе описывается использование кодировок, поддерживаемых серверами. Для того чтобы узнать настройки сервера, необходимые для поддержки нужных кодировок, обратитесь к приложению B, "Операторы и функции". В данном приложении также рассказывается, что необходимо сделать при обновлении СУБД MySQL до версии 4.1, чтобы использовать новые возможности со старыми таблицами.

Поддержка кодировок до версии 4.1

До версии 4.1 в СУБД MySQL возможность объявления кодировок явным образом не предусматривалась. Строковые константы и значения столбцов интерпретировались с учетом кодировки, принятой на сервере по умолчанию. По умолчанию это была кодировка, принятая при создании сервера (обычно это была кодировка `latin1`), но встроенное значение можно отменить при выполнении параметром `--default-character-set`. Это просто, но в известной мере ограничивает свободу действий. Так, вы не сможете работать с таблицами, в которых столбцы рассчитаны для работы с различными кодировками.

Модель, рассчитанная на работу с одной кодировкой, при изменении кодировки сервера может также привести к возникновению проблем с индексами для уже созданных таблиц и загрузки в них данных. Они возникают из-за того, что значения индексов хранятся в порядке, отсортированном в соответствии с сортировкой, принятой для кодировки, которая была в силе на момент создания записей индекса. Некоторые кодировки имеют различный порядок сортировки. Поэтому при смене кодировки таблицы вполне вероятно, что порядок сортировки записей индекса больше не будет соответствовать порядку следования символов, принятому для новой кодировки. Еще хуже то, что при добавлении к таблице новых строк индекс, созданный с учетом порядка сортировки, принятым для первоначальной кодировки, будет обновлен с учетом новой кодировки. Вследствие этого запросы, в которых используется индекс, будут работать неправильно.

Решением этой проблемы является перестройка индексов для каждой существующей таблицы, что приводит к использованию в индексах порядка сортировки новой кодировки. Таблицы могут быть преобразованы с применением самых различных методик.

- Заархивируйте таблицу с помощью программы `mysqldump`, удалите ее, а потом восстановите из архива. Эта операция приводит к перестройке индексов в процессе перезагрузки таблицы. Данный метод применим для таблиц всех типов.
- Удалите индексы и воссоздайте их (например, с помощью команды `ALTER TABLE` или команд `DROP INDEX` и `CREATE INDEX`). Это применимо для таблиц любых типов, но требует точного знания определений индексов, чтобы воссоздать их правильно.
- Для таблиц типа MyISAM существует возможность перестройки индексов с помощью программы `myisamchk` с применением параметров `--recover` и `--quick` вместе с параметром `--set-character-set`, которые определяют используемую кодировку. Аналогичный эффект обеспечивает программа `myisamchk` с заданием параметров `--repair` и `--quick` или оператора `REPAIR TABLE` с параметром `QUICK`. Программа `myisamchk` и оператор `REPAIR TABLE` более удобны в использовании, так как их выполняет сервер, который "знает", какую кодировку при этом использовать. Программа `myisamchk` работает и с отключенными таблицами, при этом кодировка объявляется явным образом.

Несмотря на множество методов, имеющихся для переопределения индексов при изменении кодировки, сам факт существования такой необходимости достаточно раздражает. К счастью, в СУБД MySQL 4.1 эта проблема была решена.

Поддержка различных кодировок в СУБД MySQL 4.1 и более поздних версиях

Механизм поддержки кодировок был существенно пересмотрен в СУБД MySQL 4.1, что позволило обеспечить поддержку следующих возможностей.

- Одновременная поддержка работы с несколькими кодировками.
- Возможность задания кодировок как на уровне сервера, так и на уровне базы данных, таблицы, столбца строковой константы:
  - для задания кодировки базы данных используется оператор `ALTER DATABASE`;
  - для задания кодировки таблицы или столбца используются операторы `CREATE TABLE` и `ALTER TABLE`.
- Функции и операторы для преобразования отдельных значений из одной кодировки в другую или для определения кодировки отдельного значения.
- Оператор `COLLATE` для обработки значений в одной кодировке, как имеющей порядок сортировки другой кодировки.
- Оператор `SHOW CHARACTER SET`, позволяющий выводить кодировки, известные серверу.
- Автоматическая пересортировка при изменении кодировки.
- Поддержка кодировки Unicode с помощью кодировок `utf8` и `ucs2`.
- Много новых кодировок.

Нельзя смешивать различные кодировки в пределах одной строки или использовать различные кодировки в различных строках одного столбца. Однако использование кодировки Unicode (которая поддерживает кодировки различных языков в одной кодировке) позволяет задавать поддержку множества языков в нужном вам типе.

Определение кодировок

Кодировку можно объявить на различных уровнях, начиная с кодировки по умолчанию, принятой на уровне сервера, и заканчивая отдельными строками.

- Кодировка на уровне сервера, принятая по умолчанию, создается во время компиляции сервера, но ее можно отменить во время запуска сервера с помощью параметра—`defaultcharacter-set`.
- Для объявления кодировки для базы данных можно воспользоваться следующим оператором:

  `ALTER DATABASE` *`db_name`* `DEFAULT CHARACTER SET` *`charset`*`;`

  где *`charset`* — имя поддерживаемой кодировки или значение `DEFAULT`. Значение `DEFAULT` отражает тот факт, что база данных не имеет кодировки, объявленной явным образом, и в этом случае сервер делает объявление кодировки на уровне базы данных, ссылаясь на стандартную кодировку, принятую на сервере.

- Для объявления стандартной кодировки для таблицы во время создания таблицы необходимо воспользоваться параметром CHARACTER SET:

  ```
  CREATE TABLE tbl_name (...) CHARACTER SET = charset;
  ```

  где *charset* — имя поддерживаемой кодировки или значение DEFAULT. Значение DEFAULT отражает тот факт, что сервер будет принимать решение о кодировке таблицы, ссылаясь на стандартную кодировку, принятую на уровне базы данных.

- Кодировка столбцов таблицы может быть присвоена явным образом с помощью параметра CHARACTER SET. Например:

  ```
  c CHAR(10) CHARACTER SET charset
  ```

  в этом случае *charset* должно быть именем поддерживаемой кодировки и не может быть значением DEFAULT. Однако атрибут CHARACTER SET можно опустить, если будет использоваться кодировка, принятая на уровне таблицы. Типы столбцов, для которых задается кодировка, могут иметь типы CHAR и VARCHAR (при объявлении без атрибута BINARY) и тип TEXT.

- Строковые константы преобразуются в заданную кодировку следующей командой (где *charset* должно быть именем поддерживаемой кодировки):

  ```
  _charset str
  ```

  Вот примеры отображения строк в кодировках latin1_de и utf8:

  ```
  _latin1_de 'abc'
  _utf8 'def'
  ```

  Эта форма записи приемлема только для литеральных строк и не приемлема для шестнадцатеричных констант, строковых выражений и значений столбцов. Однако любая строка может быть преобразована в определенную кодировку с помощью функции CONVERT():

  ```
  SELECT CONVERT(str USING charset);
  ```

Существует также возможность сортировать значения определенной кодировки в соответствии с порядком сортировки другой кодировки с помощью оператора COLLATE. Например, если столбец c является столбцом с кодировкой latin1 и при этом требуется, чтобы ее значения сортировались в соответствии с правилами сортировки, принятыми для кодировки latin1_de, необходимо сделать следующее:

```
SELECT c FROM t ORDER BY c COLLATE latin1_de;
```

Определение активной кодировки

Поддержка кодировок в СУБД MySQL версии 4.1 и выше обусловила также появление операторов, которые позволяют получить информацию на нескольких уровнях.

- С помощью следующего запроса можно определить, какие кодировки поддерживаются на сервере:

  ```
  SHOW CHARACTER SET;
  ```

  Для того чтобы определить, какая кодировка является для сервера стандартной, можно воспользоваться таким запросом:

  ```
  SHOW VARIABLES LIKE 'character_set';
  ```

- Кодировку на уровне базы данных можно определить с помощью оператора следующего вида:

  ```
  SHOW CREATE DATABASE db_name;
  ```

  Если в результате работы этого оператора не будет получено имя кодировки, это означает, что кодировка базы данных никогда не определялась или была задана явным образом как DEFAULT.

- Кодировку таблицы можно определить двумя способами:

  ```
  SHOW CREATE TABLE tbl_name;
  SHOW TABLE STATUS LIKE 'tbl_name';
  ```

- Индивидуальные установки кодировок для столбцов таблиц можно определить с помощью одного из следующих операторов:

  ```
  DESCRIBE tbl_name;
  SHOW COLUMNS FROM tbl_name;
  SHOW CREATE TABLE tbl_name;
  ```

- Для определения кодировки, с которой связаны строка, строковое выражение или значение столбца, воспользуйтесь функцией CHARSET():

  ```
  SELECT CHARSET(str);
  ```

Поддержка кодировки Unicode

Одна из причин существования такого большого количества кодировок заключается в том, что эти кодировки были созданы для поддержки алфавита различных языков. Это позволяет избежать ряда проблем. Например, определенный символ, который является общим для различных кодировок, может представляться различными кодами. Кроме того, в разных языках для представления символов требуется различное число байт. Для кодировки Latin-1 (латинский алфавит) вполне достаточно одного байта, чего не скажешь о таких языках, как японский и китайский, которые имеют так много символов, что для их отображения одного байта будет недостаточно.

Кодировка Unicode была создана с целью унифицировать систему кодировки символов, которая позволила бы отображать алфавиты всех языков одинаково. Для СУБД MySQL поддержка кодировки Unicode обеспечивается с помощью двух кодировок.

- Кодировка UTF-8 является форматом переменной длины, в котором символы представляются с помощью 1–4 символов. (UTF — аббревиатура от UCS Transformation Format, где UCS означает Universal Character Set.) Кодировка `utf8` в СУБД MySQL не включает 4-байтовых символов, хотя и предусматривает такую возможность в будущем.
- Кодировка `ucs2` представляет каждый символ с помощью двух байтов, при значащем первом байте. Эта кодировка не может отражать символы, для хранения которых требуется больше двух байтов.

Преобразование таблиц из старых форматов в формат MySQL 4.1

При обновлении сервера СУБД MySQL до версии 4.1 или выше старые таблицы могут использоваться, однако при этом нельзя будет в полной мере использо-

вать усовершенствованную поддержку кодировок. Для того чтобы получить такую возможность, необходимо преобразовать их в формат версии 4.1 (о том, как это сделать, рассказывается в приложении В, "Операторы и функции").

Выбор типа столбца

В разделе "Типы столбцов СУБД MySQL" были рассмотрены различные типы столбцов СУБД MySQL и их общие свойства, такие как вид значений, которые они могут хранить, объем требующейся памяти и т.д. Но как происходит выбор типа столбца при создании таблицы на практике? В этом разделе обсуждаются вопросы, которые необходимо рассмотреть в процессе выбора типа данных.

Чаще всего применяются столбцы строкового типа. В них можно хранить любые значения. В виде строки можно представлять как числа, так и даты. Почему бы просто не объявить все столбцы строковыми? Что при этом произойдет?

В этом случае потребуется больше памяти, так как числовой формат эффективнее строкового. Из-за различия в обработке чисел и строк запросы будут интерпретироваться по-разному. Число 2 меньше числа 11, но строка "2" больше строки "11". Эту проблему можно разрешить, преобразовав строку в число:

```
SELECT col_name + 0 As num ... ORDER BY num
```

Прибавление нуля сделает сортировку сортировкой цифровых значений. Но есть ли смысл так поступать? Вероятно, нет. Обработка строкового значения как цифрового создает массу сложностей. Так, потребуется преобразовать каждый столбец из строкового формата в цифровой формат. Это неэффективно. Если этот столбец будет задействован в вычислениях, то нельзя индексировать таблицу по данному столбцу, поскольку это замедлит обработку запросов. Такого снижения эффективности не произойдет, если первыми столбцами таблицы будут цифровые столбцы. Вот так простое решение по выбору типа представления данных окажет влияние на объем памяти, обработку запросов, производительность системы.

Следующий пример демонстрирует некоторые моменты процесса принятия решения по выбору типа столбца. Ниже кратко перечислены основные вопросы, которые нужно учитывать в процессе принятия решения о выборе типа столбца.

- **Данные какого типа будут храниться в столбце?** Числа? Строки? Даты? Несмотря на то что вопрос очевиден, его необходимо рассмотреть детально. Конечно, в виде строки можно представить любые данные. Однако только что было продемонстрировано, что для цифровых данных эффективнее использовать соответствующие типы данных. (Это же справедливо и в случае с датами и временем.) Однако судить однозначно о типе данных очень трудно. Особенно когда это чьи-то данные. Очень важно задать будущему хозяину данных вопросы о природе данных, которые будет содержать создаваемая таблица.
- **Каков диапазон значений?** Если это целые числа, будут ли они всегда неотрицательными? Если да, то можно воспользоваться параметром UNSIGNED. Если это строковые данные, попадают ли они в определенный диапазон значений? Если да, то можно воспользоваться типом SET.

Существует определенная взаимосвязь между диапазоном типа и объемом памяти, который требуется для хранения данных такого типа. Насколько "объемные" данные вам потребуются? В случае чисел можно выбрать как сверхмалые типы с ограниченным диапазоном значений, так и сверхбольшие с практически неограниченным диапазоном значений. В случае строки их можно сделать большими или маленькими и не объявлять `CHAR(255)`, если доподлинно известно, что все они будут иметь меньше 10 символов.

- **Как повысить производительность и эффективность?** Очевидно, что одни типы обрабатываются эффективнее других. Так, операции над цифровыми данными выполняются быстрее операций над строками, а короткие строки сравниваются быстрее длинных. Производительность для таблиц со строками фиксированной длины выше, чем для таблиц со строками переменной длины.
- **Каким образом будут сравниваться значения?** Операции сравнения строк могут быть чувствительны к регистру. Выбор типа может влиять на сортировку, которая также базируется на сравнении.
- **Будете ли вы индексировать столбец?** Если да, то это влияет на выбор типа столбца. Некоторые типы данных и типы таблиц не индексируются. Это типы `BLOB` и `TEXT` в таблицах типа ISAM; кроме того, индексированные столбцы не могут объявляться с параметром `NOT NULL`. В свою очередь это влияет на возможность присваивать пустые значения.

Теперь рассмотрим более детально каждый из вышеперечисленных аспектов. Но перед этим хочется подчеркнуть один момент. При создании таблиц мы стремимся к тому, чтобы они имели оптимальный тип, но если на этом этапе была допущена ошибка, это не беда. Ошибку можно исправить позднее с помощью оператора `ALTER TABLE`. Так, например, если данные занимают больше места, чем предполагалось, можно заменить тип `SMALLINT` типом `MEDUIMINT`. Задача может быть и сложнее: заменить тип `CHAR` типом `ENUM`. В версиях после 3.22 для получения информации о столбцах таблицы можно воспользоваться функцией `PROCEDURE ANALYSE()`. С помощью этой функции можно выяснить, чему равняется максимальное и минимальное значения диапазона, и определить оптимальный тип, который покроет весь диапазон.

```
SELECT * FROM tbl_name PROCEDURE ANALYSE();
```

Это поможет выбрать более компактный тип, что в свою очередь позволит оптимизировать производительность запросов, которые будут обрабатывать данные таблицы, и уменьшить объем памяти, требующейся для хранения данных таблицы.

Данные какого типа будут храниться в столбце?

Первый вопрос, который возникает в процессе выбора типа данных столбца, – это какие данные будет содержать данный столбец. Обычно происходит самое очевидное: числа хранятся в столбцах числового типа; строки – в столбцах строкового типа; даты и время – в столбцах календарного типа и т.д. Когда число имеет дробную часть, логичнее будет выбрать не целочисленный, а вещественный тип. Но мо-

гут быть и исключения. Главным критерием здесь может быть только глубокое понимание природы данных. Это позволит выбирать тип данных, полностью вооружившись знанием о предмете. Очевидно, что при работе с собственными данными, природа которых вам хорошо известна, никаких проблем не возникает. Но когда таблица создается для других — это совсем другое дело. Очень непросто узнать, с чем предстоит работать. Необходимо задать большое количество вопросов, чтобы узнать, какой должна быть структура базы данных.

Предположим, что нужен столбец для хранения информации о "прогнозируемом количестве осадков". Это будет число? Или это можно представить "частично цифровым" форматом, т.е. довольно часто, но не всегда представленное числом? Достаточно вспомнить, например, как при объявлении прогноза погоды всегда называется ожидаемая мера. Иногда это число ("25 сантиметров осадков"), но иногда это укладывается в простую, достаточно размытую формулировку ("незначительные осадки"). Это нормально для прогноза погоды, но как быть с такими данными в базе данных? В этом случае необходимы цифровой столбец для хранения числа или строковые данные для хранения текстовой информации? Возможен комплексный подход, когда в первом случае в двух столбцах будет храниться числовое значение, и текстовый столбец — для хранения текстовой информации во втором случае. При этом обязательно будет заполнен один из этих столбцов, а второй может быть пустым. Может показаться, что совершенно очевидно, что такой ситуации по возможности следует избегать, поскольку это ухудшит читаемость таблицы и затруднит создание запросов.

Попытаемся хранить все строки в цифровой форме, а затем преобразовать их при выводе на печать. Например, любое ненулевое значение ожидаемой величины меньше 0,01 см можно отобразить как отслеживаемое число:

```
SELECT IF(precip>0 AND precip<.01, 'trace', precip) FROM ....
```

Некоторые значения являются однозначно цифровыми, и для них необходимо проверить единицы измерения и требующуюся точность. Достаточно ли целого числа или необходимо как-то хранить дробную часть числа? От этого зависит выбор между целым и вещественным типом. Например, вес можно хранить в килограммах, отбросив информацию о граммах. При желании хранить дробную часть, можно воспользоваться вещественным форматом. В некоторых случаях можно прибегнуть к вычисляемым полям, например, если необходимо хранить данные о весе в килограммах и граммах.

Высоту и рост можно привести в качестве другого примера данных, которые можно представить разными способами.

- **Строка типа "6–2" для хранения значения "6 футов, 2 дюйма".** Такое представление будет легко читаемым, но его будет трудно использовать для выполнения математических операций.
- **Одно цифровое поле используется для хранения футов, другое — для дюймов.** Это представление информации легко использовать в цифровых операциях, но труднее использовать, чем значение, хранящееся в одном поле.
- **Одно цифровое поле для хранения высоты в дюймах.** Это наиболее удобное представление для базы данных, но наиболее трудное для восприятия человека. При отображении данные могут быть представлены в формате, отличном от формата, в котором они хранятся в базе данных.

Данные можно переформатировать в более понятную форму, используя различные функции СУБД MySQL. Это может оказаться наиболее удобным вариантом хранения информации о росте или высоте.

При денежных расчетах вы работаете со значениями, которые включают доллары и центы. Их можно представить в вещественном формате, но типы FLOAT и DOUBLE округляются и могут не подойти по точности. Люди очень щепетильны в денежных вопросах, поэтому потребуется тип, обладающий исключительной точностью. В распоряжении разработчика базы данных будет несколько вариантов решений.

- Деньги можно представить типом DECIMAL(M,2), выбрав при этом такое значение *M*, которое бы полностью удовлетворяло диапазону необходимых значений. Это будут вещественные значения и без десятичной части. Преимущество типа DECIMAL(M,2) заключается в том, что значения хранятся в виде строки и не округляются. Недостаток заключается в следующем: операции со строками менее эффективны, чем операции со значениями, внутренне представленными числами.
- Однако все денежные значения можно задавать в центах, которые довольно легко представить с помощью целочисленного типа данных. Здесь преимущество заключается в том, что целочисленные операции производятся очень быстро. Неудобство — в том, что всякий раз при вводе-выводе данных необходимо производить деление/умножение на 100.

И еще одна проблема. Если хранится информация о дате, включают ли эти данные информацию о времени? Другими словами, будут ли они *вообще* содержать информацию о времени? СУБД MySQL не имеет типа данных, содержащего факультативную временную часть: тип DATE не хранит информацию о времени, а DATETIME должен содержать информацию о времени. Это значит, что если информация о времени может отсутствовать, дату можно хранить в столбце типа DATE, а время — в столбце TIME. При этом столбец TIME будет объявлен с параметром NULL:

```
CREATE TABLE mytbl
(
   date DATE NOT NULL,    # дата обязательна
   time TIME NULL         # время факультативно (может быть NULL)
)
```

Особенно тщательно нужно определиться с необходимостью хранения данных о времени в столбце, если с помощью этого столбца будет осуществляться объединение двух таблиц.

Предположим, что вы проводите серию опытов. Сначала проводится стандартная первая серия опытов. В тот же день на основании результатов, полученных в ходе первой серии опытов, может понадобиться провести серию дополнительных опытов. Эту информацию можно представить связью "заголовок-деталировка", где информация о тестируемом объекте и результаты стандартной первой очереди опытов хранятся в заголовке, а информация о дополнительных опытах — в строках деталировочной таблицы. Объединить эти таблицы можно по двум параметрам: по идентификатору объекта и дате проведения опытов.

В этом случае необходимо четко определить, можно ли ограничиться одной датой, или понадобятся и дата, и время проведения опыта. Это зависит от того,

проводится ли опыт над одним и тем же предметом несколько раз в день. Если да, то сохраняем время (скажем, момент начала опыта) в столбце типа DATETIME или в отдельных столбцах типов DATE и TIME. Без данных о времени будет невозможно отличить один опыт от другого, особенно если в день проводится несколько опытов над одним объектом.

Автору уже приходилось слышать такие заявления от постановщиков задачи: "Меня не интересует время; я никогда не тестирую объект дважды в день". Иногда это так, но чаще приходилось сталкиваться с возникающей впоследствии ситуацией, когда уже невозможно разобраться с данными, перемешанными в деталировке. Остается только сожалеть. Момент упущен, и уже ничего нельзя исправить.

Этого можно было избежать, добавив столбец TIME в таблицу. Данные о времени проведения опытов можно взять из источника независимой информации (например, из протоколов, зафиксированных на бумаге). К сожалению, эта процедура очень трудоемка, но другого способа устранения неоднозначности не существует. Даже если существует источник независимой информации, это не поможет полностью устранить неоднозначность и приведет к ошибкам в работе уже написанной программы, обрабатывающей информацию из таблиц. Так что всегда лучше объяснить проблему заказчику и предпринять все возможное на этапе проектирования базы данных.

Возможен вариант неполных данных, что тоже может повлиять на выбор типа столбцов. Так, например, в процессе поиска дат рождения и смерти для создания генеалогического дерева можно не найти точной даты рождения или смерти, и, к примеру, установить только год рождения. В столбце DATE хранить такую дату рождения не получается, так как в ранних версиях СУБД MySQL столбец такого типа может хранить только полную дату. В качестве одного из вариантов решения такой проблемы можно предложить следующее: хранить год, месяц и день в отдельных столбцах. В случае неполной даты можно хранить известное значение в соответствующем столбце, а остальные столбцы оставить пустыми. Начиная с СУБД MySQL версии 3.23 формат типа DATE позволяет хранить даты с нулевыми значениями дня или месяца. Такие "размытые" даты можно использовать для представления значений неполных дат.

Каков диапазон значений?

После того как была выбрана категория типов данных для столбца, нужно определиться с диапазоном допустимых значений. Это поможет в выборе оптимального типа данных. Если ваши данные принимают значения в диапазоне от 0 до 1000, можно использовать любой тип, лежащий в диапазоне от SMALLINT до BIGINT. Если же ваши данные принимают значения до двух миллионов, лучше всего подойдут типы от MEDIUMINT до BIGINT.

Можно просто выбрать самый большой из возможных типов и больше ни о чем не беспокоиться (например, тип BIGINT, если говорить о типах, упомянутых в предыдущих абзацах). Однако следует использовать минимальный тип изо всех максимально больших. Таким образом, можно минимизировать объем дискового пространства, необходимого для хранения таблиц. Кроме того, это позволит повысить производительность, поскольку короткие столбцы всегда обрабатываются быстрее длинных.

Если диапазон значений неизвестен, можно немного подумать или, чтобы не думать, выбрать наихудший вариант — воспользоваться самым "вместительным" типом BIGINT. (Напомню, что не все потеряно, даже если в момент создания таблицы была допущена ошибка в выборе типа. Для увеличения размера столбца можно воспользоваться оператором ALTER TABLE.)

В главе 1, "Знакомство с СУБД MySQL и SQL", для проекта "Контроль успеваемости" была создана таблица score, которая содержит столбец score, предназначенный для хранения результатов тестов и викторин. Для простоты таблица была создана с типом INT, но теперь очевидно, что значения результатов лежат в пределах от 0 до 100 и что оптимальным типом будет TINYINT UNSIGNED, так как для него требуется меньше памяти.

Диапазон значений данных влияет также на перечень параметров, указываемых при объявлении типа столбца. При всех положительных значениях можно воспользоваться параметром UNSIGNED.

Строковые данные не имеют понятия "диапазона" значений, как это имеет место для цифровых типов, но они характеризуются длиной. А максимальная длина зависит от типа. Так, типы CHAR, VARCHAR, TINYTEXT и TINYBLOB могут иметь строки, содержащие не больше 256 символов. Для хранения более длинных строк можно воспользоваться типами TEXT и BLOB; типы CHAR и VARCHAR в данном случае не подойдут.

Строковые данные, являющиеся перечнем фиксированных значений, можно представить типом ENUM или SET. Эти типы имеют внутреннее цифровое представление. Так как над этим типом производятся цифровые операции, то они будут эффективнее любого другого строкового типа. Кроме того, эти типы данных позволят сэкономить память.

Здесь надо сделать небольшое отступление, связанное с определением диапазона данных в момент постановки задачи, и подчеркнуть, что чаще всего приходилось сталкиваться с такими описаниями диапазонов значений данных, как "всегда" и "никогда" (например, "всегда меньше 1000" или "никогда не принимает отрицательных значений"). Особенно будьте внимательны на этапе постановки задачи, когда в лексиконе постановщика появляются такие термины. Необходимо убедиться, что в действительности имеется в виду, когда говорится "всегда" или "никогда". Опыт показывает, что, говоря "всегда", люди часто имеют в виду "почти всегда".

Предположим, что вы проектируете таблицу. Постановщик задачи говорит: "Результаты наших тестов всегда лежат в диапазоне чиссл от 0 до 100". На основании такого утверждения вы, будучи опытным разработчиком, выбираете тип TINYINT с параметром UNSIGNED. Затем вы узнаете, что в случае пропуска экзамена по болезни присваивается значение -1. Вот так! Но об этом же никто даже не вспомнил? Для представления такого значения можно воспользоваться пустым значением, а если нет, придется отказаться от использования параметра UNSIGNED. Все это можно сделать постфактум с помощью оператора ALTER TABLE.

Иногда решения по выбору типов данных можно сделать на основании простого вопроса: "Существуют *хоть какие-то* исключения?" Если таковые появляются, пусть даже однажды, их надо предусмотреть. Конечно, есть люди, которые считают, что исключения, возникающие не очень часто, можно не принимать во внимание. Но в

процессе создания базы данных такой образ мыслей недопустим. Вопрос должен заключаться не в том, как часто возникают исключительные ситуации, а возникают ли они вообще. Если они возникают, то их необходимо предусмотреть.

Как повысить производительность и эффективность

Выбор типа данных влияет на производительность СУБД MySQL. Рекомендую принимать во внимание доводы, приведенные ниже. Они помогут вам повысить производительность обработки таблиц СУБД MySQL.

Сравнение обработки цифровых и строковых данных

В целом обработка цифровых данных происходит быстрее, чем обработка строковых данных. Для сравнения чисел достаточно одной операции. Операция сравнения строк может потребовать несколько побайтовых операций (и тем больше, чем больше размер строки).

Можно воспользоваться преимуществом числовых операций, работая со строками, применив строковой тип ENUM или SET, который имеет внутреннее цифровое представление.

Рассмотрим альтернативное представление строк. Производительность можно увеличить, представив строковые значения в виде чисел. Например, IP-адрес можно представить в строковой нотации. Альтернативным решением может быть представление каждой тетрады в одном байте четырехбайтового типа INT UNSIGNED. Это позволит сэкономить дисковое пространство и ускорит процедуру просмотра данных. Однако такое представление IP-адреса затруднит проверку по шаблонам. Это значит, что в процессе выбора типа данных необходимо все аспекты применения данных рассматривать комплексно. (Функции INET_ATON() и INET_NTOA(), независимо от вашего выбора, могут помочь производить преобразования между двумя представлениями.)

Типы данных большого и малого размера

Очевидно, что данные, занимающие меньше памяти, обрабатываются быстрее данных, требующих больше памяти. Это может происходить по одной причине: они занимают меньше места и при обработке меньше загружают диск. Время обработки строк напрямую зависит от длины строки.

В целом, чем меньше таблица, тем быстрее она будет обрабатываться, так как обработка запросов ускоряется из-за уменьшения количества операций ввода-вывода на диске. Для повышения производительности рекомендуется использовать типы фиксированной длины, выбирать типы минимальных размеров, достаточных для хранения данных. Например, не применяйте тип BIGINT, если вполне достаточно типа MEDIUMINT. Не используйте тип DOUBLE, если достаточно типа FLOAT. Для типов с переменной длиной это тоже справедливо. Так, тип BLOB резервирует 2 байта для хранения размера, в то время как тип LONGBLOB уже потребует 4 байта для этих же целей. Это означает, что при хранении данных, размер которых никогда не превысит 64 Кбайт, использование типа BLOB сэкономит 2 байта на каждое значение (аналогичные выводы применимы в отношении типа TEXT).

Типы фиксированной и переменной длины

Типы фиксированной и переменной длины характеризуются различными проблемами при обработке, хотя конкретное влияние зависит от типа таблицы.

В целом для таблиц MyISAM и ISAM типы фиксированной длины обрабатываются быстрее типов переменной длины.

- В случае со столбцами переменной длины при частых удалениях и модификациях есть опасность сильной фрагментации табличного пространства. Фрагментация потребует частого запуска команды `OPTIMIZE TABLE`. Такая проблема полностью отсутствует при использовании типов данных с фиксированной длиной.
- Таблицы со столбцами фиксированной длины, в отличие от таблиц со столбцами переменной длины, проще восстанавливать после разрушения. Эту проблему нельзя отнести к проблемам, связанным с производительностью, но это поможет значительно уменьшить временные затраты на восстановление данных.

Преобразование таблиц MyISAM и ISAM со столбцами переменной длины в таблицы со столбцами фиксированной длины повысит производительность обработки таких таблиц. Но прежде чем это сделать, необходимо учесть следующие аспекты.

- Работа с таблицами, имеющими столбцы фиксированной длины, имеет определенные недостатки. Они быстрее, но требуют больше дискового пространства. Так, тип `CHAR(n)` всегда требует для себя *n* байт на значение (даже если они будут пустыми или неполными). В то же время тип `VARCHAR(n)` требует столько пространства, сколько требуется для хранения каждого значения, плюс 1 байт для хранения длины значения. Поэтому, выбирая между типами `CHAR` и `VARCHAR`, вы делаете выбор между экономией пространства и экономией времени. Если превалирует фактор производительности, выбирайте тип `CHAR`, если фактор пространства — выбирайте `VARCHAR`.
- Изменение только одного столбца переменной длины ничего не дает, здесь возымеет действие только изменение типов всех столбцов. Это позволяет сделать оператор `ALTER TABLE`. В противном случае все усилия окажутся напрасными.
- В некоторых случаях использование типов с фиксированной длиной просто невозможно, например, при длине строк, превышающей 255 символов.

Для таблиц типа InnoDB записи переменной и фиксированной длины хранятся одинаковым образом (имеется заголовок строки, хранящий указатели на отдельные значения столбцов, плюс пространство, необходимое для хранения значений). Это означает, что в данном случае строки фиксированной длины обрабатывать не проще. Следовательно, главнейшим фактором обработки здесь становится объем памяти, необходимый для хранения строки. Единственным затруднением является то, что для хранения строк переменной длины в таблицах типа InnoDB требуется меньше памяти, а значит, и меньше операций ввода-вывода.

Индексируемые типы

Наличие индекса ускоряет работу запросов. Поэтому выбирайте типы, которые можно индексировать. Детальную информацию об этой проблеме можно найти в разделе "Будете ли вы индексировать столбец?" этой главы.

Пустые и непустые типы

СУБД MySQL не проверяет на пустые значения столбцы, объявленные как NOT NULL, что позволяет сэкономить один бит в строке. Чем проще таблица, тем быстрее она будет обрабатываться. Отсутствие пустых значений в столбцах позволит не заботиться об обработке пустых значений как об особом типе данных.

Каким образом сравниваются значения

В зависимости от объявления строковые типы сравниваются и сортируются как без анализа регистра, так и с его анализом. В табл. 2.15 приведены нечувствительные к регистру строковые типы и их аналоги, чувствительные к регистру. Некоторые типы могут быть дополнительно объявлены как двоичные (CHAR, VARCHAR). Для этого указывается ключевое слово BINARY. "Двоичность" других типов (BLOB, TEXT) вытекает из самой природы типа.

Таблица 2.15. Чувствительность строковых типов к регистру

Двоичные типы (чувствительны к регистру)	Недвоичные типы (не чувствительны к регистру)
CHAR(M) BINARY	CHAR(M)
VARCHAR(M) BINARY	VARCHAR(M)
TINYBLOB	TINYTEXT
BLOB	TEXT
MEDIUMBLOB	MEDIUMTEXT
LONGBLOB	LONGTEXT

Столбцы могут предположительно использоваться как в чувствительных к регистру операциях, так и в нечувствительных к регистру операциях сравнения. Тут может помочь параметр BINARY. Такая строка будет обрабатываться как двоичная строка. Например, если столбец mycol имеет тип CHAR, то его можно сравнивать разными способами. Следующее сравнение не будет реагировать на регистр:

```
mycol = "ABC"
```

Ниже приведены две операции сравнения, которые реагируют на регистр (обратите внимание на то, что не имеет никакого значения тот факт, к какой из строк применен оператор BINARY):

```
BINARY mycol = "ABC"
mycol = BINARY "ABC"
```

Если необходимо отсортировать данные не в лексикографическом порядке, воспользуйтесь типом ENUM. Сортировка этого строкового типа происходит в со-

ответствии с порядком значений столбца, указанного в объявлении. Это позволяет настраивать порядок сортировки по желанию.

Будете ли вы индексировать столбец?

Индексирование позволяет ускорить выборку данных. Эта тема рассматривается детально в главе 4, "Оптимизация запросов". Для индексирования выбираются столбцы, которые указываются в предложении WHERE.

При создании как простого, так и сложного индекса следует помнить о том, что возможно наличие самых различных ограничений, наложенных на выбранные типы. Например, в ранних версиях (до версии 3.23.2) проиндексированные столбцы не могут быть объявлены как NOT NULL, кроме того, такие столбцы не могут иметь тип BLOB или TEXT. Эти ограничения сняты в версии 3.23.2, но если нет возможности перейти на новую версию, придется работать с учетом этих ограничений. Однако есть несколько способов обойти эти ограничения.

- Данные типа BLOB и TEXT не индексируются, но если длина строки не превышает 255 символов, можно воспользоваться типом VARCHAR. В качестве заменителя типа BLOB можно воспользоваться типом VARCHAR(255) BINARY, а в качестве заменителя типа TEXT – VARCHAR(255).

Можно принять какое-нибудь значение в качестве пустого значения. Например, для столбца типа DATE в качестве такового можно рассматривать значение '0000-00-00'. В столбце строкового типа в качестве пустого значения можно принять пустую строку. Для цифрового типа, содержащего неотрицательные значения, значением, эмулирующим пустое значение, может служить значение -1. (Конечно, в таком случае столбец не может быть объявлен как UNSIGNED.)

Как учесть внутреннюю зависимость при выборе типа столбца

Нужно заметить, что все аспекты, принимаемые во внимание при выборе типа данных, нельзя рассматривать независимо друг от друга. Так, например, диапазон значений цифровых данных определенного типа зависит от размера памяти, требующейся для хранения данных этого типа. Увеличение диапазона значений влечет за собой увеличение объемов памяти. А увеличивая потребление памяти, мы влияем на производительность. В качестве другого примера можно упомянуть проблемы, которые повлечет за собой указание параметра AUTO_INCREMENT при объявлении последовательных номеров. Только такой выбор будет иметь несколько последствий, затрагивающих тип столбца, индексирование и возможности хранения пустых значений.

- Атрибут AUTO_INCREMENT можно использовать только с данными целого типа. Это сразу же ограничивает круг возможных типов типами от TINYINT до BIGINT.
- Столбцы AUTO_INCREMENT предназначаются только для генерирования последовательностей положительных значений, поэтому они должны объявляться как UNSIGNED.

- Столбцы AUTO_INCREMENT следует индексировать таким образом, чтобы текущее максимальное значение определялось быстро и без необходимости полного просмотра таблицы. Более того, с целью предотвращения повторного использования индекс должен быть уникальным. Это значит, что индекс должен быть объявлен как PRIMARY KEY или UNIQUE.
- Столбцы AUTO_INCREMENT должны объявляться как NOT NULL.

  Отсюда следует, что такого объявления строки будет явно недостаточно:

  mycol *произвольный_тип* AUTO_INCREMENT

  Характеристики столбца необходимо детализировать следующим образом:

  mycol *целый_тип* AUTO_INCREMENT NOT NULL PRIMARY KEY

  или так:

  mycol *целый_тип* AUTO_INCREMENT NOT NULL, UNIQUE (mycol)

Вычисление выражений и преобразование типов

Синтаксис СУБД MySQL позволяет создавать выражения, содержащие константы, вызовы функций и ссылки на столбцы таблиц. Эти значения могут указываться в комбинированном виде с использованием арифметических и логических операторов. Они могут объединяться в группы с помощью скобок. Чаще всего выражения появляются в перечне выбираемых столбцов и в предложениях WHERE оператора SELECT:

```
SELECT
  CONCAT(last_name, ', ', first_name),
  (TO_DAYS(death) - TO_DAYS(birth)) - IF(RIGHT(death,5) <
RIGHT(birth,5),1,0))
FROM president WHERE
  birth > '1900-1-1' AND DEATH IS NOT NULL;
```

В этом запросе выражения присутствуют как в перечне выбираемых столбцов, так и в предложении WHERE. Выражения могут указываться также в предложениях WHERE в операторах DELETE и UPDATE, предложениях VALUES () в операторах INSERT и т.д.

Найдя выражение, СУБД MySQL вычисляет его. Например, выражение (4*3) / (4-2) возвращает значение 2. Вычисление выражения может потребовать операции преобразования типов. Например, при использовании числа в контексте, требующем даты, MySQL преобразует число 960821 в дату '1996-08-21'.

Этот раздел посвящен правилам написания выражений и преобразования типов, которые создаются при вычислении выражений, содержащих данные разного типа. Объем этого раздела не позволяет привести примеры применения всех функций. Их можно найти в приложении B, "Операторы и функции".

Правила написания выражений

Выражение может быть представлено одной константой.

0	Цифровая константа
"abc"	Символическая константа

В выражениях может присутствовать вызов функции. Одни функции требуют указания аргументов (значения, ограниченные скобками), другие – нет. Несколько аргументов разделяется запятыми. Синтаксис описания функций в СУБД MySQL позволяет включать пробелы до и после аргументов, но пробел между именем функции и открывающейся скобкой недопустим.[5]

`NOW()`	Функция без аргументов
`STRCMP("abc","def")`	Функция с двумя аргументами
`STRCMP("abc", "def" )`	Пробелы до и после аргументов допустимы
`STRCMP ("abc","def")`	Пробел после имени функции недопустим

Дело в том, что имя функции с последующим пробелом будет воспринято интерпретатором как имя столбца. (Имена функций не являются зарезервированными словами и могут использоваться для именования столбцов.) Интерпретатор реагирует на эту ситуацию сообщением об ошибке.

Выражения могут содержать значения из столбцов таблицы. В простейшем случае достаточно указать имя столбца. В каждом из приведенных примеров указана только одна таблица, поэтому ссылки на столбцы будут однозначны:

```
SELECT last_name, first_name FROM president;
SELECT last_name, first_name FROM member;
```

Если не совсем понятно, какая из таблиц будет использоваться, имени столбца должно предшествовать имя таблицы, отделенное точкой. Если неизвестно, из какой базы данных используется таблица, можно указать и имя базы данных:

```
SELECT
  president.last_name, president.first_name,
  member.last_name, member.first_name
FROM president, member
WHERE president.last_name = member.last_name;
SELECT samp_db.student.name FROM samp_db.student;
```

Наконец, можно комбинировать все эти типы значений для создания более сложных форм выражений.

Типы операторов

СУБД MySQL воспринимает несколько типов операторов, используемых в выражениях. Арифметические операторы, перечисленные в табл. 2.16, включают операторы сложения, вычитания, умножения и деления, а также оператор вычисления модуля. Арифметические вычисления `'+'`, `'-'` и `'*'` выполняются с использованием целых значений типа `BIGINT` (64 бит). Это же можно сказать об операциях деления и модуля, когда по контексту ожидается целочисленный результат. Нужно помнить, что при вычислениях, в которых задействованы большие значения, такие, что результат может выйти за пределы 64-битового диапазона, могут быть получены непредсказуемые результаты.

---

[5] В действительности в MySQL можно задать разрешение указания имен после имен функций. Для этого достаточно запустить сервер с параметрами `--ansi` или `--sql-mode=IGNORE_SPACE`. Однако после этого названия функций рассматриваются как резервные слова.

Таблица 2.16. Арифметические операторы

Оператор	Синтаксис	Значение
+	a + b	Суммирование; сумма операндов
-	a - b	Вычитание; разность операндов
-	- a	Унарный минус; операнд с обратным знаком
*	a * b	Умножение; произведение операндов
/	a / b	Деление; частное от деления операндов
%	a % b	Модуль; остаток от деления операндов

Логические операторы, представленные в табл. 2.16, определяют, является выражение "истинным" (не равно нулю) или "ложным" (равно нулю). СУБД MySQL поддерживает операторы в стиле языка C — `'&&'`, `'||'` и `'!'` — как альтернативную форму традиционным AND, OR и NOT. Обратите особое внимание на оператор `'||'`. Стандарт ANSI языка SQL определяет оператор `'||'` как оператор конкатенации строк, но в СУБД MySQL это оператор логического "ИЛИ".[6]

Вы очень удивитесь, если будете ожидать от оператора, приведенного ниже, конкатенации строк, а в результате получите 0:

```
SELECT "abc" || "def"                                    -> 0
```

Перед выполнением этой операции строки `"abc"` и `"def"` преобразуются в целые числа, а результат выполнения операции дает 0. Для выполнения конкатенации строк в MySQL используется функция `CONCAT("abc","def")`.

Таблица 2.17. Логические операторы

Оператор	Синтаксис	Значение
AND, &&	a AND b, a && b	Логическое пересечение; дает значение "истина", если оба операнда имеют значение "истина"
OR, ||	a OR b, a || b	Логическое объединение; дает значение "истина", если один из операндов имеет значение "истина"
XOR	a XOR b	Логическое исключающее ИЛИ; дает значение "истина", если один из операндов имеет значение "истина"
NOT, !	NOT a, !a	Логическое отрицание; дает значение "истина", если операнд имеет значение "ложь"

Побитовые операции, представленные в табл. 2.18, предназначены для выполнения операций побитового пересечения, объединения или исключающего "ИЛИ", в которых каждый бит результата рассматривается как логическое "И" или "ИЛИ" соответствующих битов операндов. (Операции исключающего "ИЛИ" (XOR и ^) появились только в СУБД MySQL 4.0.2.) Есть операции побитового сдвига влево и вправо. Побитовые операции выполняются с применением целых значений типа BIGINT (64 бит).

[6] Для того чтобы добиться поведения в соответствии с определением операции || данным ANSI, запустите сервер с параметрами `--ansi` или `--sql-mode=PIPES_AS_CONCAT`.

Таблица 2.18. Побитовые операторы

Оператор	Синтаксис	Значение
&	a & b	Побитовое пересечение; каждый результирующий бит установлен, если установлен соответствующий бит в обоих операндах
|	a | b	Побитовое объединение; каждый результирующий бит результата установлен, если установлен соответствующий бит в одном из операндов
^	a ^ b	Побитовое исключающее "ИЛИ"; каждый результирующий бит установлен, если точно установлен соответствующий бит в операндах
<<	a << b	Побитовый сдвиг влево значения из операнда a на b битов
>>	a >> b	Побитовый сдвиг вправо значения из операнда a на b битов

Операторы сравнения (табл. 2.19) включают операторы сравнения относительных значений и сортировки чисел и строк, операторы сравнения по шаблону и сравнения с пустыми значениями. Оператор `'<=>'` применяется только в СУБД MySQL. Этот оператор был впервые введен в версии 3.23.

В MySQL 3.23 появился оператор `BINARY`, позволяющий преобразовать простую строку в двоичную. Это сделает такую строку чувствительной к регистру при выполнении операций сравнения. Обратите внимание на то, что если первый из приведенных ниже примеров не чувствителен к регистру, то остальные два уже будут чувствительны к регистру:

```
'abc' = 'Abc'                    -> 1
BINARY 'abc' = 'Abc'             -> 0
'abc' = BINARY 'Abc'             -> 0
```

Таблица 2.19. Операторы сравнения

Оператор	Синтаксис	Значение
=	a = b	"Истина", если операнды равны
<=>	a <=> b	"Истина", если операнды равны (даже если это пустые значения)
!=, <>	a != b, a <> b	"Истина", если операнды не равны
<	a < b	"Истина", если a меньше b
<=	a <= b	"Истина", если a меньше или равно b
>=	a >= b	"Истина", если a больше или равно b
>	a > b	"Истина", если a больше b
IN	a IN (b1, b2, ...)	"Истина", если a равно одному из b1, b2 ...
BETWEEN	a BETWEEN b AND c	"Истина", если a лежит в диапазоне b и c включительно
NOT BETWEEN	a NOT BETWEEN b AND c	"Истина", если a не лежит в диапазоне b и c включительно
LIKE	a LIKE b	"Истина", если a совпадает с шаблоном b
NOT LIKE	a NOT LIKE b	"Истина", если a не совпадает с шаблоном b

Окончание табл. 2.19

Оператор	Синтаксис	Значение
REGEXP	a REGEXP b	"Истина", если a совпадает с регулярным выражением b
NOT REGEXP	a NOT REGEXP b	"Истина", если a не совпадает с регулярным выражением b
IS NULL	a IS NULL	"Истина", если операнд имеет пустое значение
IS NOT NULL	a IS NOT NULL	"Истина", если операнд не имеет пустого значения

Оператора, альтернативного оператору BINARY (NOT BINARY), не существует. И если столбец предполагается использовать как в контекстах, чувствительных к регистру, так и наоборот, задайте тип столбца, который не будет реагировать на регистр, чтобы для сравнений, где необходимо иметь чувствительность к регистру, воспользоваться оператором BINARY. С другой стороны, сравнивая данные, которые чувствительны к регистру, их необходимо приводить к одному и тому же регистру с помощью оператора UPPER() или LOWER():

```
UPPER(col_name) < UPPER('Smith')
LOWER(col_name) < LOWER('Smith')
```

При выполнении операций сравнения строк, не чувствительных к регистру, очень вероятна ситуация, когда несколько символов, в зависимости от кодировки, будут рассматриваться как эквивалентные. Например, символы 'Е' и 'Й' могут рассматриваться как подобные при выполнении операций сравнения и сортировки. Двоичные (чувствительные к регистру) операции сравнения производятся с учетом ASCII-значений этих символов.

Проверка по образцу позволяет производить выборку значений, не определяя точно всего значения полностью. В СУБД MySQL это делается с помощью оператора LIKE и шаблона '%' (все последовательности символов) или шаблона '_' (любой один символ). СУБД MySQL также позволяет выполнять проверку по образцу с помощью оператора REGEXP и расширенного регулярного выражения, аналогичного тому, который имеется в программах grep, sed и vi ОС UNIX. При проверке по шаблону можно воспользоваться одним из этих операторов, а не '='. Для того чтобы инвертировать смысл проверки, воспользуйтесь оператором NOT LIKE или NOT REGEXP.

Два типа проверки по образцу различаются в двух важных аспектах.

- Оператор LIKE не реагирует на регистр в том случае, если по крайней мере один из операндов является двоичной строкой. Оператор REGEXP чувствителен к регистру.
- Выборка по образцу в SQL возможна тогда и только тогда, когда есть полное совпадение. Выборка по регулярным выражениям возможна при совпадении шаблона где-либо в строке.

Шаблоны, работающие с оператором LIKE, могут содержать символы '%' и '_'. Например, по шаблону 'Frank%' будет найдена любая строка, начинающаяся с "Frank":

```
'Franklin' LIKE 'Frank%'                    -> 1
'Frankfurter' LIKE 'Frank%'                 -> 1
```

Символ '%' найдет любую последовательность символов, включающую пустую строку. Таким образом, "Frank%" дает совпадение "Frank":

```
'Frank' LIKE 'Frank%'                       -> 1
```

Это означает, что по символу '%' система найдет любую строку, включая и пустую. Однако символ '%' не нейдет пустого значения (NULL). Любой поиск по шаблону с операндом, имеющим пустое значение (NULL), даст сбой:

```
'Frank' LIKE NULL                           -> NULL
NULL LIKE 'Frank%'                          -> NULL
```

В СУБД MySQL оператор LIKE не реагирует на регистр, если ни один из операндов не является двоичной строкой. Таким образом, строка 'Frank%' при обычном сравнении найдет обе строки, 'Frankly' и 'frankly', но при двоичном сравнении будет найдена только первая из них:

```
'Frankly' LIKE 'Frank%'                     -> 1
'frankly' LIKE 'Frank%'                     -> 1
BINARY 'Frankly' LIKE 'Frank%'              -> 1
BINARY 'frankly' LIKE 'Frank%'              -> 0
```

В этом заключается различие оператора LIKE в СУБД MySQL от оператора LIKE в стандарте языка SQL, принятом Американским национальным институтом стандартов (ANSI). В соответствии с этим стандартом оператор LIKE реагирует на регистр.

Шаблон может быть указан в любом месте образца. Так, по образцу '%bert' будут найдены значения 'Englebert', 'Bert' и 'Albert'. А по образцу '%bert%' будут найдены все эти значения, а также строки 'Berthold', 'Bertram' и 'Alberta'.

Другой шаблон, '_', выбирает строки, где на данной позиции может стоять любой символ. Образец '___' выберет любую строку, состоящую из любых трех символов. Образец 'c_t' выберет строки наподобие 'cat', 'cot', 'cut' и даже 'c_t' (символ '_' совпадает с самим собой).

Для отключения специальной функциональности символов-шаблонов '%' и '_' их необходимо предварить обратной косой чертой ('\%' или '_'):

```
'abc' LIKE 'a%e'                            -> 1
'abc' LIKE 'a\%c'                           -> 0
'a%c' LIKE 'a\%c'                           -> 1
'abc' LIKE 'a_c'                            -> 1
'abc' LIKE 'a\_c'                           -> 0
'a_c' LIKE 'a\_c'                           -> 1
```

Другая форма поиска по образцу в MySQL позволяет использовать регулярные выражения. Для этого используется оператор REGEXP (RLIKE синоним REGEXP). Шаблоны, употребляемые наиболее часто:

- '.' — шаблон любого символа:

```
'abc' REGEXP 'a.c'                                -> 1
```

- '[...]' — шаблон любого символа, заключенного в квадратные скобки:

```
'e' REQEXP '[aeiou]'                              -> 1
'f' REGEXP '[aeiou]'                              -> 0
```

Символ '*' означает "любое число символов, совпадающих с предшествующим символом". Это значит, что выражение 'x*' найдет любое количество символов 'x':

```
'abcdef' REGEXP 'a.*f'                    -> 1
'abc' REGEXP '[0-9]*abc'                  -> 1
'abc' REGEXP '[0-9][0-9]*'                -> 0
```

"Любое количество" включает нулевые экземпляры, поэтому второе выражение в результате дает "истину".

Образцы '^pat' и 'pat$' закрепляют шаблон таким образом, что комбинация символов pat будет найдена только в том случае, если она встречается в начале или в конце строки, а '^pat$' означает, что комбинация символов pat будет найдена только в том случае, если совпадает со всей строкой:

```
'abc' REGEXP 'b'                          -> 1
'abc' REGEXP '^b'                         -> 0
'abc' REGEXP 'b$'                         -> 0
'abc' REGEXP '^abc$'                      -> 1
'abcd' REGEXP '^abc$'                     -> 0
```

Оператор REGEXP позволяет выбирать шаблон из табличного столбца. Такая процедура работает медленней, чем при сравнении констант. Шаблон проверяется и запоминается во внутреннем представлении при каждом новом изменении значения.

Кроме того, регулярные выражения имеют и другие специальные символы. С ними можно познакомиться в приложении В, "Операторы и функции".

Приоритет операций

При вычислении выражений интерпретатор СУБД MySQL определяет порядок действий. Одни операции имеют более высокий приоритет и выполняются раньше других. Например, операции умножения и деления имеют самый высокий приоритет, поэтому они будут вычисляться раньше других. Вот два выражения, которые эквивалентны потому, что операции '*' и '/' вычисляются до операций '+' и '-'. Результат этих выражений тоже будет идентичен:

```
1+2*3-4/5                                 -> 6.2
1 + 6 - .8                                -> 6.2
```

Приоритет операторов приведен в нижеследующем списке. Он начинается операторами, обладающими высоким приоритетом, и заканчивается операторами, имеющими низкий приоритет. Операторы, представленные в одной строке, имеют равный приоритет. Операторы с высоким приоритетом вычисляются раньше операторов с низким приоритетом.

```
BINARY COLLATE
NOT !
- (унарный минус) ~ (унарное побитовое отрицание)
*   /   %
 +   -
<< >>
&
|
< <= = <=> != <> >= > IN IS LIKE REGEXP RLIKE
BETWEEN CASE WHEN THEN ELSE
AND &&
OR ||
:=
```

Приоритет операций меняется с помощью скобок, что меняет порядок выполнения операции:

```
1+2*3-4/5                                   -> 6.2
(1 + 2) * (3 - 4) / 5                       -> -0.6
```

Пустые значения в выражениях

С пустыми значениями нужно работать крайне осторожно. Результат может быть далек от ожидаемого. Вот основные принципы, которые позволят вам избежать неприятностей в дальнейшем.

Пустое значение (NULL) в качестве операнда любого арифметического или битового оператора даст в результате NULL:

```
1 + NULL                                    -> NULL
1 | NULL                                    -> NULL
```

При логических операциях результат будет равен пустому значению (NULL), хотя результат при этом является определимым:[7]

```
1 AND NULL                                  -> NULL
1 OR NULL                                   -> 1
0 AND NULL                                  -> 0
0 OR NULL                                   -> NULL
```

Пустое значение (NULL) в качестве операнда в операторах сравнения в результате дает пустое значение (NULL). Исключением являются операторы `'<=>'`, `IS NULL` и `IS NOT NULL`, которые специально рассчитаны на работу с пустыми значениями (NULL):

```
1 = NULL                                    -> NULL
NULL = NULL                                 -> NULL
1 <=> NULL                                  -> 0
NULL <=> NULL                               -> 1
NULL LIKE '%'                               -> NULL
NULL REGEXP '.*'                            -> NULL
NULL <=> NULL                               -> 1
1 IS NULL                                   -> 0
NULL IS NULL                                -> 1
```

Все функции возвращают пустое значение (NULL) при наличии значения аргумента (NULL). Исключением являются функции, которые были специально разработаны для работы с пустыми аргументами. Примером такой функции может служить функция `IFNULL()`, в зависимости от значения аргумента возвращающая соответственно "истину" или "ложь". А функция `STRCMP()` работает только с непустыми данными, и, если среди аргументов попадается пустое значение, она возвратит только пустое значение, а не значение "истина" или "ложь".

При выполнении операций сортировки пустые значения (NULL) группируются вместе. При этом они размещаются при сортировке до или после непустых значений, что зависит от версии используемого сервера СУБД MySQL (см. раздел "Сортировка результатов запроса" в главе 1, "Знакомство с СУБД MySQL и SQL").

---

[7] В версиях до 3.23.9 пустое значение (NULL) в качестве операнда логического оператора рассматривается как "ложное" значение; такое поведение можно считать ошибкой в программе.

Преобразование типов

СУБД MySQL автоматически преобразовывает типы при выполнении операций. Преобразование типов данных происходит в следующих ситуациях:

- преобразование операндов в тип, подходящий для вычисления;
- преобразование типа аргумента функции в тип, необходимый для работы функции;
- преобразование значения для присвоения табличному столбцу, тип которого отличается от типа значения.

Для следующего выражения потребуется преобразование типа. Оно состоит из оператора `'+'` и двух операндов `1` и `'2'`:

```
1 + '2'
```

Здесь операнды имеют различные типы данных (число и строка). СУБД MySQL преобразует один из них, чтобы привести операнды к одному типу. В данном случае мы имеем дело с арифметической операцией (об этом свидетельствует знак `'+'`), поэтому MySQL преобразует строку `'2'` в число 2. Результатом вычисления выражения будет 3. Вот другой пример. Функция `CONCAT()` объединяет строки, следовательно, она возвращает значение строкового типа. Таким образом, аргументы должны интерпретироваться как строки, независимо от их первоначального типа. В данном примере в качестве аргументов функция получает перечень чисел. Функция преобразует их в строки и возвращает конкатенацию этих строк:

```
CONCAT(1,2,3)                    -> '123'
```

Если вызов функции `CONCAT()` является частью большего выражения, потребуется дальнейшее преобразование типов. Рассмотрим следующее выражение:

```
REPEAT('X',CONCAT(1,2,3)/10)     -> 'XXXXXXXXXXXX'
```

Функция `CONCAT(1,2,3)` возвращает строку `'123'`. Выражение `'123'/10` преобразуется в выражение `123/10`, поскольку деление — это арифметическая операция. В контексте операций с вещественными числами результатом этого выражения будет `12.3`, но функция `REPEAT()` требует в качестве аргумента целое значение, поэтому она получит в качестве аргумента число 12. Это значит, что результатом этого выражения будет оператор `REPEAT('X',12)`, возвращающий 12 символов `'X'`.

Главный принцип преобразования типов заключается в том, что MySQL сначала пытается преобразовать данные, а не выдает сообщение об ошибке. В зависимости от контекста MySQL преобразует значения трех основных категорий (числа, строки или календарные данные) в значения любой другой категории. Однако значения не всегда могут быть преобразованы из одного типа в другой. Операция преобразования может дать сбой, если она неосуществима. Преобразование строк вида `'abc'`, которые никак не похожи на числа, вернет числовой результат, равный 0. Преобразование данных, которые не похожи на календарные данные, в календарный тип вернет 0. Например, преобразование строки `'abc'` в дату вернет "нулевую" дату `'0000-00-00'`, но любое значение может быть рассмотрено как строка. Это значит, что при преобразовании значения в строку никогда не должно возникать проблем.

MySQL выполняет и менее серьезные операции преобразования. Если в целочисленном контексте используется вещественное значение, значение будет преобразовано (округлено). Преобразование в обратном направлении тоже работает. Использование целого числа в качестве вещественного числа не будет ошибкой.

Шестнадцатеричные константы рассматриваются как строки в том случае, если контекст однозначно не свидетельствует о том, что это число. В шестнадцатеричном контексте каждая пара шестнадцатеричных цифр рассматривается в качестве символа, а результат используется как строка. Вот как это происходит:

```
0x61                                    -> 'a'
0x61 + 0                                -> 97
X'61'                                   -> 'a'
X'61' + 0                               -> 97
CONCAT(0x61)                            -> 'a'
CONCAT(0x61 +0)                         -> '97'
CONCAT(X'61')                           -> 'a'
CONCAT(X'61' +0)                        -> '97'
```

Обработка операций сравнения шестнадцатеричных констант зависит от версии сервера. Начиная с версии 3.23.22 шестнадцатеричные константы обрабатываются как числа:

```
0x0a = '\n'                             -> 0
0xaaab < 0xab                           -> 0
0xaaab > 0xab                           -> 1
0x0a = 10                               -> 1
```

До версии 3.23.22 шестнадцатеричные константы обрабатываются как двоичные строки, поэтому некоторые из вышеуказанных операций дадут совершенно обратные результаты:

```
0x0a = '\n'                             -> 1
0xaaab < 0xab                           -> 1
0xaaab > 0xab                           -> 0
0x0a = 10                               -> 1
```

Есть операторы, которые инициируют преобразование в тип, необходимый оператору независимо от типа операндов. Примером таких операторов могут служить арифметические операторы. В качестве операндов они ожидают числа, и операнды преобразуются соответствующим образом:

```
3+4                                     -> 7
'3' +4                                  -> 7
'3' + '4'                               -> 7
```

Для того чтобы символьная строка была преобразована в число, она должна содержать числа в начале строки. Если таковых в начале строки найдено не будет, то преобразование такой строки вернет значение 0 (см. пятую строку):

```
'1973-2-4' +0                           -> 1973
'12:14:01' +0                           -> 12
'23-skidoo' +0                          -> 23
'-23-skidoo' +0                         -> -23
'carbon-14' +0                          -> 0
```

Начиная с версии 3.23 правила преобразования строк в числа несколько меняются. До этой версии строки преобразовывались в целые числа с округлением. Начиная с версии 3.23 строки стали преобразовываться в вещественный формат:

```
'-428.9' +0                          -> -429 (MySQL < 3.23)
'-428.9' +0                          -> -428.9 (MySQL 3.23)
```

Еще строже правила преобразования при выполнении логических и побитовых операций. Операнды должны быть не просто числами, а только целыми числами. Это значит, что такое число, как `0.3`, не дает значение "истина", несмотря на то, что оно не равно нулю. Это происходит потому, что при преобразовании в целое число оно возвращает 0. В других выражениях операторы тоже не могут дать значение "истина", если их значение не превышает 1.

```
0.3 OR .04                                   -> 0
1.3 OR .04                                   -> 1
0.3 AND .04                                  -> 0
1.3 AND .04                                  -> 0
1.3 AND 1.04                                 -> 1
```

Преобразование такого же типа происходит в функции `IF()`, первый оператор которой должен иметь целочисленный тип. Это значит, что числа, которые округляются до нуля, будут восприняты как "ложь":

```
IF(1.3, 'non-zero','zero')                   -> 'non-zero'
IF(0.3, 'non-zero','zero')                   -> 'zero'
IF(-0.3, 'non-zero','zero')                  -> 'zero'
IF(-1.3, 'non-zero','zero')                  -> 'non-zero'
```

При обработке вещественных значений лучше воспользоваться явным сравнением:

```
IF(0.3>0, 'non-zero', 'zero')                -> 'non-zero'
```

Операторы сравнения по образцу работают со строками. Но операторы сравнения по образцу СУБД MySQL могут работать с числами, которые они преобразуют в строки при поиске совпадений!

```
12345 LIKE '1%'                              -> 1
12345 REGEXP '1.*5'                          -> 1
```

Операторы сравнения значений реагируют на контекст (`'<'`, `'<='`, `'='` и т.д.), т.е. они производят сравнение в соответствии с типом их операндов. Так, выражение, представленное внизу, сравнивает операторы как числа, поскольку оба оператора являются числами:

```
2 < 11                                       -> 1
```

Вот выражение, в котором происходит лексографическое сравнение строк, потому что оба оператора являются строками:

```
'2' < '11'                                   -> 0
```

В следующих операциях сравнения разные типы данных сравниваются таким образом, что СУБД MySQL сравнивает их как числа. В результате оба выражения возвращают результат "истина":

```
'2' < 11                                     -> 1
2 < '11'                                     -> 1
```

При сравнениях СУБД MySQL преобразует типы данных в соответствии со следующими правилами.

- Операции сравнения, отличные от операции `'<=>'`, где один из операндов — пустое значение (NULL), возвращают NULL. (Операция `'<=>'` анало-

гична операции '=', но сравнение NULL <=> NULL возвращает значение "истина".)

- Когда оба операнда являются строками, они сравниваются лексографическим образом как строки. Двоичные строки сравниваются в побитовом режиме с использованием числового значения каждого типа. Сравнение строк производится на основании анализа кодировок строк. При различных кодировках сравниваемых строк (что возможно в СУБД MySQL 4.1) сравнение не даст какого-либо значащего результата. Сравнение между двоичными и недвоичными строками производится как сравнение двоичных строк.
- Когда оба операнда являются целыми числами, они сравниваются как целые числа.
- Начиная с версии 3.23.22 шестнадцатеричные константы сравниваются как числа. До этого они сравнивались как двоичные строки.
- Если один из операндов имеет значение типа TIMESTAMP или DATETIME, а другой операнд является константой, операнды сравниваются как значения TIMESTAMP. Это сделано для оптимизации работы ODBC-приложений.
- В любом другом случае операнды сравниваются в вещественном формате. Это правило включает случай сравнения строки и числа. Строка преобразуется в значение 0, если в ней нет чисел. Например, '14.3' преобразуется в 14.3, а 'L4.3' — в 0.

Правила интерпретации календарных данных

MySQL позволяет свободно преобразовывать строковые данные и числа в календарные типы данных и наоборот. Календарные данные преобразуются в числа в числовом контексте, числа преобразуются в календарные данные в контексте даты и времени. Преобразование данных в календарный тип происходит в момент присвоения значения столбцу с календарным типом данных. При выполнении операций сравнения основное правило гласит: "календарные данные сравниваются как строки".

Для таблицы mytbl, имеющей в своем составе столбец date_col типа DATE, следующие операторы будут аналогичны:

```
INSERT INTO mytbl SET date_col = '2004-04-13'
INSERT INTO mytbl SET date_col = '20040413'
INSERT INTO mytbl SET date_col = 20040413
```

Аргументы функции TO_DAYS() интерпретируется как одно и то же значение во всех трех выражениях:

```
TO_DAYS('2004-04-10')                    -> 732046
TO_DAYS('20040410')                      -> 732046
TO_DAYS(20040410)                        -> 732046
```

Тестирование и запуск преобразования типов

Посмотрим, как происходит преобразование типов. Для этого сделаем запрос SELECT из программы mysql:

```
mysql> SELECT 0x41, 0x41 + 0;
+------+-----------+
```

```
| 0x41 | 0x41 + 0 |
+------+----------+
| A    |       65 |
+------+----------+
```

Тестирование выражений необходимо при программировании операций, производящих изменения в строках (оператор DELETE или UPDATE). Это нужно для гарантии удаления или модификации соответствующих данных. Наилучшим способом проверки работы таких операторов может быть предварительный запуск оператора SELECT с аналогичным предложением WHERE. Предположим, что таблица mytbl имеет столбец типа CHAR char_col, содержащий следующие значения:

```
'abc'
'def'
'00'
'ghi'
'jkl'
'00'
'mno'
```

Таков будет результат запроса:

```
DELETE FROM mytbl WHERE char_col = 00;
```

Этот запрос создавался для удаления двух строк, содержащих значение '00'. В действительности, и это может стать неприятным сюрпризом, он удалит все строки таблицы. Это можно объяснить последствием работы правил сравнения, принятых в СУБД MySQL. Столбец char_col является столбцом строкового типа. Но 00 в этом операторе не заключено в скобки. Таким образом, он рассматривается как число. В соответствии с правилами, принятыми в СУБД MySQL, операция сравнения, в которой участвуют строка и число, выполняется как сравнение двух чисел. При выполнении запроса DELETE каждое значение столбца char_col преобразуется в число и сравнивается со значением 0. Значение '00' будет преобразовано в 0, но ведь то же самое произойдет и со всеми другими строками, которые не похожи на числа. Поэтому предложение WHERE будет давать значение "истина" для всех строк, и оператор DELETE очистит всю таблицу. Очевидно, что в этом и подобном ему случаях (да и во многих других) будет благоразумнее предварительно проверять работу предложения WHERE с помощью оператора SELECT. После чего можно спокойно запускать его в операторе DELETE без риска удалить по ошибке "не те" строки:

```
mysql> SELECT char_col FROM mytbl WHERE char_col = 00;
+----------+
| char_col |
+----------+
| 'abc'    |
| 'def'    |
| '00'     |
| 'ghi'    |
| 'jkl'    |
| '00'     |
| 'mno'    |
+----------+
```

Когда нет полной уверенности в том, каким образом будет использоваться значение, можно принудительно воспользоваться механизмом преобразования

типов, который запускается при вычислении выражений. Это даст полную картину преобразования значения в заданный тип данных.

- Прибавьте к выражению +0 или +0.0. Это инициирует преобразование в цифровое значение:

```
0x65                                    -> 'e'
0x65 +0                                 -> 101
0x65 + 0.0                              -> 101.0
```

- Для преобразования вещественного значения в целое можно воспользоваться функцией FLOOR(), а для преобразования целого числа в вещественное прибавить 0.0:

```
FLOOR(13.3)                             -> 13
13 + 0.0                                -> 13.0
```

  При необходимости округления можно воспользоваться функцией ROUND().

- Для преобразования значения в строку воспользуйтесь функцией CONCAT():

```
14                                      -> 14
CONGAT(14)                              -> '14'
```

  Начиная с версии 4.0.2 для преобразования чисел в шестнадцатеричную строку воспользуйтесь функцией HEX():

```
HEX(255)                                -> 'FF'
HEX(65535)                              -> 'FFFF'
```

  Функцию HEX() можно использовать для преобразования строк в строки, содержащие пары шестнадцатеричных цифр, представляющих коды символов:

```
HEX('abc')                              -> '616263'
```

- Для получения ASCII-кода символа воспользуйтесь функцией ASCII():

```
'A'                                     -> 'A'
ASCII('A')                              -> 65
```

- Для обратной операции получения символа из ASCII-кода воспользуйтесь функцией CHAR():

```
CHAR(65)                                -> 'A'
```

- Для того чтобы строку или число преобразовать в календарный тип данных, можно прибегнуть к помощи функции DATE_ADD():

```
20030101                                -> 20030101
DATE_ADD(20030101, INTERVAL 0 DAY)      -> '2003-01-01'
'20030101'                              -> '20030101'
DATE_ADD('20030101', INTERVAL 0 DAY)    -> '2003-01-01'
```

- Временные данные можно преобразовывать в цифровую форму, прибавив нуль:

```
CURDATE()                               -> '2002-09-18'
CURDATE()+0                             -> 20020918
CURTIME()                               -> '12:05:41'
CURTIME()+0                             -> 120541
```

- В версии СУБД MySQL 4.1 и выше существует возможность преобразовывать кодировку строки с помощью функции CONVERT() или с указанием идентификатора кодировки:

```
'abc'                                   -> abc
CONVERT('abc' USING ucs2)               -> '\0a\0b\0c'
CHARSET('abc')                          -> 'latin1'
CHARSET(CONVERT('abc' USING ucs2))      -> 'ucs2'
CHARSET(_ucs2 'abc')                    -> 'ucs2'
```

Преобразование ошибочных значений и значений, лежащих вне диапазона

В этом случае работает принцип: "мусор на входе, мусор на выходе". Если данные не проверяются при вводе, нет никакой гарантии, что будут получены верные данные. Определим детально основные принципы обработки ошибочных значений и значений, лежащих вне допустимого диапазона.

- Для столбцов цифрового типа или типа TIME значения, находящиеся за пределом допустимых значений, "округляются" до ближайшего предельного значения диапазона, и полученное значение сохраняется.
- При присвоении значений, не попадающих в допустимый для этого типа диапазон, столбцам всех календарных типов (исключая TIME) может быть присвоено "нулевое" значение, пустое или любое другое значение (другими словами, результаты будут непредсказуемыми).
- Все строковые типы (за исключением типов ENUM и SET) при вводе слишком длинных строк усекаются до размера, заданного при объявлении столбца. Допустимые значения для типов ENUM и SET перечисляются при объявлении столбца. При попытке присвоить значение, которое отсутствует в этом списке, столбцу присваивается значение, сигнализирующее об ошибке (т.е. пустая строка, соответствующая в этом случае нулевому значению). При присвоении столбцу, объявленному как SET, отсутствующего в списке значения это значение игнорируется, и столбцу присваивается значение из оставшегося множества.
- При присвоении ошибочных календарных значений столбцам присваиваются соответствующие "нулевые" значения, принятые для данного типа (см. табл. 2.12).

Возникновение таких ситуаций при выполнении операторов ALTER TABLE, LOAD DATA, UPDATE, INSERT INTO ... SELECT и оператора INSERT в операции многострочного добавления обязательно диагностируется. В программе mysql эта информация выводится в строке состояния после выполнения запроса. При программировании запроса средствами языка программирования эту информацию можно получить по-другому. При работе с помощью интерфейса MySQL C API для получения этой информации предназначается функция mysql_info(). При работе с помощью интерфейса Perl DBI API для получения этой информации можно воспользоваться атрибутом mysql_info, задаваемым в момент подключения к базе данных. Выдается информация о количестве предупреждений.

3

Синтаксис и использование языка SQL

В этой главе...

Правила присвоения имен в СУБД MySQL
Создание, удаление и выборка баз данных
Создание, удаление, индексирование и изменение структуры таблиц
Получение информации о базах данных и таблицах
Выборка записей из нескольких таблиц
Удаление и обновление нескольких таблиц
Выполнение транзакций
Внешние ключи и ссылочная целостность
Полнотекстный поиск типа FULLTEXT
Комментарии
Возможности, которые не поддерживает СУБД MySQL

Для работы с СУБД MySQL необходимо отличное знание языка SQL. Оно необходимо не только для интерактивной работы с клиентской программой `mysql`, являющейся передатчиком операторов SQL с клиента на сервер. Знание SQL необходимо и при создании программ, работающих через интерфейс СУБД MySQL с языками программирования и базами данных. И в этом случае происходит взаимодействие с сервером и посылка запросов.

В главе 1, "Знакомство с СУБД MySQL и SQL", уже были продемонстрированы многие возможности СУБД MySQL. В этой главе мы детальнее изучим многие аспекты диалекта языка SQL, нашедшего применение в СУБД MySQL. Здесь описаны правила обращения к элементам базы данных, правила присвоения имен и ог-

раничения, накладываемые наличием чувствительности к регистру. Кроме того, в главе описаны многие операторы SQL, выполняющие следующие типы операций:

- создание и удаление баз данных, таблиц и индексов;
- получение информации о базах данных и таблицах;
- выборка данных с использованием объединений различного типа (join и union) и подчиненных операторов выборки;
- использование множественных удалений и модификаций;
- выполнение транзакций, которые позволяют рассматривать группу операторов как единый блок;
- установка отношений по внешним ключам;
- использование поискового механизма FULLTEXT.

Как показано в табл. 3.1, все операторы SQL, нашедшие применение в СУБД MySQL, можно сгруппировать по категориям. В этой таблице указаны операторы, включенные в каждую группу. В некоторых случаях может пригодиться программа, осуществляющая интерфейс с СУБД MySQL с помощью командной строки. Например, программа `mysqlshow` позволяет задавать операторы `SHOW` из командной строки. В данной главе такие соответствия подчеркиваются в соответствующих местах.

В этой таблице не представлены некоторые операторы, которые обсуждаются в других главах. Например, операторы администрирования `GRANT` и `REVOKE`, предназначенные для установки привилегий пользователей, описаны в главе 11, "Общее администрирование MySQL", и в главе 12, "Безопасность", посвященных привилегиям и их функциональной нагрузке. Синтаксис всех операторов, нашедших применение в СУБД MySQL, приводится в приложении Г, "Синтаксис SQL". Кроме того, дополнительную информацию о последних изменениях, появившихся в недавних версиях СУБД MySQL, можно найти в *Справочном руководстве по СУБД MySQL (MySQL Reference Manual)*.

Таблица 3.1. Типы операторов SQL, которые поддерживаются СУБД MySQL

Выборка, создание, удаление и изменение баз данных
`USE`
`CREATE DATABASE`
`DROP DATABASE`
`ALTER DATABASE`
Создание, удаление и изменение таблиц и индексов
`CREATE TABLE`
`DROP TABLE`
`CREATE INDEX`
`DROP INDEX`
`ALTER TABLE`
Сбор информации о таблицах и базах данных
`DESCRIBE`
`SHOW`

Окончание табл. 3.1

Выборка информации из таблиц
SELECT
UNION
Выполнение транзакций
BEGIN
COMMIT
ROLLBACK
SET AUTOCOMMIT
Изменение информации в таблицах
DELETE
INSERT
LOAD DATA
REPLACE
UPDATE
Операторы администрирования
FLUSH
GRANT
REVOKE

Последний раздел этой главы посвящен возможностям, имеющимся в остальных СУБД, но отсутствующим в СУБД MySQL. Это триггеры, хранимые процедуры и курсоры. Значит ли их отсутствие, что СУБД MySQL является "ненастоящей"? Да, есть такое мнение. Но имеет место и другое: отсутствие этих возможностей не останавливает потенциальных пользователей, и популярность СУБД MySQL растет. Наверное, это происходит потому, что такие возможности для многих приложений и не требуются. Во многих случаях эти недостатки очень просто обойти.

Необходимо еще раз подчеркнуть, что возможности СУБД MySQL постоянно расширяются. В первом издании этой книги список недостающих возможностей включал транзакции, вложенные выборки, внешние ключи и ссылочную целостность. Сейчас все эти возможности уже числятся в арсенале СУБД MySQL. Следующими на очереди стоят триггеры, хранимые процедуры и курсоры.

Правила присвоения имен в СУБД MySQL

Почти каждый оператор SQL неким образом обращается к базе данных или элементам, ее составляющим. Этот раздел освещает синтаксические правила ссылки на базы данных, таблицы, столбцы, индексы и псевдонимы. Имена рассматриваются с точки зрения чувствительности к регистру.

Ссылка на элементы баз данных

При присвоении имен элементам базы данных мы ограничены набором допустимых символов и длиной имени. Вид имени также зависит от контекста, в кото-

ром оно используется. Есть еще один фактор, который влияет на правило присвоения имен — это то, что сервер может быть запущен в различных режимах присвоения имен.

- **Допустимые символы.** Имя может состоять из набора любых алфавитно-цифровых символов, которые использует сервер, а также символов "_" (знак подчеркивания) и "$" (знак доллара). Имена могут начинаться с любого допустимого символа, включая цифры. Однако имя столбца не может состоять из одних цифр: это сделает его неотличимым от чисел. В СУБД MySQL предусмотрена уникальная возможность начинать имя с цифры. При этом необходимо быть особенно внимательным при использовании имен, содержащих символы "E" и "e". Такие имена приводят к неоднозначности. Выражение 23e + 14 может означать сложение значения столбца 23e плюс 14. А что может значить 23e+14? Это имя или это число в научной нотации?

  Псевдонимы присваиваются достаточно произвольно, но если они совпадают с ключевым словом языка SQL, то полностью состоят из цифр, содержат пробелы или любые другие специальные символы, и их необходимо брать в кавычки.

  Начиная с версии 3.23.6 имена можно заключить в обратные одинарные кавычки ("`"), что позволяет использовать в именах любые символы за исключением обратной одинарной кавычки, ASCII код 0 и ASCII код 255. Это полезно, когда имя содержит специальные символы или ключевое слово. Заключение имен в кавычки также позволяет быть именам полностью цифровыми, что не совсем корректно, если речь идет о незакавыченных именах.

  Однако для имен баз данных и имен таблиц существуют два дополнительных ограничения, даже при условии, что имя взято в кавычки. Во-первых, нельзя использовать точку, так как она является разделителем в записях типа *db_name.tbl_name* и *db_name.tbl_name.col_name*. Во-вторых, нельзя использовать разделители, используемые при написании имени пути ОС UNIX или Windows ("/" или "\"). Это запрещение на использование разделителей в именах баз данных и таблиц появилось потому, что базы данных представлены на диске каталогами, а таблицы представлены как минимум одним файлом. Следовательно, имена этих типов не могут содержать символы, недопустимые в именах каталогов и файлов. Разделитель в имени пути, принятый для ОС UNIX, не разрешен в ОС Windows (и наоборот). Это соглашение упрощает экспорт/импорт баз данных и таблиц между серверами, работающими на различных платформах. Например, предположим, что разрешено использование косой черты в именах таблиц на ОС Windows. Это сделает невозможным перенос таблицы на платформу UNIX, так как имена файлов на этой платформе не могут содержать косых.

- **Длина имени.** Имена баз данных, таблиц, столбцов и индексов имеют длину до 64 символов. Имена псевдонимов могут иметь до 256 символов.

- **Квалификаторы имен.** Для обращения к базе данных достаточно просто указать ее имя.

  ```
  USE db_name
  ```

```
SHOW TABLES FROM db_name
```

Для ссылки на таблицу есть два варианта: во-первых, дать полностью квалифицированное имя таблицы, состоящее из двух частей, разделенных точкой, – имени базы данных и имени таблицы:

```
SHOW TABLES FROM db_name.tbl_name
SELECT * FROM db_name.tbl_name
```

Во-вторых, при указании только имени таблицы обращение происходит к текущей базе данных. Таким образом, два следующих оператора идентичны:

```
SHOW TABLES FROM member
SELECT * FROM sampdb.member
```

Указание имени таблицы без имени базы данных при невыбранной базе данных приводит к ошибке, так как сервер не может определить, к какой базе данных принадлежит данная таблица.

Для ссылки на столбец есть три варианта: указание полностью квалифицированного имени столбца, указание частично квалифицированного имени столбца и указание неквалифицированного имени столбца. Полностью квалифицированное имя столбца (в нотации *db_name.tbl_name.col_name*) дает ссылку на определенный столбец из определенной таблицы, входящей в состав определенной базы данных. Частично квалифицированное имя столбца (записывается как *tbl_name.col_name*) ссылается на определенный столбец в определенной таблице. Вот два запроса, ссылающиеся на одноименные столбцы. Но из контекста предложений FROM видно, что они относятся к разным таблицам:

```
SELECT last_name, first_name FROM president
SELECT last_name, first_name FROM member
```

Обычно давать полностью квалифицированные имена нет необходимости, хотя это и не является ошибкой. Использование оператора USE делает базу данных текущей. В операторе SELECT таблица становится текущей для всех столбцов в данном операторе, если она одна указана в предложении FROM. Имена таблиц нужно квалифицировать только при наличии в операторе нескольких таблиц (когда таблицу или базу данных нельзя определить по контексту). Опишем несколько ситуаций, при которых возникает такая неоднозначность. Так происходит при запросах, ссылающихся на таблицы из нескольких баз данных. Любая таблица, не являющаяся в данный момент текущей, должна указываться нотацией *db_name.tbl_name*. Это помогает определить, из какой базы данных использовать таблицу. Аналогично, при запросах, выбирающих столбцы из нескольких таблиц. Это запросы, в которых столбец с одним и тем же именем присутствует в нескольких таблицах.

- **Режим запуска сервера.** При запуске сервера с параметрами `--ansi` или `--sql-mode=ANSI_QUOTES` имена можно заключать в обычные двойные кавычки (при этом обратные одинарные кавычки по-прежнему можно использовать).

Чувствительность к регистру в операторах SQL

Правила, которым подчиняется синтаксис операторов SQL с точки зрения чувствительности к регистру, зависят от части оператора и от операционной системы, под управлением которой работает данный сервер.

- **Ключевые слова и имена функций.** Ключевые слова и имена функций не зависят от регистра. Они могут вводиться в любом регистре. Вот абсолютно идентичные операторы:

```
SELECT NOW()
select now()
SeLeCt nOw()
```

- **Имена баз данных и таблиц.** Имена баз данных и таблиц подчиняются правилам именования каталогов и файлов в операционной системе, работающей на сервере. Вследствие этого чувствительность к регистру имен баз данных и таблиц подчиняется правилу, предусмотренному для имен файлов и принятому в операционной системе. Для сервера, работающего под управлением ОС UNIX, имена баз данных и таблиц зависят от регистра, а в ОС Windows имена файлов не зависят от регистра, поэтому имена баз данных и таблиц поведут себя аналогичным образом. (Исключение составляют имена в файловых системах HFS+ на платформе ОС Mac OS X, не чувствительные к регистру.)

  Об этом нужно помнить, когда ставится задача переноса базы данных из ОС UNIX в ОС Windows. Если в процессе работы с базой данных были созданы две таблицы, например abc и ABC, то после переноса базы данных в среду ОС Windows они будут неразличимы. Как этого избежать? Достаточно выработать для себя правило, регламентирующее создание всех баз и таблиц только в одном регистре. Только тогда при переносе данных из одной операционной среды в другую можно не беспокоиться о неоднозначности данных.

- **Имена столбцов и индексов.** В СУБД MySQL имена столбцов и индексов не чувствительны к регистру. Следующие операторы будут эквивалентны:

```
SELECT name FROM student
SELECT NAME FROM student
SELECT nAmE FROM student
```

- **Имена псевдонимов.** В СУБД MySQL псевдонимы чувствительны к регистру. Псевдоним можно объявлять в любом регистре (верхнем или нижнем, в обоих сразу), но при дальнейших обращениях к нему необходимо четко соблюдать первоначальный регистр.

Опыт показывает, что независимо от того, являются ли база данных, таблица или псевдоним чувствительными к регистру, их необходимо указывать в одном и том же регистре на протяжении одного и того же запроса. Это правило не относится к ключевым словам языка SQL, именам функций, столбцов и индексов, которые можно вводить в разных регистрах. Естественно, необходимо стремиться делать запросы как можно более читабельными и вырабатывать определенный стиль ввода элементов запроса.

Создание, удаление и выборка баз данных

В СУБД MySQL имеется несколько операторов манипулирования базами данных: оператор USE — для объявления текущей базы данных, CREATE DATABASE — для создания баз данных, DROP DATABASE — для удаления баз данных и ALTER DATABASE — для модификации глобальных характеристик баз данных.

Выбор баз данных

Оператор USE предназначен для объявления базы данных текущей базой данных:

```
USE db_name
```

Для осуществления такой операции необходимо иметь право доступа к базе данных. В противном случае работать с ней будет невозможно. Эта операция совсем не обязательна, так как к базе данных можно обращаться в формате *db_name.tbl_name*. Но очевидно, что удобнее ссылаться на таблицы, не прибегая при этом к помощи полного квалификатора. Например, для выборки содержимого таблицы president из базы данных sampdb без предварительного объявления этой базы данных текущей необходимо сделать следующее:

```
SELECT * FROM sampdb.president;
```

Однако, следует признать, что значительно удобней ссылаться на таблицы, не указывая имени базы данных.

Объявление базы данных базой данных по умолчанию совсем не означает, что она останется таковой на протяжении всего сеанса соединения с сервером: оператор USE можно задавать сколько угодно раз, тем самым переключаясь между текущими базами данных. Кроме того, выбор одной базы данных в качестве текущей совсем не означает, что при этом нельзя манипулировать данными из таблиц, принадлежащих другой базе данных, указывая при этом полный квалификатор.

После отключения от сервера не остается ни одного свидетельства о том, какая база данных была текущей в последний раз. Это значит, что при последующем подключении к серверу он не "вспомнит", какая база данных была текущей во время предыдущего сеанса. Да в этом и нет никакого смысла! СУБД MySQL является многопотоковой СУБД и может обрабатывать одновременно несколько подключений пользователей, которые подключаются и отключаются в совершенно произвольном порядке. Поэтому выражение наподобие "база данных, которая была текущей во время последнего сеанса" лишено всякого смысла.

Создание баз данных

Создание базы данных — очень простая операция. Для ее выполнения достаточно ввести оператор CREATE DATABASE:

```
CREATE DATABASE db_name
```

Ограничения здесь заключаются в том, что имя должно содержать только допустимые символы; база данных с таким именем существовать не должна, и пользователь должен обладать правами на создание баз данных.

Удаление баз данных

Удаление базы данных — такая же простая операция, как и создание базы данных, но и на ее выполнение необходимо иметь соответствующее право.

```
DROP DATABASE db_name
```

Использовать оператор DROP DATABASE в практической работе нужно очень осторожно. Он целиком удаляет базу данных и все таблицы, принадлежащие ей! После такого удаления база данных исчезает навсегда. Другими словами, не используйте этот оператор только для того, чтобы посмотреть, каков он в действии. Но если вы случайно все-таки удалили базу данных и есть надежда на то, что администратор регулярно сохраняет базу данных, вероятность ее восстановления остается. Но я не думаю, что администратор базы данных будет в восторге от вашей просьбы.

Обратите внимание на то, что база данных представлена подкаталогом в каталоге данных. Если в этом подкаталоге хранятся не табличные файлы, они не будут удалены. При удалении базы данных с помощью оператора DROP DATABASE такие файлы удалены не будут, не будет удален и сам подкаталог. Это значит, что каталог данной базы данных остается, не храня при этом никаких таблиц. Для того чтобы полностью удалить всю базу данных, удалите все оставшиеся файлы и сам каталог базы данных.

Изменение структуры базы данных

Оператор ALTER DATABASE, впервые появившийся в версии 4.1, вносит изменения в глобальные параметры баз данных. Сейчас единственной характеристикой такого рода является кодировка:

```
ALTER DATABASE db_name DEFAULT CHARACTER SET charset;
```

где *charset* является именем кодировки, поддерживаемой сервером, например `latin1_de` или `sjis`. (Определить кодировку, поддерживаемую сервером, можно с помощью оператора SHOW CHARACTER SET.) В качестве значения charset может быть ключевое слово DEFAULT, что отражает использование по умолчанию базой данных кодировки сервера. Детальное обсуждение темы кодировок можно найти в главе 2, "Работа с данными в MySQL". Атрибуты базы данных хранятся в файле `db.opt`, расположенном в каталоге базы данных.

Создание, удаление, индексирование и изменение структуры таблиц

СУБД MySQL позволяет создавать и удалять таблицы, модифицировать их структуру с помощью традиционных операторов SQL CREATE TABLE, DROP TABLE и ALTER TABLE. Операторы CREATE INDEX и DROP INDEX позволяют создавать или удалять индексы существующих таблиц. Но перед тем, как углубиться в детальное изучение индексов, необходимо разобраться с различными типами таблиц, которые поддерживает СУБД MySQL.

Типы таблиц

СУБД MySQL поддерживает различные типы таблиц, которые имеют специфический набор свойств и характеристик. Типы таблиц, которые действительно имеются в вашем распоряжении, зависят от версии СУБД MySQL, каким образом они настраивались при создании сервера и параметров, с которыми он был запущен. Текущие типы дескрипторов таблиц и версии, в которых они впервые появились, перечислены в следующей таблице.

Тип таблицы	Версия MySQL
ISAM	Все версии
MyISAM	3.23.0
MERGE	3.23.25
HEAP	3.23.0
BDB	3.23.17/3.23.34a
InnoDB	3.23.29/3.23.34a

Для типов BDB и InnoDB указаны две версии. Первый номер версии указывает, когда данный тип таблицы впервые появился в двоичных дистрибутивах, второй номер указывает, когда они появились в дистрибутивах с исходными текстами. Типы MRG_MyISAM и BerkeleyDB являются синонимами MERGE и BDB. (В версиях 3.23.29–3.23.36 тип таблиц InnoDB был известен как Innobase; позже появилось новое название InnoDB, хотя Innobase все еще остается в качестве синонима.)

Так как СУБД MySQL может настраиваться самыми различными способами, то вполне возможно, что она поддерживает не все типы таблиц, которые она может поддерживать. Информацию о том, какие типы поддерживает конкретный сервер, можно найти ниже, в разделе "Получение информации о базах данных и таблицах". Детальная информация о настройке сервера содержится в главе 11, "Общее администрирование MySQL".

Общие характеристики типов таблиц MySQL описываются в следующих разделах.

Таблицы ISAM

Тип ISAM предусматривает применение таблиц, в которых используется индексно-последовательный метод доступа. Формат хранения ISAM является первоначальным типом таблиц СУБД MySQL и был единственным до версии 3.23. Впоследствии тип ISAM был дополнен дескриптором MyISAM. Сейчас использование таблиц типа MyISAM предпочтительнее таблиц типа ISAM, так как они имеют меньше ограничений. Тип ISAM все еще существует, но считается устаревшим. Со временем его поддержка будет удалена. (Во встроенном сервере таблицы типа ISAM сейчас не поддерживаются, и, вероятнее всего, он исчезнет полностью в СУБД MySQL 5.)

Таблицы MyISAM

Формат хранения MyISAM является стандартным типом таблиц с версии 3.23, если сервер не был настроен по-другому.

- Этот тип позволяет создавать таблицы большего размера, чем тип ISAM, если это позволяет операционная система.
- Содержимое таблиц хранится в формате, не зависящем от платформы. Это значит, что таблицы можно копировать непосредственно из одной таблицы в другую, даже если они имеют различную архитектуру.

  По сравнению с типом ISAM, тип MyISAM снял некоторые ограничения по индексированию. Подробно этот вопрос рассматривается ниже, в разделе "Индексирование таблиц".
- Формат MyISAM позволяет эффективнее осуществлять сжатие ключей, чем это делает формат ISAM. Оба формата используют сжатие при хранении последовательностей одинаковых значений строковых индексов, так как в цифровых значениях сначала сохраняется старший байт. (Индексные значения имеют тенденцию к быстрому изменению в младших байтах, поэтому старшие байты без большого ущерба для данных можно сжимать.) Для запуска цифрового сжатия при создании таблицы необходимо задать параметр `PACK_KEYS=1`.
- Формат MyISAM имеет более мощный механизм работы с параметром `AUTO_INCREMENT`, чем это имеет место для других типов таблиц. Детально эта проблема обсуждается в разделе "Обработка последовательностей" в главе 2, "Работа с данными в MySQL".
- Для улучшения проверки целостности каждая таблица MyISAM имеет специальный флаг, который устанавливается при проверке таблиц сервером или программой `myisamchk`. Таблицы MyISAM имеют также флаг, указывающий на правильность закрытия таблицы. Если сервер отключается ненормально или аварийно останавливается компьютер, этот флаг используется для обнаружения таблиц, которые нуждаются в проверке. Все это задается автоматически во время запуска сервера параметром `--myisam-recover`.
- Дескриптор MyISAM поддерживает полный поиск по тексту. Для этих целей используются индексы типа `FULLTEXT`.

Таблицы типа MERGE

Таблицы типа MERGE представляют собой средство группировки таблиц типа MyISAM в единую логическую единицу. Отправляя запрос к таблице типа MERGE, вы осуществляете запрос ко всем составляющим ее таблицам. Одно из преимуществ этого заключается в том, что для таблиц данного типа можно превышать предельный размер таблиц, установленный для отдельных составляющих таблиц типа MyISAM.

Таблицы, составляющие таблицу типа MERGE, должны иметь одинаковую структуру. Это значит, что столбцы в каждой таблице должны объявляться с теми же именами, типами и в том же порядке. Это же справедливо и по отношению к индексам. Допустимо смешивать сжатые и несжатые таблицы. (Сжатые таблицы можно получить с помощью программы `myisampack`; подробно об этом рассказывается в приложении Д, "Программы MySQL".)

Таблицы типа MERGE не могут работать с таблицами из другой базы данных.

Таблицы типа HEAP

Формат хранения типа HEAP использует таблицы, хранящиеся в оперативной памяти, которые имеют строки фиксированной длины, что делает их довольно быстродействующими. Таблицы типа HEAP являются временными в том смысле, что они исчезают при остановке сервера. Однако, в отличие от временных таблиц, которые можно создать с помощью оператора `CREATE TEMPORARY TABLE`, таблицы типа HEAP являются видимыми для других клиентов. Несколько ограничений, которые имеют отношение к таблицам типа HEAP, позволяют обрабатывать их проще, а следовательно, и быстрее.

- Индексы используются только для операций сравнения по операторам = и <=>. Это происходит благодаря хешированию индексов, которое существенно ускоряет операции сравнения по равенству, чего нельзя сказать о процедуре поиска в заданном диапазоне (по условиям < или >). По этой же причине индексы не используются в выражениях `ORDER BY`.
- До версии 4.0.2 нельзя было использовать пустые значения в индексированных столбцах.
- До версии 4.1 не было возможности использовать столбцы с атрибутом `AUTO_INCREMENT`.
- Нельзя использовать столбцы типа `BLOB` и `TEXT`. Так как строки хранятся в формате с фиксированной длиной, нельзя также использовать столбцы переменной длины, такие как `BLOB` и `TEXT`. Использование типа `VARCHAR` разрешено, но внутренне он обрабатывается как тип `CHAR`.

Таблицы типа BDB

Таблицы типа BDB поддерживаются дескриптором Berkeley DB, разработанным компанией Sleepycat. Дескриптор BDB обеспечивает следующие возможности.

- Обработка таблиц с использованием транзакций и применением операций `commit` и `rollback`.
- Автоматическое восстановление после сбоев.
- Блокирование на уровне страниц, обеспечивающее хорошую производительность при обработке параллельных запросов, которые содержат выборки и модификации.

Таблицы типа InnoDB

Таблицы типа InnoDB являются самыми новыми таблицами, недавно добавленными в СУБД MySQL. Этот тип таблиц поддерживается дескриптором InnoDB, который был разработан компанией Innobase Oy. Дескриптор InnoDB обеспечивает следующие возможности.

- Обработка таблиц с использованием транзакций и применением операций `commit` и `rollback`.
- Автоматическое восстановление после сбоев.
- Поддержка внешних ключей, включая и каскадное удаление.

- Блокирование на уровне строк, обеспечивающее хорошую производительность при обработке параллельных запросов, включающих выборки и модификации.
- Таблицы типа InnoDB обрабатываются в пределах отдельного табличного пространства, а не с использованием табличных файлов, что является характерным отличием таблиц этого типа от таблиц других типов. Табличное пространство может содержать несколько файлов и может включать разделы. Дескриптор InnoDB в действительности работает с табличным пространством как с виртуальной файловой системой, в пределах которой он хранит все таблицы типа InnoDB.
- Таблицы могут выходить за пределы, установленные файловой системой для отдельных файлов. Для этого применяются множественные файлы или разделы табличного пространства.

Представление таблиц на диске

Каждая таблица независимо от ее формата представлена на диске файлом, который содержит формат таблицы (т.е. ее определение). Этот файл имеет имя, соответствующее имени таблицы, и расширение `.frm`. Для большинства типов таблиц содержимое таблиц хранится на диске с использованием других файлов, уникальных для таблицы. Исключением являются таблицы типов HEAP и InnoDB, для которых файл `.frm` является единственным файлом, связанным уникальным образом с данной таблицей. (Содержимое таблиц HEAP хранится в оперативной памяти. Содержимое таблиц типа InnoDB обрабатывается в пределах табличного пространства InnoDB вместе с другими таблицами типа InnoDB, а не в отдельных файлах, хранящих определенные таблицы.) Различные типы таблиц используют файлы со следующими расширениями.

Тип таблицы	Файлы на диске
ISAM	.frm (определение), .ISD (данные), .ISM (индексы)
MyISAM	.frm (определение), .MYD (данные), .MYI (индексы)
MERGE	.frm (определение), .MRG (список составляющих таблиц типа MyISAM)
HEAP	.frm (определение)
BDB	.frm (определение), .db (данные и индексы)
InnoDB	.frm (определение)

Файлы, содержащие таблицы баз данных, располагаются в соответствующих каталогах, представляющих базу, которой принадлежит данная таблица.

Переносимость различных типов таблиц

Любая таблица может быть переносимой в том смысле, что ее можно скопировать в текстовый файл с помощью программы `mysqldump`, перенести файл на машину, где работает другой сервер, и загрузить этот файл в восстановленную таблицу. В этом разделе о переносимости говорится в том смысле, что можно непосредственно скопировать на другую машину представляющие таблицу файлы,

установив их в каталоге базы данных, и быть полностью уверенным, что сервер СУБД MySQL сможет работать с этой таблицей. Конечно же, исключением из этого правила являются таблицы типа HEAP, которые не удовлетворяют этому определению, так как их содержимое хранится не на диске, а в оперативной памяти. Что же касается других типов таблиц, то одни из них переносимы, а другие – нет.

- Таблицы ISAM хранятся в машинно-зависимом формате, поэтому они переносимы только между машинами, имеющими идентичные характеристики оборудования.
- Таблицы типа BDB не переносимы, так как размещение таблиц закодировано в табличном файле типа .db. Это делает размещение таблиц BDB привязанным к файловой системе машины, на которой она была создана. (Это несколько консервативный взгляд на переносимость таблиц BDB. Я имел возможность экспериментировать с файлами BDB, перемещая их между различными каталогами, переименовывая файлы и т.д., и отрицательных результатов это не дало. Но лучше всего застраховаться и переносить таблицы типа BDB, используя программу `mysqldump` и воссоздавая их на принимающей машине после загрузки архивного файла.)
- Таблицы типа MyISAM и InnoDB хранятся в машинно-независимом формате и являются переносимыми при допущении, что ваш процессор использует целочисленную арифметику. Кроме того случая, когда ваш компьютер совсем не соответствует никаким стандартам, ни одно из этих условий не должно создавать никаких проблем. На практике при использовании встроенного сервера для устройства специального назначения вы скорее всего попытаетесь найти компромиссное решение, так как в них зачастую используются процессоры, имеющие нестандартные рабочие характеристики.
- Таблицы типа MERGE переносимы настолько, насколько переносимы составляющие их таблицы типа MyISAM.

В сущности, требования к переносимости таблиц типа MyISAM и InnoDB таковы, что они не могут иметь в своей структуре столбцов вещественного типа или обе машины должны использовать одинаковый формат хранения вещественных чисел. Это типы `FLOAT` и `DOUBLE`. Столбцы `DECIMAL` хранятся как строки, которые с этой точки зрения нареканий не вызывают.

Заметим, что для типа InnoDB переносимость должна быть присуща на уровне табличного пространства, а не на уровне таблиц. Дескриптор таблиц типа InnoDB хранит содержимое всех таблиц типа InnoDB в табличном пространстве, а не в индивидуальных таблицах типа InnoDB. Это значит, что ограничение по переносимости относительно вещественного формата чисел формулируется так: "если *хотя бы одна* из таблиц типа InnoDB использует вещественные столбцы".

Независимо от общих характеристик переносимости, присущих определенному типу таблицы, не следует даже пытаться копировать табличные файлы или файлы с табличным пространством, если предварительно сервер не был корректно отключен. При выполнении копирования после некорректного отключения сервера целостность таблиц нельзя гарантировать. При этом может потребоваться определенная очистка, поскольку информация об определенных транзак-

циях может по-прежнему храниться в регистрационном файле дескриптора таблицы и эта информация еще ждет своей обработки. Или должен быть осуществлен откат, чтобы таблицы вновь стали актуальными.

Аналогично, если сервер работает и активно модифицирует таблицы, содержимое таблиц на диске будет постоянно изменяться и связанные файлы не позволят получить копии используемых таблиц. Но и при работающем сервере ему можно дать знать, чтобы он не использовал таблицу, пока вы ее копируете. Подробную информацию на эту тему можно найти в главе 13, "Резервное копирование и восстановление баз данных".

Создание таблиц

Создание таблиц осуществляется с помощью оператора CREATE TABLE. Полный синтаксис этого оператора имеет много вариантов, но на практике его использовать нетрудно. Например, все операторы CREATE TABLE, которые были использованы в главе 1, "Знакомство с СУБД MySQL и SQL", достаточно просты.

Оператор CREATE TABLE объявляет имя таблицы и перечень столбцов, ее составляющих:

```
CREATE TABLE mytbl
(
   name CHAR(20),
   age INT NOT NULL,
   weight INT,
   sex ENUM('F', 'M')
);
```

Кроме перечня столбцов, составляющих таблицу, в момент создания таблицы можно указать, как таблица индексируется. Есть и другой вариант: при создании таблицы не индексировать ее, а оставить эту процедуру на будущее. (Такая стратегия уместна для таблиц ISAM и MyISAM, если таблицы сначала наполняются данными, а уже потом начинается их использование. Модификация индексов при добавлении новых записей отнимает гораздо больше времени, чем простое добавление записей в неиндексированную таблицу с последующим созданием индекса.)

В главе 1, "Знакомство с СУБД MySQL и SQL", мы уже рассмотрели основной синтаксис оператора CREATE TABLE. Описание столбцов таблиц приведено в главе 2, "Работа с данными в MySQL". Я предполагаю, что читатель знаком с этими главами, поэтому не будем повторяться. Обратим внимание на добавления, сделанные в синтаксисе оператора CREATE TABLE в версии 3.23 (они перечислены ниже). Это придает гибкость процедуре создания таблиц:

- спецификация типа памяти;
- создание таблицы только в том случае, если ее не существует (условное создание таблиц);
- временные таблицы, которые удаляются сразу же по завершении сеанса соединения;
- возможность создания таблицы методом выборки данных из уже существующих таблиц;
- использование таблиц типа MEGRE.

Спецификация типа таблицы

Начиная с версии 3.23 в операторе CREATE TABLE после закрывающей скобки появилась возможность влиять на характеристики форматов хранения таблиц. Например, до появления версии 3.23 все таблицы создавались только в формате ISAM, так как он был тогда единственным форматом. Начиная с версии 3.23 появилась возможность задавать тип таблицы TYPE = *tbl_type* явным образом. Например, для того чтобы создать таблицу HEAP или InnoDB, можно воспользоваться оператором вида (имя типа таблицы нечувствительно к регистру):

```
CREATE TABLE mytbl ( ... ) TYPE = HEAP;
CREATE TABLE mytbl ( ... ) TYPE = INNODB;
```

Без спецификации TYPE сервер создает таблицу с использованием типа, принятого по умолчанию. Скорее всего это будет тип MyISAM, если сервер не будет настроен для использования в качестве стандартного другого типа. Это можно сделать при создании сервера или с помощью параметра --default-table-type при запуске сервера. Задавая имя типа таблицы, которое является синтаксически допустимым, но дескриптор которой отсутствует, СУБД MySQL создает таблицу с использованием стандартного типа. При указании несуществующего типа таблиц диагностируется ошибка.

Могут быть заданы и другие параметры таблиц. Многие из них относятся только к конкретным типам таблиц. Например, параметр MIN_ROWS = n применяется только с таблицами типа HEAP и позволяет дескриптору HEAP оптимизировать использование памяти:

```
CREATE TABLE mytbl ( ... ) TYPE = HEAP MIN_ROWS = 10000;
```

При больших значениях параметра MIN_ROWS дескриптор может размещать данные в больших блоках, чтобы не создавать перегрузок одними запросами перераспределения памяти. Полный перечень параметров таблиц перечислен в разделе, посвященном оператору CREATE TABLE, приложения Г, "Синтаксис SQL".

Для внесения изменений в характеристики таблицы существующих таблиц эти же параметры могут использоваться с оператором ALTER TABLE. Например, для того чтобы преобразовать текущий тип таблицы mytbl в тип InnoDB, необходимо сделать следующее:

```
ALTER TABLE mytbl TYPE = INNODB;
```

Допустимые типы при преобразовании таблицы могут зависеть от совместимости такой возможности между старыми и новыми типами. Предположим, что в нашем распоряжении есть таблица, содержащая столбец BLOB. Эту таблицу нельзя преобразовать в формат HEAP, так как таблицы HEAP не поддерживают столбцы BLOB.

Условное создание таблиц

Оператор CREATE TABLE IF NOT EXISTS предназначен для создания таблицы по условию ее существования. Этот прием появился впервые в версии 3.23.0. Таким приемом можно воспользоваться в приложении, которое при создании таблицы предварительно не проверяет факта ее существования. Модификатором IF NOT EXISTS можно воспользоваться в сценариях, запускающихся из

пакетных файлов. В этом варианте обычный оператор CREATE TABLE подходит не всегда. Если при первом запуске такого сценария таблицы создаются успешно, то при последующих запусках происходит ошибка, поскольку создаваемая таблица уже существует. При использовании предложения IF NOT EXISTS таких проблем никогда не возникает. При первом запуске задания, как и в предыдущем случае, таблица будет создана. При последующих попытках создания файлов файл не создается, но и не генерируется код ошибки. Задание продолжает успешно выполняться.

Временные таблицы

Для создания временных таблиц предусмотрен специальный оператор CREATE TEMPORARY TABLE. Такие таблицы автоматически исчезают после завершения сеанса работы с базой данных. Это очень удобно, так как отпадает необходимость специально удалять таблицы такого типа. Предположим, у вас работает длительный запрос и вы решаете прервать его выполнение. При этом можно совершенно не беспокоиться за судьбу временных таблиц, созданных в процессе выполнения запроса, — они все будут успешно удалены.

Временная таблица доступна только клиенту, ее создавшему. Имя временной таблицы может совпадать с именем уже существующей постоянной таблицы. Такое повторение не является ошибкой. Создадим временную таблицу member в базе данных sampdb. Постоянная таблица member на период существования временной таблицы становится недоступной, и все ссылки на таблицу member будут переадресованы на временную таблицу member. Оператор DROP TABLE member удаляет временную таблицу, и первоначальная таблица member становится видимой снова. Аналогичного эффекта можно добиться простым отключением от сервера. При последующем подключении основная таблица member снова становится доступной. (Первоначальная таблица также станет опять доступной при переименовании временной таблицы. Но если при этом новое имя тоже является повторением уже существующего имени, эта таблица также станет недоступной на время существования временной таблицы.)

Этот механизм имеет только один уровень, т.е. невозможно создать две временные таблицы с одним именем.

Временная таблица может создаваться с указанием определенного типа с помощью параметра TYPE. (До версии 3.23.54 таблицы типа MERGE не могли быть временными, т.е. TEMPORARY.)

До версии 3.23.2 в СУБД MySQL настоящих временных таблиц не было, за исключением тех, которые считает таковыми сам разработчик. При создании приложений, использующих такие таблицы, разработчик должен сам помнить о необходимости удалять эти таблицы из базы данных. В противном случае они будут постоянно "путаться под руками" и даже могут провоцировать возникновение ошибочных ситуаций.

Создание таблиц по результатам выборки данных

Одной из ключевых концепций реляционных баз данных является представление всей информации в виде таблиц, состоящих из столбцов и строк. Результат работы каждого оператора SELECT также является таблицей, состоящей из

столбцов и строк. В любом случае "таблица", являющаяся результатом работы оператора SELECT, изображается в виде строк и столбцов. Иногда требуется сохранять результаты выборки в таблицах, чтобы иметь возможность ссылаться на них позднее.

Начиная с версии 3.23 сделать это очень просто. Теперь оператор CREATE TABLE ... SELECT позволяет создавать таблицу на основании произвольного оператора SELECT. В этом случае все можно сделать одним действием, не определяя при этом структуру создаваемой таблицы. Это исключительно облегчает задачу создания таблиц, уже заполненных данными. Например, следующий оператор создает новую таблицу student_f, содержащую информацию обо всех ученицах из таблицы student:

```
CREATE TABLE student_f SELECT * FROM student WHERE sex = 'f';
```

Для того чтобы скопировать всю таблицу, опустите выражение WHERE:

```
CREATE TABLE new_tbl_name SELECT * FROM tbl_name;
```

Таким образом, таблицу можно скопировать полностью или с помощью заведомо неработающего предложения WHERE:

```
CREATE TABLE new_tbl_name SELECT * FROM tbl_name WHERE 0
```

Таким приемом можно воспользоваться при загрузке данных в таблицу оператором LOAD DATA, когда нет полной уверенности в том, что формат данных определен верно, и необходимо немного поэкспериментировать со структурой таблицы. Пустая копия позволит делать это до тех пор, пока пользователь не будет полностью удовлетворен результатами. После этого данные можно смело загружать в реальную таблицу. Это можно сделать, повторно запустив оператор LOAD DATA с оригинальным именем таблицы или скопировав данные в нее из копии:

```
INSERT INTO orig_tbl SELECT * FROM copy_tbl;
```

Аналогичная последовательность действий допустима и для оператора CREATE TEMPORARY TABLE:

```
CREATE TEMPORARY TABLE mytbl SELECT * FROM mytbl;
```

Это позволяет модифицировать содержимое таблицы mytbl, не нанося никакого вреда основной таблице. Такая возможность окажет существенную помощь при отладке запросов, изменяющих содержимое таблицы. При этом не нужно подвергать серьезным изменениям уже существующие сценарии. Достаточно в начало сценария добавить оператор CREATE TEMPORARY TABLE. Такой сценарий создаст временную копию и будет ее обрабатывать, а после завершения работы сценария сервер удалит временную копию. (Необходимо помнить, что такие клиенты, как mysql, делают попытку автоматически восстановить соединение при его разрыве. В этом случае после восстановления соединения все запросы, работавшие с временной таблицей, после восстановления соединения будут работать с основной таблицей. Это необходимо принимать во внимание при работе с ненадежной сетью.)

Для создания таблицы, являющейся пустой копией самой себя, воспользуйтесь комбинацией предложения WHERE с оператором CREATE TEMPORARY ... SELECT:

```
CREATE TEMPORARY TABLE mytbl SELECT * FROM mytbl WHERE 0;
```

Такой метод создания таблиц "на лету" является мощным инструментом, но он имеет свои недостатки. И о них необходимо знать.

При создании таблицы таким образом ее столбцы получают имена, соответствующие именам таблицы-родителя. Вычисляемые столбцы могут иметь в качестве имени текст выражения. До версии 3.23.6 выражение в качестве имени столбца было недопустимо. В этом легко убедиться, запустив следующий запрос:

```
mysql> CREATE TABLE mytbl SELECT PI();
Error 1166 at line 1: Incorrect column name 'PI()'
Ошибка 1166 в строке 1: Ошибочное имя столбца 'PI()'
```

Начиная с версии 3.23.6 правила присвоения имен столбцам упростились, и теперь этот оператор сработает, создав при этом столбец с несколько необычным именем:

```
mysql> CREATE TABLE mytbl SELECT PI();
mysql> SELECT * FROM mytbl;
+----------+
| PI()     |
+----------+
| 3.141593 |
+----------+
```

При этом ссылаться на имя столбца можно только после заключения его в обратные одинарные кавычки:

```
mysql> SELECT `PI()` FROM mytbl;
+----------+
| PI()     |
+----------+
| 3.141593 |
+----------+
```

Для того чтобы упростить работу по выборке выражения, лучше присвоить ему псевдоним:

```
mysql> CREATE TABLE mytbl SELECT PI() AS mycol;
mysql> SELECT mycol FROM mytbl;
+----------+
| mycol    |
+----------+
| 3.141593 |
+----------+
```

Аналогичная ситуация возникает и при создании новых таблиц на основании данных, взятых из двух таблиц, имеющих в своем составе одноименные столбцы. Предположим, что таблицы `t1` и `t2` имеют в своей структуре столбец `c`. Следующий оператор не будет выполнен из-за существования в двух таблицах одноименных столбцов:

```
mysql> CREATE TABLE t3 SELECT * FROM t1, t2;
ERROR 1060: Duplicate column name 'c'
Ошибка 1060: Дублирование имени столбца 'c'
```

Чтобы решить эту проблему, присвоим псевдонимы столбцам с из обеих таблиц:

```
mysql> CREATE TABLE t3 SELECT t1.c AS c1, t2.c AS c2 FROM t1, t2;
```

Другим моментом, который никогда не следует упускать из виду, является то, что характеристики исходных таблиц, которые не отображены в выбранных дан-

ных, не будут учтены в структуре новой таблицы. Например, при создании таблиц с помощью выборки данных из других таблиц ни один индекс из первоначальных таблиц автоматически не копируется, так как результирующие наборы не самоиндексируются. Аналогично, такие атрибуты, как AUTO_INCREMENT, или значения по умолчанию в новую таблицу могут не скопироваться. (Новые версии лучше старых.) В некоторых случаях можно заставить использовать атрибуты в новой таблице, запустив функцию CAST(), которая появилась в СУБД MySQL 4.0.2. Оператор CREATE TABLE ... SELECT позволит рассматривать столбцы, созданные оператором SELECT, как INT UNSIGNED, DATE и CHAR BINARY, что можно проверить с помощью оператора DESCRIBE:

```
mysql> CREATE TABLE mytbl SELECT
    -> CAST(1 AS UNSIGNED) AS i,
    -> CAST(CURDATE() AS DATE) AS d,
    -> CAST('Hello, world' AS BINARY) AS c;
mysql> DESCRIBE mytbl;
+-------+-----------------+------+-----+------------+---------+
| Field | Type            | Null | Key | Default    | Extra   |
+-------+-----------------+------+-----+------------+---------+
| i     | int(1) unsigned |      |     | 0          |         |
| d     | date            |      |     | 0000-00-00 |         |
| c     | char(12) binary |      |     |            |         |
+-------+-----------------+------+-----+------------+---------+
```

Функцию CAST() можно применить также к значениям столбцов из других таблиц. Допустимыми переносимыми типами являются BINARY (двоичная строка), DATE, DATETIME, TIME, SIGNED, SIGNED INTEGER, UNSIGNED и UNSIGNED INTEGER.

В версии 4.1 появилась возможность передачи в новую таблицу дополнительной информации о столбцах. Это достигается с помощью явных объявлений. Столбцы в таблице соответствуют выбранным столбцам по имени, таким образом, для того, чтобы обеспечить полное соответствие, необходимо присваивать столбцам псевдонимы:

```
mysql> CREATE TABLE mytbl (i INT UNSIGNED, d DATE, c CHAR(20) BINARY)
    -> SELECT
    -> 1 AS i,
    -> CURDATE() AS d,
    -> 'Hello, world' AS c;
mysql> DESCRIBE mytbl;
+-------+-----------------+------+-----+---------+-------+
| Field | Type            | Null | Key | Default | Extra |
+-------+-----------------+------+-----+---------+-------+
| i     | int(10) unsigned | YES  |     | NULL    |       |
| d     | date            | YES  |     | NULL    |       |
| c     | char(20) binary | YES  |     | NULL    |       |
+-------+-----------------+------+-----+---------+-------+
```

Обратите внимание на то, что это позволяет создавать символьные столбцы, имеющие длину, отличную от длины результирующего набора. Также заметим, что атрибуты Null и Default в этом примере отличаются от тех, которые мы рассматривали в предыдущем примере. По необходимости можно задать явные объявления для таких атрибутов.

До версии 3.23 оператора CREATE TABLE ... SELECT в арсенале СУБД MySQL не было. Чтобы сохранить результаты работы оператора SELECT для использования в последующих запросах, необходимо предварительно выполнить определенные подготовительные действия.

1. Запустите запросы DESCRIBE или SHOW COLUMNS, чтобы определить типы столбцов, имеющиеся в таблицах, из которых вы копируете информацию.
2. Введите явный оператор CREATE TABLE, чтобы создать таблицу, в который вы будете хранить результаты оператора SELECT. Этот оператор должен определить имена и типы столбцов, которые будут выбраны оператором SELECT.
3. После создания таблицы введите запрос INSERT INTO ... SELECT для выборки результатов и ввода их в таблицу. По сравнению с оператором CREATE TABLE ... SELECT в этой процедуре присутствует некоторая избыточность.

Работа с таблицами MERGE

Таблицы типа MERGE, впервые появившиеся в версии 3.23.25, обеспечивают обработку запросов к таблицам, рассматривая их как единую логическую единицу. Как отмечалось в разделе "Типы таблиц", тип MERGE применим к группе таблиц типа MyISAM, имеющих идентичную структуру. Предположим, что у вас имеется набор таблиц с регистрационными данными, собранными по годам, которые можно представить в формате CCYY (столетие и год соответственно):

```
CREATE TABLE log_CCYY
(
   dt DATETIME NOT NULL,
   info VARCHAR(100) NOT NULL,
   INDEX (dt)
) TYPE = MYISAM;
```

Текущий набор регистрационных таблиц включает таблицы log_1999, log_2000, log_2001, log_2002 и log_2003, но с помощью следующего оператора их можно объединить в единую таблицу типа MERGE:

```
CREATE TABLE log_all
(
   dt DATETIME NOT NULL,
   info VARCHAR(100) NOT NULL,
   INDEX (dt)
) TYPE = MERGE UNION = (log_1999, log_2000, log_2001, log_2002, log_2003);
```

Параметр TYPE должен иметь значение MERGE, а в параметре UNION дается перечень таблиц, которые будут объединены в таблицу MERGE. После создания этой таблицы к ней можно обращаться как к любой другой таблице, про этом запросы будут направляться ко всем таблицам одновременно. Ниже приведен запрос, по которому определяется суммарное количество строк во всех регистрационных таблицах.

```
SELECT COUNT(*) FROM log_all;
```

А этот запрос определяет, сколько записей было сделано в году:

```
SELECT YEAR(dt) AS y, COUNT(*) AS entries FROM log_all GROUP BY y;
```

Кроме удобства, позволяющего работать с несколькими идентичными таблицами, не создавая при этом нескольких запросов, таблицы типа MERGE предлагают еще несколько отличных возможностей.

- Таблица типа MERGE может быть использована для создания логического объекта, превышающего размеры, допустимые для отдельных таблиц типа MyISAM.
- В группу можно включить сжатые таблицы. Например, в конце года для сохранения дискового пространства файл архивируется с помощью программы `myisampack`. Таблица типа MERGE позволит работать с такой таблицей, как и прежде.
- Операции с таблицами типа MERGE аналогичны операциям `UNION`. Операции `UNION` в СУБД MySQL отсутствовали до версии 4, но таблицы типа MERGE могут использоваться как рабочая среда.

Таблицы типа MERGE также поддерживают операции `DELETE` и `UPDATE`. С операцией `INSERT` дело обстоит немного сложнее, так как СУБД MySQL должна иметь информацию о том, в какую таблицу следует добавлять новые строки. Начиная с версии 4.0.0 определения таблиц MERGE могут включать параметр `INSERT_METHOD` со значениями `NO`, `FIRST` или `LAST`, чтобы показать, что операции `INSERT` запрещены или что записи должны вставляться в первую или последнюю таблицу, перечисленную в параметре `UPDATE`. Например, вот объявление, в соответствии с которым операция добавления в таблицу `log_all` будет производиться как операция добавления в таблицу `log_2003`, которая была последней таблицей, перечисленной в параметре `UNION`:

```
CREATE TABLE log_all
(
   dt DATETIME NOT NULL,
   info VARCHAR(100) NOT NULL,
   INDEX (dt)
) TYPE = MERGE UNION = (log_1999, log_2000, log_2001, log_2002,
log_2003) INSERT_METHOD = LAST;
```

Удаление таблиц

Процедура удаления таблицы значительно проще процедуры создания, поскольку при этом не требуется предварительно определять структуру таблицы. Достаточно только указать ее имя:

```
DROP TABLE tbl_name;
```

Функции оператора `DROP TABLE` в СУБД MySQL несколько расширены. Во-первых, появилась возможность удалять сразу несколько таблиц, указав их имена в одном операторе:

```
DROP TABLE tbl_name1, tbl_name2 ...;
```

Во-вторых, с помощью условного оператора `IF EXISTS` можно проверить, существует ли такая таблица:

```
DROP TABLE IF EXISTS tbl_name;
```

Такая конструкция оператора `DROP TABLE` может очень пригодиться в работе с клиентской программой `mysql`. При обнаружении ошибки он обычно завершa-

ет работу, а ошибочная ситуация может возникнуть при попытке удалить несуществующую таблицу. Предположим, что есть сценарий, создающий таблицы, задействованные в других сценариях. В таком случае необходимо убедиться, что этот сценарий имеет "поле деятельности" для своей работы. При первом запуске сценария, в начале которого используется обычный оператор DROP TABLE, обязательно произойдет сбой, потому что такие файлы никогда не создавались. При использовании модифицированного синтаксиса с использованием IF EXISTS таких проблем не возникает. Если таблицы существуют, они будут удалены, если нет – выполнение сценария продолжается дальше.

Индексирование таблиц

Индексы являются основным средством ускорения доступа к содержимому таблиц; особенно это касается запросов, включающих объединение нескольких таблиц. Это достаточно важная тема, которая обсуждается в главе 4, "Оптимизация запросов" (там вы узнаете, зачем используются индексы, как они работают, как с их помощью можно оптимизировать работу запросов). В этом разделе мы рассмотрим характеристики индексов и синтаксис, который используется для их создания и удаления.

Характеристики индексов

СУБД MySQL обеспечивает достаточно гибкие способы создания индексов.

- Таблицу можно индексировать как по одному столбцу, так и по комбинации столбцов.
- Индексу может быть задан режим содержания повторяющихся значений или содержания только уникальных значений.
- Любой строковый тип (кроме ENUM и SET) можно индексировать по первым n символам слева. (Нельзя создавать индексы по столбцам типа BLOB и TEXT, пока не задана длина префикса.) Префиксы могут иметь длину до 255 байт. Если первые n символов уникальны, это уменьшает размер индекса и увеличивает скорость обработки.

Конечно, на создание индексов тоже существуют некоторые ограничения, но по мере развития СУБД MySQL их круг будет сужаться. Ниже приведена таблица, которая отражает различие возможностей индексирования таблиц в форматах ISAM и MyIASM. (Эта таблица не содержит характеристик типа MERGE, так как таблицы этого типа создаются из таблиц типа MyISAM и имеют аналогичные характеристики индексирования.)

Характеристика индекса	ISAM	MyISAM	Heap	BDB	InnoDB
Пустые значения	Не допустимы	Допустимы	Начиная с версии 4.0.2	Допустимы	Допустимы
Столбцы типа BLOB и TEXT	Не индексируются	Индексируются	Индексируются	Индексируются	Индексируются
Количество столбцов на один индекс	16	16	16	16	16

Характеристика индекса	ISAM	MyISAM	Heap	BDB	InnoDB
Количество индексов на одну таблицу	16	32	32	31	32
Максимальный размер строки индекса	256 байт	500 байт	500 байт	500/1024 байт	500/1024 байт
Допустимость индексирования по типам BLOB и TEXT	Нет	Да (максимально 255 байт)	Нет	Да (максимально 255 байт)	Нет

В строке, отражающей максимальный размер строки индекса для типов BDB и InnoDB, показаны два числа. Для таблиц этих типов до версии 4.0.3 размер равен 500 байтам и 1024 байтам – в последующих версиях. Из этой таблицы видно, что таблицы, созданные в формате ISAM, не могут иметь пустых значений в индексируемом столбце. Кроме того, недопустимо индексирование столбцов типа BLOB и TEXT. Формат MyISAM снимает все эти ограничения.

Одним из проявлений этих различий для двух данных типов столбцов является то, что в зависимости от версии СУБД MySQL индекс создать будет просто невозможно. Например, в СУБД MySQL до версии 3.23 работает только метод ISAM. Это не позволяет индексировать столбцы, содержащие пустые значения. Следовательно, если вы хотите, чтобы индекс обладал определенными свойствами, следует использовать определенные типы таблиц. Если требуется индексирование по столбцам типа BLOB, необходимо использовать только таблицы типа MyISAM или BDB.

Преобразовать таблицу из одного типа в другой можно с помощью оператора ALTER TABLE. Начиная с версии 3.23 возможен экспорт старых таблиц из формата ISAM в формат MyISAM. Такую возможность предоставляет оператор ALTER TABLE. Это позволяет воспользоваться новыми возможностями индексирования.

```
ALTER TABLE tbl_name TYPE = MYISAM;
```

Создание индексов

СУБД MySQL позволяет создавать несколько типов индексов.

- **Обычный (неуникальный) индекс.** Индекс такого типа предоставляет все выгоды индексирования, позволяя при этом дублировать записи.
- **Уникальный индекс.** Этот тип индекса запрещает дублирование записей. Для индекса, состоящего из одного столбца, это гарантирует отсутствие повторяющихся значений. Для индексов, содержащих несколько столбцов (сложных индексов), это гарантирует отсутствие повторяющихся комбинаций столбцов.
- **Индекс типа FULLTEXT, который используется при осуществлении поиска по контексту.** Индекс этого типа поддерживается только для таблиц типа MyISAM. (Более полная информация содержится ниже, в разделе "Полнотекстный поиск типа FULLTEXT".)

Индексы можно создавать в процессе создания таблицы с помощью оператора CREATE TABLE. Посредством операторов CREATE INDEX и ALTER TABLE можно

создавать индексы для уже существующих таблиц. Оператор CREATE INDEX был добавлен в версии 3.22, в более ранних версиях для этой цели можно воспользоваться оператором ALTER TABLE. (СУБД MySQL предусматривает внутреннюю отработку операторов CREATE INDEX в операциях ALTER TABLE.)

Создать индекс для существующей таблицы можно с помощью оператора ALTER TABLE или CREATE INDEX. При этом следует учитывать, что возможности оператора ALTER TABLE несколько шире. В СУБД MySQL с его помощью можно создать обычный индекс, индексы UNIQUE, PRIMARY KEY и FULLTEXT.

```
ALTER TABLE tbl_name ADD INDEX index_name (index_columns);
ALTER TABLE tbl_name ADD UNIQUE index_name (index_columns);
ALTER TABLE tbl_name ADD PRIMARY KEY (index_columns);
ALTER TABLE tbl_name ADD FULLTEXT (index_columns);
```

где *tbl_name* — имя индексируемой таблицы, *index_columns* — список индексируемых столбцов. Если индекс состоит из нескольких столбцов, они разделяются в списке запятыми. Имя индекса *index_name* можно не указывать. В случае его отсутствия СУБД MySQL присвоит индексу имя первого индексируемого столбца. Оператор ALTER TABLE позволяет создавать с помощью одного оператора сразу несколько индексов. (Это будет быстрее, чем создавать по одному индексу одним оператором в отдельности.)

В СУБД MySQL с помощью оператора CREATE INDEX можно создать только обычный индекс и индекс UNIQUE.

Для того чтобы индекс содержал только уникальные значения, необходимо создать индекс как PRIMARY KEY или UNIQUE. Индексы PRIMARY KEY и UNIQUE подобны (просто индекс PRIMARY KEY представляет собой индекс UNIQUE с именем PRIMARY), хотя между ними есть два различия.

- Таблица может иметь только один индекс PRIMARY KEY, поскольку нельзя иметь два индекса с одним и тем же именем PRIMARY. Таблица может содержать несколько индексов UNIQUE, хотя это и несколько неестественно.
- В отличие от индекса UNIQUE, индекс типа PRIMARY KEY не может содержать пустых значений. При этом индекс UNIQUE может содержать несколько пустых значений. Причиной этого является то, что не предоставляется возможности сравнения двух пустых значений, поэтому они не могут рассматриваться как равные. (Таблицы типа BDB являются исключением — они допускают наличие только одного пустого значения в индексе UNIQUE.)

Оператор CREATE INDEX позволяет добавить обычный индекс, индекс UNIQUE или FULLTEXT, но не позволяет создать индекс PRIMARY KEY:

```
CREATE UNIQUE INDEX index_name ON tbl_name (index_columns);
CREATE INDEX index_name ON tbl_name(index_columns);
CREATE FULLTEXT INDEX index_name ON tbl_name(index_columns);
```

где *tbl_name*, *index_name* и *index_columns* имеют значение, аналогичное принятому для оператора ALTER TABLE. Но в этом операторе необходимо указать имя индекса. С помощью одного оператора CREATE INDEX нельзя создать множественный индекс.

При создании индекса в момент создания таблицы используется синтаксис, аналогичный синтаксису оператора ALTER TABLE, но предложения, от-

вечающие за создание индекса, указываются в том же месте, где объявляются столбцы таблицы:

```
CREATE TABLE tbl_name
(
   ...
   INDEX index_name (index_columns),
   UNIQUE index_name (index_columns),
   PRIMARY KEY (index_columns),
   FULLTEXT (index_columns),
   ...
);
```

Как и в операторе ALTER TABLE, имя индекса здесь можно не указывать. СУБД MySQL в таком случае присвоит имя сама.

Есть одно исключение, позволяющее задавать индекс непосредственно при объявлении столбца, но это можно сделать только для индекса PRIMARY KEY, содержащего один столбец. Начиная с СУБД MySQL 3.23 можно поступать подобным образом и с индексом UNIQUE:

```
CREATE TABLE mytbl
(
   i INT NOT NULL PRIMARY KEY,
   j CHAR(10) NOT NULL UNIQUE
);
```

Этот оператор эквивалентен следующему:

```
CREATE TABLE mytbl
(
   i INT NOT NULL,
   j CHAR(10) NOT NULL,
   PRIMARY KEY (i),
   UNIQUE (j)
);
```

Все примеры создания таблиц предусматривают указание NOT NULL для индексируемых столбцов. Этого правила необходимо придерживаться для таблиц, созданных методом ISAM (и для таблиц HEAP до версии 4.0.2), где индексируемые столбцы не могут содержать пустых значений. Для таблиц других типов индексируемые столбцы могут иметь пустые значения. Но это справедливо только тогда, когда индекс не является индексом PRIMARY KEY.

Существует возможность индексирования по префиксу столбца строкового типа (по первым n символам строки). При этом спецификация меняется с *column_list* на *column_list(n)*. Например, в этом операторе при создании индекса используется только часть столбца:

```
CREATE TABLE mytbl
(
   name CHAR(30),
   address CHAR(60),
   INDEX (name(10), address(10))
);
```

Индексирование по префиксам поддерживается для таблиц типа ISAM, MyISAM, HEAP и BDB, но не поддерживается для таблиц типа InnoDB.

Длина префикса, в отличие от длины столбца, относится к байтам, а не к символам. И если для однобайтовых кодировок это одно и то же, то для многобайтовых – нет. СУБД MySQL будет хранить столько полных символов, сколько их туда входит. Например, если префикс индекса имеет длину 5 байтов, а столбец содержит 2-байтовые символы, индекс будет содержать 2-байтовые символы, в индексе будут храниться два символа, а не 2,5.

В некоторых случаях индексирование по префиксу может оказаться как нельзя кстати. Длина строки индекса ограничена, поэтому для индексирования по строке, длина которой выходит за допустимый диапазон, такая возможность очень удобна.

- Индексирование по префиксу уместно при индексировании по столбцу типа BLOB или TEXT в таблицах типов, для которых это разрешено. Префикс может иметь длину до 255 байт.
- Длина строки индекса равна суммарной длине индексируемых частей столбцов, входящих в индекс. Если эта длина превышает допустимую длину индексируемых строк, можно "ограничиться" индексированием префикса столбца. Предположим, что таблица типа MyISAM содержит два столбца типа CHAR(255), c1 и c2 и необходимо выполнить индексирование по обоим столбцам. Таким образом, суммарная длина строки индекса будет составлять 255+255, что превышает ограничение, существующее для индексов таблиц типа MyISAM, в соответствии с которым длина строки индекса не может превышать 500 байтов. В этом случае можно создать индекс, в котором используется только часть одного или обоих столбцов.

Индексирование по префиксу накладывает определенные ограничения на возможность внесения изменений в длину таких столбцов. Нельзя делать их длину меньше длины индексируемого префикса. Это можно сделать, только предварительно удалив индекс и воссоздав его потом с коротким префиксом. Например, есть индекс по первым 30 байтам столбца типа CHAR, имеющего 40-байтовую длину. Впоследствии обнаружилось, что больше 20 байт в этом столбце вам никогда не потребуется, поэтому в целях экономии пространства можно уменьшить длину столбца до 20 символов. В этом случае, перед тем как уменьшить длину столбца, необходимо предварительно удалить индекс. После этого можно создать новый индекс, в котором будет использоваться 20 или менее байт. Столбцы в индексах типа FULLTEXT не могут иметь префиксов, и в случае объявления таковых они будут проигнорированы.

Удаление индексов

Удаление индексов производится оператором DROP INDEX *или* ALTER TABLE. Аналогично оператору CREATE INDEX, оператор DROP INDEX внутренне обрабатывается как оператор ALTER TABLE. Он впервые появился в СУБД MySQL версии 3.22 и не может использоваться для работы с индексом PRIMARY KEY. Операторы удаления индексов имеют следующий синтаксис:

```
DROP INDEX index_name ON tbl_name
ALTER TABLE tbl_name DROP INDEX index_name
ALTER TABLE tbl_name DROP PRIMARY KEY
```

Два первых оператора идентичны. Третий используется только для удаления индекса PRIMARY KEY. В последнем случае нужно указывать имя индекса, так как таблица может иметь только один индекс с таким именем. Если такой индекс создан не был, а таблица имеет один или несколько индексов UNIQUE, будет удален первый из них.

Удаление столбцов из таблицы влияет на индексы. Удаляя столбец из таблицы, вы тем самым удаляете этот столбец из индекса. Удаляя все индексируемые столбцы из таблицы, вы удаляете весь индекс.

Изменение структуры таблиц

Оператор ALTER TABLE в СУБД MySQL обладает достаточно универсальными возможностями. Мы уже демонстрировали несколько вариантов его использования (создание и удаление индексов, преобразование таблицы из одного формата в другой). В этом разделе будут продемонстрированы и другие "способности" этого оператора. Полный синтаксис оператора ALTER TABLE приведен в приложении Г, "Синтаксис SQL".

Оператор ALTER TABLE может пригодиться в том случае, когда становится очевидно, что старая структура таблицы не соответствует текущим задачам. Появилась какая-то дополнительная информация, или таблица содержит данные, ставшие уже избыточными. Существующие столбцы оказались слишком малы, или опыт работы с таблицей показал, что они непредусмотрительно были объявлены слишком большими и для экономии пространства и оптимизации запросов их следует уменьшить. Возможно, просто была допущена ошибка в имени столбца при создании таблицы с помощью оператора CREATE TABLE. Вот несколько таких ситуаций.

- Предположим, что вы участвуете в научном проекте. Каждому эксперименту вы присваиваете номер и сохраняете его в столбце с параметром AUTO_INCREMENT. По всем оценкам, исследования должны длиться достаточно долго. Количество регистрирующих записей должно составлять около 50000. Вы совершенно справедливо приняли решение о том, что для хранения номеров опытов следует выбрать тип UNSIGNED SMALLINT, который может хранить до 65535 значений. Однако по полученным результатам проект был признан успешным, и его финансирование возобновилось. По предварительным оценкам, может быть сделано еще 50000 записей. Необходимо изменить имеющийся тип столбца на более вместительный.
- Размеры столбцов могут изменяться и иначе. Предположим, был создан столбец типа CHAR(255), но в процессе работы с таблицей оказалось, что длина значений в этом столбце не превышает 100 символов. Длину столбца можно уменьшить.
- Возникла необходимость преобразовать тип таблицы в другой тип, чтобы воспользоваться возможностями, которые он предоставляет. Например, таблицы типа ISAM не позволяют иметь пустые значения в индексированных столбцах. При необходимости сделать это можно преобразовать таблицу типа ISAM в таблицу типа MyISAM.

Синтаксис оператора ALTER TABLE имеет следующий вид:

```
ALTER TABLE tbl_name action, ...
```

Спецификатором *action* определяется операция, которая будет произведена над таблицей. Некоторые СУБД позволяют выполнять с помощью оператора ALTER TABLE только одну операцию, в то время как СУБД MySQL позволяет выполнять посредством одного оператора ALTER TABLE сразу несколько операций, которые разделяются запятыми. Таким образом, в СУБД MySQL был расширен диапазон действий оператора ALTER TABLE. Это удобно и уменьшает объемы работ, необходимые для ввода с клавиатуры, но более веской причиной этого расширения является то, что невозможно заменить таблицу со строками переменной длины на таблицу со строками фиксированной длины, не изменив одновременно типы всех столбцов с типа VARCHAR на тип CHAR. Ниже приведены новые возможности оператора ALTER TABLE.

- **Переименование таблицы.** Это просто, достаточно указать новое и старое имя:

  ```
  ALTER TABLE tbl_name RENAME AS new_tbl_name;
  ```

  Еще один способ переименования таблиц заключается в использовании оператора RENAME TABLE, впервые появившегося в версии 3.23.23. Его синтаксис выглядит следующим образом:

  ```
  RENAME TABLE old_name TO new_name;
  ```

  Оператор RENAME TABLE отличается от оператора ALTER TABLE тем, что он позволяет переименовывать несколько таблиц в одном операторе. Например, можно поменять имена таблиц следующим образом:

  ```
  RENAME TABLE t1 TO tmp, t2 TO t1, tmp TO t1;
  ```

  Указав имя таблицы и имя базы данных, вы можете переносить таблицы из одной базы данных в другую. Любой из указанных ниже операторов перенесет таблицу t из базы данных sampdb в базу данных test.

  ```
  ALTER TABLE sampdb.t RENAME TO test.t;
  RENAME TABLE sampdb.t TO test.t;
  ```

  Однако нельзя переименовать таблицу, если новое имя совпадает с именем существующей таблицы.

- **Модификация типа столбца.** Тип столбца меняется с помощью предложения CHANGE или MODIFY в операторе ALTER TABLE. Предположим, что столбец в таблице mytbl имеет тип SMALLINT UNSIGNED и появилась необходимость изменить его на MEDUIM UNSIGNED. Это можно сделать с помощью одной из следующих команд:

  ```
  ALTER TABLE mytbl MODIFY i MEDIUMINT UNSIGNED;
  ALTER TABLE mytbl CHANGE i i MEDIUMINT UNSIGNED;
  ```

  Почему во втором случае столбец назван дважды? Потому, что операция CHANGE, в отличие от операции MODIFY, не только меняет тип данных, но и переименовывает столбец. В случае необходимости можно не только изменить тип данных столбца, но и изменить имя столбца с i, скажем, на j:

  ```
  ALTER TABLE mytbl CHANGE i j MEDIUMINT UNSIGNED;
  ```

Тут важно понять семантику этого оператора. Сначала указывается имя столбца, который будет меняться, а затем дается полностью новое объявление столбца с указанием его же имени. Имя указывается обязательно, даже если оно является старым именем столбца.

В версии 4.1 появилась возможность присваивать кодировки отдельным столбцам, что позволяет в объявлении столбца воспользоваться атрибутом CHARACTER SET для изменения его кодировки:

```
ALTER TABLE t MODIFY c CHAR(20) CHARACTER SET ucs2;
```

Существенной причиной для изменения типа столбца может служить необходимость оптимизировать объединения, сравнивающие столбцы из разных таблиц. Операция сравнения занимает меньше времени, если столбцы однотипные. Предположим, вы запустили запрос

```
SELECT ... FROM t1, t2 WHERE t1.name = t2.name;
```

Очевидно, что если столбец `t1.name` имеет тип `CHAR(10)`, а столбец `t2.name` – тип `CHAR(15)`, то запрос будет обрабатываться медленнее, чем в том случае, если бы оба столбца имели тип `CHAR(15)`. Изменим длину `t1.name` с помощью одного из следующих операторов:

```
ALTER TABLE t1 MODIFY name CHAR(15);
ALTER TABLE t1 CHANGE name name CHAR(15);
```

В версиях до 3.22 были возможны объединения только по столбцам одного и того же типа. В СУБД MySQL версии 3.22 и выше появилась возможность выполнять объединения по столбцам разного типа. Но эффективнее все же осуществляются объединения однотипных данных.

- **Преобразование таблиц с переменной длиной строки в таблицы с фиксированной длиной строки.** Предположим, что мы имеем таблицу `chartbl` со столбцами типа VARCHAR. Создадим такую таблицу:

  ```
  CREATE TABLE chartbl (name VARCHAR(40), address VARCHAR(80));
  ```

  Таблица содержит столбцы типа VARCHAR, и требуется преобразовать их в столбцы типа CHAR и проверить, к каким результатам это приведет. (Для таблиц, использующих формат хранения ISAM или MyISAM, строки с фиксированной длиной обрабатываются быстрее, чем строки с переменной длиной.) Сложность этой процедуры состоит в том, что изменять все столбцы нужно одновременно, используя один оператор ALTER TABLE. Если изменить только один или несколько столбцов, СУБД MySQL "решит", что таблица по-прежнему имеет строки переменной длины и преобразует CHAR-столбцы обратно в VARCHAR, чтобы сэкономить пространство. Такое изменение структуры можно осуществить только с помощью оператора вида

  ```
  ALTER TABLE chartbl MODIFY name CHAR(40), MODIFY address CHAR(80);
  ```

  Теперь запрос DESCRIBE покажет, что таблица содержит только столбцы CHAR. Это наглядно демонстрирует необходимость поддержки множественных операций с помощью оператора ALTER TABLE.

  Нужно предупредить, что при осуществлении таких операций следует быть очень осторожным: наличие столбцов типа BLOB или TEXT будет препятствовать преобразованию таблицы в таблицу с фиксированной длиной стро-

ки. Эти типы не имеют аналогичного типа фиксированной длины, а наличие хотя бы одного столбца переменной длины приводит к присвоению всем столбцам типа с переменной длиной.

- **Преобразование таблиц с фиксированной длиной строки в таблицы с переменной длиной строки.** Да, все запросы к таблице `chartbl` работают быстрее, когда она имеет строки фиксированной длины, но при этом занимает больше места, чем хотелось бы. Итак, мы решили вернуть ее в первоначальное состояние. Преобразование таблицы в этом направлении происходит значительно проще. Достаточно преобразовать тип только одного столбца в тип VARCHAR, и СУБД MySQL автоматически преобразует тип оставшихся столбцов из CHAR в VARCHAR:

```
ALTER TABLE chartbl MODIFY name VARCHAR(40);
ALTER TABLE chartbl MODIFY address VARCHAR(80);
```

- **Преобразование типа таблицы.** Для преобразования формата хранения таблицы можно воспользоваться предложением TYPE в операторе ALTER TABLE:

```
ALTER TABLE tbl_name TYPE = tbl_type;
```

где *tbl_type* — спецификация типа, которая может быть ISAM, MYISAM, HEAP, BDB или INNODB (регистр не имеет значения).

Изменение типа таблицы может помочь модифицировать работающую СУБД MySQL таким образом, чтобы извлечь всю выгоду от применения новых возможностей последних версий СУБД MySQL. Например, при переходе с версии ниже 3.23 на версию 3.23 или выше все ваши старые таблицы должны иметь формат ISAM. Для того чтобы преобразовать их в формат MyISAM, можно воспользоваться следующим оператором:

```
ALTER TABLE tbl_name TYPE = MYISAM;
```

Зачем это делать? Первая из причин была уже описана выше, в разделах "Типы таблиц" и "Индексирование таблиц". Этот формат лучше проявляет себя при индексировании и позволяет индексировать по столбцам, содержащим пустые значения, и по столбцам типа BLOB и TEXT. Второй причиной является то, что таблицы типа MyISAM машинно-независимы, поэтому их можно переносить прямым копированием, даже если машины имеют различную аппаратную конфигурацию.

Третья причина заключается в том, чтобы предоставить возможность работать с транзакциями. Предположим, что в вашем распоряжении имеется таблица типа MyISAM и обнаружилось, что требуется выполнять операции с транзакциями, включая откаты в случае возникновения сбоев. Таблицы типа MyISAM не поддерживают транзакций, но этого можно добиться, преобразовав их в формат BDB или InnoDB:

```
ALTER TABLE tbl_name TYPE = BDB;
ALTER TABLE tbl_name TYPE = INNODB;
```

Оператор ALTER TABLE имеет множество применений, но есть обстоятельства, при которых его не следует использовать. Вот два примера.

- Таблицы типа HEAP хранятся в памяти и исчезают при отключении от сервера. Поэтому не следует преобразовывать таблицу в этот тип, если эта таблица потребуется вам после отключения сервера.

- При использовании таблиц типа MERGE для объединения таблиц типа MyISAM следует избегать применения оператора ALTER TABLE для внесения изменений в структуру одной из составляющих таблиц MyISAM, если аналогичному изменению не подверглись все другие составляющие таблицы. Правильное функционирование таблицы MERGE зависит от подобия структур всех составляющих ее таблиц типа MyISAM.

Получение информации о базах данных и таблицах

СУБД MySQL имеет несколько разновидностей оператора SHOW, позволяющих отображать информацию о базах данных и размещенных в них таблицах. Оператор SHOW полезен для отслеживания структуры базы данных, особенно для предварительного просмотра свойств столбцов таблицы перед внесением изменений в их структуру с помощью оператора ALTER TABLE.

- Перечень баз данных на сервере:

```
SHOW DATABASES;
```

- Перечень таблиц в текущей или заданной базе данных:

```
SHOW TABLES;
SHOW TABLES FROM db_name;
```

  Оператор SHOW TABLES не позволяет просматривать структуру временных таблиц.

- Отображение информации о столбцах и индексах в указанной таблице:

```
SHOW COLUMNS FROM tbl_name;
SHOW INDEX FROM tbl_name;
```

  Операторы DESCRIBE *tbl_name* и EXPLAIN *tbl_name* являются синонимами оператора SHOW COLUMNS FROM *tbl_name*.

- Отображение описательной информации о таблицах в текущей и указанной базе данных:

```
SHOW TABLE STATUS;
SHOW TABLE STATUS FROM db_name;
```

  Этот оператор появился в СУБД MySQL 3.23.0.

- Отображение оператора CREATE TABLE, соответствующего текущей структуре таблицы:

```
SHOW CREATE TABLE tbl_name;
```

  Этот оператор появился в СУБД MySQL 3.23.20.

Некоторые форматы оператора SHOW могут работать с предложением LIKE 'pat', позволяющим задавать шаблон соответствия. Здесь 'pat' интерпретируется аналогично шаблону SQL, который может включать групповые символы '%' и '_'. Например, следующие операторы отображают имена таблиц в текущей базе данных, которые начинаются с 'geo':

```
SHOW TABLES LIKE 'geo%';
```

Для того чтобы осуществить поиск по групповому символу в шаблоне LIKE, его необходимо предварять символом обратной косой черты. Чаще всего это приходится делать для поиска символа '_', который очень часто применяется в именах баз данных, таблиц и столбцов.

Команда mysqlshow позволяет получить аналогичную информацию о базе данных и таблицах непосредственно из оболочки UNIX:

- перечень баз данных на сервере:

```
% mysqlshow
```

- перечень таблиц в указанной базе данных:

```
% mysqlshow db_name
```

- отображение информации о столбцах в указанной таблице:

```
% mysqlshow db_name tbl_name
```

- отображение информации об индексах в указанной таблице:

```
% mysqlshow -- keys db_name tbl_name
```

- отображение описательной информации о таблицах в указанной базе данных:

```
% mysqlshow -- status db_name
```

Программа mysqldump также позволяет просматривать структуру таблиц в формате оператора CREATE TABLE (подобно оператору SHOW CREATE TABLE). Но при использовании mysqldump вызывайте его с ключом -- no-data, чтобы случайно не получить дамп всей таблицы:

```
% mysqldump -- no-data db_name tbl_name
```

Без имени таблицы mysqldump отображает структуру всех таблиц, имеющихся в базе данных.

Для команд mysqldump и mysqlshow может понадобиться определить обычные параметры, такие, например, как -- host или -- user, для подключения к другому серверу.

Определение типов, поддерживаемых сервером

До версии 3.23 поддерживался только формат ISAM, в последующих версиях стали поддерживаться форматы MyISAM, MERGE и HEAP, а в наличии других типов можно убедиться с помощью соответствующего оператора SHOW VARIABLES:

```
SHOW VARIABLES LIKE 'have_isam';
SHOW VARIABLES LIKE 'have_bdb';
SHOW VARIABLES LIKE 'have_inno%';
```

Если результат запроса показывает, что переменная имеет значение YES, это значит, что поддержка соответствующего формата обеспечена. Если будет получено другое значение или вывод вообще отсутствует, то поддержка данного формата не предусмотрена. Для определения поддержки формата InnoDB можно воспользоваться шаблонами have_innodb и have_innobase. (Последняя форма использовалась в версиях MySQL от 3.23.30 до 3.23.36, а затем она была переименована в have_innodb.)

Информация о типе таблицы может использоваться при определении поддержки транзакций вашим сервером. Таблицы BDB и InnoDB представляют собой

два типа таблиц, поддерживающих транзакции, поэтому проверку активизации их поддержки можно осуществить, как было описано выше.

Начиная с версии 4.1 перечень типов таблиц можно получить непосредственно с помощью операторов SHOW TABLE TYPES:

```
mysql> SHOW TABLE TYPES;
+--------+---------+----------------------------------------------------+
| Type   | Support | Comment                                            |
+--------+---------+----------------------------------------------------+
| MyISAM | DEFAULT | Default type from 3.23 with great performance      |
| HEAP   | YES     | Hash based, stored in memory, useful for temporary |
|        |         | tables                                             |
| MERGE  | YES     | Collection of identical MyISAM tables              |
| ISAM   | YES     | Obsolete table type; Is replaced by MyISAM         |
| InnoDB | YES     | Supports transactions, row-level locking and foreign|
|        |         | keys                                               |
| BDB    | YES     | Supports transactions and page-level locking       |
+--------+---------+----------------------------------------------------+
```

Значение Support может быть равно YES или NO, что указывает на наличие поддержки или на ее отсутствие, DISABLED — если поддержка имеется, но она отключена, или DEFAULT — для типов таблиц, которые используются сервером по умолчанию. Дескриптор, обозначенный как DEFAULT, можно рассматривать как имеющийся в наличии.

Проверка существования таблиц или типов таблиц

Иногда может очень пригодиться возможность определять существование заданной таблицы непосредственно из приложения. Для этих целей предназначен оператор SHOW TABLES:

```
SHOW TABLES LIKE 'tbl_name';
SHOW TABLES FROM db_name LIKE 'tbl_name';
```

Если оператор SHOW выводит информацию о таблице, можно сказать, что она существует. Существование таблицы можно определить с помощью одного из следующих операторов:

```
SELECT COUNT(*) FROM tbl_name;
SELECT * FROM tbl_name WHERE 0;
```

Любой из этих операторов завершается успешно, если таблица существует, и неуспешно — если она не существует. Первый оператор лучше всего подходит для таблиц MyISAM и ISAM, для которых выражения COUNT(*) без предложения WHERE являются самыми оптимальными. (Это несправедливо для таблиц типа InnoDB или BDB, для подсчета количества строк которых требуется полное сканирование.) Второй оператор — более общего вида, так как он работает достаточно быстро для таблиц любого типа. Использование этих запросов больше всего подходит для использования в таких языках программирования, как Perl или PHP, поскольку это позволяет проверить успешность завершения запроса, чтобы соответствующим образом продолжить выполнение программ. От них мало пользы при использовании в пакетных сценариях, которые можно запускать из mysql, так как здесь ничего не остается делать, кроме завершения (или проигнорировать ошибку, но в таком случае, зачем вообще нужно было запускать выполнение запроса.)

Для определения типа таблицы начиная с версии 3.23.0 используется оператор SHOW TABLE STATUS и оператор SHOW CREATE TABLE — начиная с версии 3.23.20. Вывод обоих операторов включает индикатор типа. В версиях ниже 3.23.0 нет ни одного из этих операторов. Но единственным имеющимся там типом таблиц является тип ISAM, поэтому здесь не возникает неоднозначности в том, какой тип формата хранения используется.

Выборка записей из нескольких таблиц

Нет никакого смысла помещать записи в базу данных, если не выбирать их потом и не обрабатывать определенным образом. В этом и заключается основное предназначение оператора SELECT — получить и обработать данные. Оператор SELECT, вероятно, используется чаще других операторов языка SQL. Но он также может быть самым мощным оператором, а ограничения, которые используются вами при выборке строк, могут быть достаточно сложными, позволяющими производить сравнение значений столбцов в нескольких таблицах.

Основной синтаксис оператора SELECT имеет следующий вид:

```
SELECT selection_list # какие столбцы выбирать
FROM table_list # откуда их выбирать
WHERE primary_constraint # каким условиям должны удовлетворять строки
GROUP BY grouping_columns # как группируются результаты
ORDER BY sorting_columns # как сортируются результаты
HAVING secondary_constraint # вторичные условия, которым должны
удовлетворять строки
LIMIT count; # ограничение количества полученных результатов
```

В синтаксисе этого оператора обязательным являются только слова SELECT и *selection_list*, определяющие перечень выбираемых столбцов. Для некоторых баз данных слово FROM тоже является обязательным. О СУБД MySQL этого сказать нельзя. Здесь существует возможность вычислять выражения, не обращаясь к таблицам:

```
SELECT SQRT(POW(3,2)+POW(4,2));
```

В главе 1, "Знакомство с СУБД MySQL и SQL", при рассмотрении синтаксиса оператора SELECT акцент в основном делался на списках выборки, предложениях WHERE, GROUP BY, ORDER BY, HAVING и LIMIT. В этой главе мы сконцентрируем основное внимание на наиболее сложном аспекте оператора SELECT — на объединении, т.е. какие типы объединения поддерживает СУБД MySQL, что они делают и как задаются. Это помогает эффективнее использовать СУБД MySQL потому, что самой серьезной задачей при написании сложного запроса выборки является определение правильного варианта объединения таблиц. Далее в этой главе мы рассмотрим решения некоторых проблем, многие из которых требуют применения всех возможностей оператора SELECT.

Основной нюанс, с которым часто сталкиваются при разработке запросов SQL, заключается в том, что при возникновении проблемы не всегда легко найти ее решение. Но после того, как выход найден, его можно использовать для решения подобных проблем в будущем. Оператор SELECT как раз и является оператором такого типа. Для эффективной работы с ним накопленный опыт

играет существенную роль, поскольку все пути его эффективного использования вам известны.

По мере накопления опыта вам будет проще находить пути решения новых задач. Вы уже будете ловить себя на мысли, что такая задача просто решается с помощью конструкции LEFT JOIN или что это объединение ограничено общей парой ключевых столбцов. (Я, конечно, не беру на себя полной ответственности утверждать, что будет именно так. Но мысль о том, что опыт, накопленный в ходе практической работы, пригодится вам в будущем, будет вам утешительна.)

В трех последующих разделах мы рассмотрим использование всех форм объединений, которые поддерживает СУБД MySQL. В примерах будут использованы две небольшие таблицы.

```
Table t1:   Table t2:
+----+----+ +----+----+
| i1 | c1 | | i2 | c2 |
+----+----+ +----+----+
|  1 | a  | |  2 | c  |
|  2 | b  | |  3 | b  |
|  3 | c  | |  4 | a  |
+----+----+ +----+----+
```

Тривиальное объединение

Самым простым типом объединения является тривиальное объединение, в котором задействована только одна таблица. В этом случае строки выбираются из одной таблицы, указанной в предложении FROM.

```
mysql> SELECT * FROM t1;
+------+------+
| i1   | c1   |
+------+------+
|    1 | a    |
|    2 | b    |
|    3 | c    |
+------+------+
```

Некоторые авторы не считают такое объединение объединением вообще и применяют это понятие к выборкам из двух и более таблиц. Но я считаю такое объединение всего лишь частным случаем.

Полное объединение

Объединение будет полным, если в нем задействовано несколько таблиц. Например, при объединении двух таблиц каждая строка из первой таблицы объединяется с каждой строкой из второй таблицы.

```
mysql> SELECT t1.*, t2.* FROM t1, t2;
+------+------+------+------+
| i1   | c1   | i2   | c2   |
+------+------+------+------+
|    1 | a    |    2 | c    |
|    2 | b    |    2 | c    |
|    3 | c    |    2 | c    |
|    1 | a    |    3 | b    |
```

```
|    2 | b    |    3 | b    |
|    3 | c    |    3 | b    |
|    1 | a    |    4 | a    |
|    2 | b    |    4 | a    |
|    3 | c    |    4 | a    |
+------+------+------+------+
```

Полное объединение называют еще и перекрестным объединением. Здесь каждая строка одной таблицы объединяется с каждой строкой другой таблицы, создавая тем самым все возможные комбинации строк обеих таблиц. Есть еще одно название – декартово произведение. Такое объединение создает огромное количество строк, потому что результирующее количество строк будет произведением количества строк каждой таблицы, задействованной в объединении. Например, полное объединение трех таблиц, содержащих соответственно 100, 200 и 300 строк, в результате возвращает 100x200x300 = 6 миллионов строк. Несмотря на то что размеры исходных таблиц невелики, мы получим очень много строк. В таких случаях для уменьшения количества строк до приемлемого количества можно воспользоваться предложением WHERE.

При добавлении условия в предложении WHERE, требующем соответствия определенных значений определенных столбцов из двух таблиц, мы получим объединение типа equi–join (объединение по равенству).

```
mysql> SELECT t1.*, t2.* FROM t1, t2 WHERE t1.i1 = t2.i2;
+------+------+------+------+
| i1   | c1   | i2   | c2   |
+------+------+------+------+
|    2 | b    |    2 | c    |
|    3 | c    |    3 | b    |
+------+------+------+------+
```

Типы объединений JOIN и CROSS_JOIN эквивалентны оператору объединения ', '. Например, следующие операторы подобны:

```
SELECT t1.*, t2.* FROM t1, t2 WHERE t1.i1 = t2.i2;
SELECT t1.*, t2.* FROM t1 JOIN t2 WHERE t1.i1 = t2.i2;
SELECT t1.*, t2.* FROM t1 CROSS JOIN t2 WHERE t1.i1 = t2.i2;
```

Объединение STRAIGHT_JOIN аналогично полному объединению, но при этом таблицы объединяются в порядке, указанном в предложении FROM. Оптимизатор СУБД MySQL при объединении таблиц сортирует их в абсолютно произвольном порядке, чтобы строки выбирались как можно быстрее. Конечно, оптимизатор может ошибиться, но это вполне устранимо с помощью ключевого слова STRAIGHT_JOIN.

Слово STRAIGHT_JOIN может быть указано в операторе SELECT в двух местах. Его можно указать между словом SELECT и списком выбираемых столбцов или в предложении FROM. Первый способ окажет глобальное воздействие на все объединения в операторе SELECT:

```
SELECT STRAIGHT_JOIN ... FROM table1, table2, table3 ...
SELECT... FROM table1 STRAIGHT_JOIN table2 STRAIGHT_JOIN table3 ...
```

Квалификация ссылок на столбцы

Все ссылки на столбцы в операторе SELECT должны быть однозначно соотнесены с таблицами, указанными в предложении FROM. Когда в предложении FROM указана одна таблица, неоднозначность исключена. Когда в предложении FROM указано сразу несколько таблиц, но рассматриваемый столбец есть только в одной таблице, неоднозначность также исключена.

Но когда в предложении FROM указано несколько таблиц и рассматриваемый столбец присутствует сразу в нескольких таблицах, столбец должен быть квалифицирован только полностью с применением синтаксиса *tbl_name.col_name*. Это однозначно определит, какой таблице принадлежит указанный столбец. Пусть таблица mytbl1 содержит столбцы a и b, а таблица mytbl2 — b и c. Ссылки на столбцы a и c будут однозначными, а вот ссылки на b должны быть полностью квалифицированы mytbl1.b или mytbl2.b:

```
SELECT a, mytbl1.b, mytbl2.b, c FROM my_tbl1, mytbl2 ...;
```

В некоторых ситуациях для удаления неоднозначности квалификатора имени таблицы будет недостаточно. Такая ситуация возникает тогда, когда таблица используется в запросе несколько раз. В таком случае для полной квалификации столбца может понадобиться псевдоним. Псевдонимы можно присвоить всем экземплярам таблицы, задействованным в данном запросе. Теперь ссылка на столбцы таблиц будет иметь вид *alias_name.col_name*. Предлагаю вашему вниманию запрос, объединивший в себе два экземпляра одной и той же таблицы. Второму экземпляру таблицы присвоен псевдоним, что позволяет однозначно ссылаться на столбцы из этой таблицы:

```
SELECT mytbl.col1, m.col2 FROM mytbl, mytbl AS m WHERE mytbl.col1 > m.col1;
```

Левое и правое объединения

Полное объединение отображает только те строки, значения которых по критерию выборки полностью совпадают. Левое и правое объединения действуют аналогичным образом. Кроме того, они включают строки одной из таблиц, которые *не* имеют соответствия в другой таблице. В примерах этого раздела используется только левое объединение, в котором содержатся строки из левой таблицы, не удовлетворяющие критерию выборки. Правое объединение аналогично, но в этом случае таблицы меняются ролями. (Правое объединение появилось в версии 3.23.25.)

Оператор левой выборки работает следующим образом: вы задаете столбцы, которые будут использоваться для поиска соответствия в двух таблицах. Строки обеих таблиц выбираются в том случае, если критерии выборки совпадают; если критерии выборки не совпадают, строка левой таблицы выбирается, а правой присваиваются пустые значения. Другими словами, оператор выборки с левым объединением выбирает все строки левой таблицы, независимо от того, совпадают они по критерию выборки или нет. Строки, для которых соответствия не обнаружено, можно установить по пустым строкам справа.

Вернемся к нашим таблицам t1 и t2:

```
Table t1:   Table t2:
+----+----+ +----+----+
| i1 | c1 | | i2 | c2 |
+----+----+ +----+----+
|  1 | a  | |  2 | c  |
|  2 | b  | |  3 | b  |
|  3 | c  | |  4 | a  |
+----+----+ +----+----+
```

Если для объединения этих таблиц воспользоваться перекрестным объединением по столбцам t1.i1 и t2.i2, результат будет получен только для значений 2 и 3, так как только они присутствуют в обеих таблицах:

```
mysql> SELECT t1.*, t2.* FROM t1, t2 WHERE t1.i1 = t2.i2;
+------+------+------+------+
| i1   | c1   | i2   | c2   |
+------+------+------+------+
|    2 | b    |    2 | c    |
|    3 | c    |    3 | b    |
+------+------+------+------+
```

В результате левого объединения получим вывод, содержащий все строки из таблицы t1, независимо от того, найдены ли для них совпадения в таблице t2. Для того чтобы создать левое объединение, необходимо использовать выражение LEFT JOIN (вместо разделительной запятой) и задать условие соответствия с помощью выражения ON (а не предложения WHERE):

```
mysql> SELECT t1.*, t2.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2;
+------+------+------+------+
| i1   | c1   | i2   | c2   |
+------+------+------+------+
|    1 | a    | NULL | NULL |
|    2 | b    |    2 | c    |
|    3 | c    |    3 | b    |
+------+------+------+------+
```

Теперь появилась строка для значения 1 из таблицы t1, которое не имеет соответствия в таблице t2. Левое объединение очень полезно для выяснения, почему по данному критерию выборки не "подтягиваются" строки из правой таблицы. Такой запрос можно получить, добавив предложение WHERE, которое позволит просматривать только те строки выборки, в которых правая таблица представлена пустыми значениями.

В этой ситуации нужно следить, чтобы столбцы, по которым выполняется объединение, не были объявлены как NOT NULL, в противном случае может быть получен неадекватный результат.

```
mysql> SELECT t1.*, t2.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2 WHERE
t2.i2 IS NULL;
+------+------+------+------+
| i1   | c1   | i2   | c2   |
+------+------+------+------+
|    1 | a    | NULL | NULL |
+------+------+------+------+
```

Но в выборке пустых столбцов нет необходимости. Единственное, что действительно может интересовать при подобных выборках, — это строки из левой таблицы, не имеющие соответствия во второй таблице.

```
mysql> SELECT t1.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2
    -> WHERE t2.i2 IS NULL
+------+------+
| i1   | c1   |
+------+------+
|    1 | a    |
+------+------+
```

Объединение LEFT JOIN позволяет задавать условия соответствия двумя способами. Одним из них является ON, который может использоваться независимо от того, имеют ли столбцы, по которым производится объединение, одинаковые имена:

```
SELECT t1.*, t2.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2
```

Другим способом является использование предложения USING(). Оно во многом аналогично предложению ON, но имена столбцов, по которым производится объединение, здесь должны совпадать полностью. Например, следующий запрос объединяет таблицы по соответствию mytbl1.b и mytbl2.b:

```
SELECT mytbl1.*, mytbl2.* FROM mytbl1 LEFT JOIN mytbl12 USING(b);
```

Объединение LEFT JOIN имеет несколько синонимов и вариантов. Один из них — объединение LEFT OUTER JOIN. Существует также специальная нотация для ODBC (Open DataBase Connectivity) СУБД MySQL (где oj обозначает "outer join" — "внешнее объединение"):

```
oj tbl_name1 LEFT OUTER JOIN tbl_name2 ON join_expr }
```

Объединение NATURAL LEFT JOIN тоже ведет себя подобно объединению LEFT JOIN. Оно осуществляет левое объединение, выбирая по критерию совпадения все столбцы, которые имеют одинаковое имя в правой и левой таблицах. Здесь единственным отличием от левого объединения является то, что если объединяемые столбцы не объявлены как NOT NULL, результат может быть неоднозначным. Например, если правая таблица содержит столбцы с пустыми значениями, их нельзя будет отличить от пустых значений, которыми идентифицируются несоответствующие строки.

Как уже подчеркивалось, левое объединение может пригодиться для ответа на вопрос "Какие значения отсутствуют?" Когда необходимо узнать, какие значения, представленные в одной таблице, отсутствуют в другой, воспользуйтесь конструкцией LEFT JOIN и посмотрите, в каких строках для второй таблицы стоит значение NULL. Рассмотрим более сложный пример задачи такого типа, чем та, которая была показана для таблиц t1 и t2.

Для проекта "Учет успеваемости", впервые появившегося в нашем повествовании в главе 1, "Знакомство с СУБД MySQL и SQL", у нас есть таблица student со списком учащихся, таблица event со списком экзаменов и таблица score с оценками, выставленными учащимся за каждый экзамен. Однако как быть с учащимся, который пропустил день экзамена по болезни, чтобы назначить ему переэкзаменовку. Как найти эти недостающие записи, чтобы быть уверенным в том, кто требуется переэкзаменовка?

Задача заключается в определении того, какие учащиеся не имеют оценок по определенному экзамену. Иначе говоря, требуется найти, какие комбинации учащихся и событий отсутствуют в таблице score. Словосочетание "какие значения отсутствуют" должно быть своеобразным сигналом того, что требуется применение конструкции LEFT JOIN. Это объединение нельзя назвать таким же простым, как и в предыдущем случае, так как мы не просто ищем значения, которые не представлены в одном столбце, мы ищем сочетание двух столбцов. Это сочетание включает все комбинации учащийся/событие, которые представляет собой пересечение таблиц student и event:

```
FROM student, event
```

После этого необходимо взять результат этого объединения и выполнить объединение типа LEFT JOIN с таблицей score:

```
FROM student, event
   LEFT JOIN score ON student.student_id = score.student.id
                          AND event.event_id = score.event_id
```

Заметим, что предложение ON позволяет столбцам из таблицы score объединяться в соответствии с объединениями в различных таблицах. Вот что является ключом к решению этой задачи.

Объединение LEFT JOIN заставляет генерировать ряды для всех строк, полученных от пересечения таблиц student и event, даже если нет соответствую-

щей записи в таблице score. Результирующие строки для этих записей с недостающими записями по оценке можно обнаружить по тому, что их значение будет равно NULL. Выборку этих записей можно задать в выражении WHERE. При этом можно указать любой столбец из таблицы score, но, вероятнее всего, правильно будет указать столбец score:

```
WHERE score.score IS NULL
```

Отсортировать полученные результаты можно с помощью предложения ORDER BY. Наиболее естественной будет сортировка по событиям и учащимся *или по учащимся и событиям. Сначала зададим*

```
ORDER BY student.student_id, event.event_id
```

Теперь нам остается перечислить имена нужных нам столбцов. Вот окончательный вид нашего запроса:

```
SELECT
   student.name, student.student_id,
   event.date, event.event_id, event.type
FROM
   student, event
   LEFT JOIN score ON student.student_id = core.student_id
                          AND event.event_id = score.event_id
WHERE
   score.score IS NULL
ORDER BY
   student.student_id, event.event_id;
```

В результате работы запроса получим:

```
+-----------+------------+------------+----------+------+
| name      | student_id | date       | event_id | type |
+-----------+------------+------------+----------+------+
| Megan     |          1 | 2002-09-16 |        4 | Q    |
| Joseph    |          2 | 2002-09-03 |        1 | Q    |
| Katie     |          4 | 2002-09-23 |        5 | Q    |
| Devri     |         13 | 2002-09-03 |        1 | Q    |
| Devri     |         13 | 2002-10-01 |        6 | T    |
| Will      |         17 | 2002-09-16 |        4 | Q    |
| Avery     |         20 | 2002-09-06 |        2 | Q    |
| Gregory   |         23 | 2002-10-01 |        6 | T    |
| Sarah     |         24 | 2002-09-23 |        5 | Q    |
| Carter    |         27 | 2002-09-16 |        4 | Q    |
| Carter    |         27 | 2002-09-23 |        5 | Q    |
| Gabrielle |         29 | 2002-09-16 |        4 | Q    |
| Grace     |         30 | 2002-09-23 |        5 | Q    |
+-----------+------------+------------+----------+------+
```

Здесь есть один неприятный момент. В результате мы имеем идентификаторы учащихся и событий. Столбец student_id присутствует в таблицах student и score. Таким образом, можно допустить, что список выборки может содержать столбец student.student_id или score.student_id. Но в данном случае так делать нельзя, поскольку из таблицы score интересующие нас записи возвращаются как NULL. Выборка score.student_id даст нам столбец со значениями NULL. Аналогичный принцип лежит в основе принятия решения о том, из какой таблицы выбирать столбец event_id. Они присутствуют как в таблице event, так и в таблице score. Но значения score.event_id всегда будут равны NULL.

Использование вложенных выборок

Одной из особенностей версии 4.1 является то, что в ней появилась поддержка вложенных выборок. Потребность создания вложенных запросов SELECT в СУБД MySQL назрела уже довольно давно. Ниже приведен пример, в котором производится выборка идентификаторов событий, имеющих тип 'T', которые в свою очередь будут использоваться при выборке результатов экзаменов.

```
SELECT * FROM score
WHERE event_id IN (SELECT event_id FROM event WHERE type = 'T');
```

В некоторых случаях вложенные выборки можно записать как объединения, что и будет показано позднее. Возможно, это поможет вам, если вы работаете с версией СУБД MySQL, предшествующей версии 4.1

Подобная возможность позволяет удалять или модифицировать записи в одной таблице на основании данных, полученных из другой таблицы. Эти типы операций обсуждаются в этой главе в разделе "Удаление и обновление нескольких таблиц".

Существует несколько приемов создания вложенных выборок, но в данной главе будут рассмотрены лишь некоторые из них.

- **Использование вложенной выборки для получения ссылки.** В этом случае внутренний оператор SELECT используется для идентификации одного значения, которое будет использоваться в операциях сравнения внешнего оператора SELECT. Например, при определении результатов викторины, проведенной '2002-09-23', для определения идентификатора викторины можно воспользоваться внутренним оператором SELECT, а затем выбрать все оценки по этому идентификатору с помощью внешнего оператора SELECT:

  ```
  SELECT * FROM score
  WHERE event_id =
  (SELECT event_id FROM event WHERE date = '2002-09-23' AND type =
  'Q');
  ```

  В соответствии с этим применением вложенной выборки, где внутреннему запросу предшествует оператор сравнения, необходимо, чтобы в результате внутреннего запроса было получено не больше одного значения (т.е. одна строка и один столбец). Если будет получено несколько значений, оператор не сработает. (В некоторых случаях уместно обеспечить выполнение этого ограничения с помощью выражения LIMIT 1.)

  Эта форма вложенной выборки может пригодиться для использования в предложении WHERE агрегатной функции. Например, для того чтобы определить, какой президент родился раньше, необходимо задать следующее выражение:

  ```
  SELECT * FROM president WHERE birth = MIN(birth);
  ```

  Но этот оператор содержит ошибку: агрегированные функции нельзя применять в выражениях WHERE. (С помощью предложения WHERE запись выбирается, но функция MIN() не может быть известна до тех пор, пока значение не выбрано.) Однако выход есть. Этот оператор можно переписать следующим образом:

```
SELECT * FROM president
WHERE birth = (SELECT MIN(birth) FROM president);
```

- **Вложенные выборки EXISTS и NOT EXISTS.** Эта форма вложенной выборки работает с помощью передачи значений из внешнего запроса во внутренний для проверки, соответствуют ли они условиям, заданным во внутреннем запросе. Поэтому при малейшей вероятности возникновения какой-либо неоднозначности имена столбцов необходимо квалифицировать. Вложенные выборки типа EXISTS и NOT EXISTS полезны при поиске записей в таблице, записи которой соответствуют или не соответствуют записям в другой таблице.

  Обратимся еще раз к нашим таблицам t1 и t2.

```
Table t1:   Table t2:
+----+----+ +----+----+
| i1 | c1 | | i2 | c2 |
+----+----+ +----+----+
|  1 | a  | |  2 | c  |
|  2 | b  | |  3 | b  |
|  3 | c  | |  4 | a  |
+----+----+ +----+----+
```

  Следующий пример запроса определяет соответствие таблиц, т.е. производится выборка по значениям, которые присутствуют в обеих таблицах:

```
mysql> SELECT i1 FROM t1
    -> WHERE EXISTS (SELECT * FROM t2 WHERE t1.i1 = t2.i2);
+----+
| i1 |
+----+
|  2 |
|  3 |
+----+
```

  NOT EXISTS определяет несоответствия, или значения, присутствующие в одной таблице, но отсутствующие в другой:

```
mysql> SELECT i1 FROM t1
    -> WHERE NOT EXISTS (SELECT * FROM t2 WHERE t1.i1 = t2.i2);
+----+
| i1 |
+----+
|  1 |
+----+
```

  В такой вложенной выборке внутренний запрос использует в качестве списка выводимых столбцов *. Здесь нет необходимости объявлять столбцы явным образом, так как результат внутреннего запроса оценивается только как "истина" или "ложь" на основании того, возвращает или не возвращает он строки вообще, а не на основании какого-то конкретного возвращаемого значения. В СУБД MySQL в действительности можно написать самые различные имена в списке выборки, но необходимо четко понимать, что в результате вы получите значение "истина", если запрос SELECT будет успешным, поэтому можно написать запрос следующего типа:

```
SELECT i1 FROM t1
WHERE EXISTS (SELECT 1 FROM t2 WHERE t1.i1 = t2.i2);
```

```
SELECT i1 FROM t1
WHERE NOT EXISTS (SELECT 1 FROM t2 WHERE t1.i1 = t2.i2);
```

- **Вложенные выборки IN и NOT IN.** Формы IN и NOT IN выборки возвращают один столбец значений из внутреннего оператора SELECT, который будет оцениваться во внешнем операторе SELECT. Например, предыдущие выборки, в которых использовались конструкции EXISTS и NOT EXISTS, можно переписать с помощью синтаксиса IN и NOT IN:

```
mysql> SELECT i1 FROM t1 WHERE i1 IN (SELECT i2 FROM t2);
+----+
| i1 |
+----+
|  2 |
|  3 |
+----+
mysql> SELECT i1 FROM t1 WHERE i1 NOT IN (SELECT i2 FROM t2);
+----+
| i1 |
+----+
|  1 |
+----+
```

Как переписать вложенную выборку в виде объединения

В версиях, предшествующих 4.1, возможность создавать вложенные выборки отсутствовала. Однако существует возможность перезаписи запроса, в котором используются вложенные выборки, используя объединение. Даже если вы располагаете СУБД MySQL 4.1 и выше, будет нелишним проверить работу запросов, которые были написаны как вложенные выборки, с помощью запросов, использующих объединения, так как они часто работают эффективнее, чем вложенные выборки.

Перезапись вложенных выборок, выбирающих совпадающие значения

Ниже приведен пример запроса, в котором используется вложенная выборка. Ее задачей является выборка оценок за экзамены из таблицы score (т.е. они игнорируют оценки по викторинам):

```
SELECT * FROM score
WHERE event_id IN (SELECT event_id FROM event WHERE type = 'T');
```

Аналогичный запрос можно написать, не пользуясь вложенной выборкой, а выполнив простое объединение:

```
SELECT score.* FROM score, event
WHERE score.event_id = event.event_id AND event.type = 'T';
```

Вот другой пример выборки оценок, полученных только девочками:

```
SELECT * from score
WHERE student_id IN (SELECT student_id FROM student WHERE sex = 'F');
```

Его можно преобразовать в такое объединение:

```
SELECT score.* FROM score, student
WHERE score.student_id = student.student_id AND student.sex = 'F';
```

Можно создать такой шаблон для выборки, использующей вложенную выборку:

```
SELECT * FROM table1
WHERE column1 IN (SELECT column2a FROM table2 WHERE column2b = value);
```

Запросы такого типа можно преобразовать в запросы, использующие объединение такого вида:

```
SELECT table1.* FROM table1, table2
WHERE table1.column1 = table2.column2a AND table2.column2b = value;
```

Перезапись вложенных выборок, которые выбирают несовпадающие (отсутствующие) значения

Другим общим типом запроса, который использует вложенные выборки, является запрос, ищущий значения в одной таблице по отсутствию их в другой. Как было показано выше, тип задачи, который можно сформулировать как "найти значения, которые отсутствуют", является сигналом для применения объединений LEFT JOIN.

Вот пример запроса с вложенной выборкой, которая проверяет экзамены учащихся, *не* упоминавшихся в таблице absence (т.е. найти учащихся с отличной посещаемостью):

```
SELECT * FROM student
WHERE student_id NOT IN (SELECT student_id FROM absence);
```

Этот запрос можно переписать с помощью конструкции LEFT JOIN:

```
SELECT student.*
FROM student LEFT JOIN absence ON student.student_id =
absence.student_id
WHERE absence.student_id IS NULL;
```

В общих выражениях форму с вложенной выборкой можно сформулировать так:

```
SELECT * FROM table1
WHERE column1 NOT IN (SELECT column2 FROM table2);
```

Этот запрос можно переписать следующим образом:

```
SELECT table1.*
FROM table1 LEFT JOIN table2 ON table1.column1 = table2.column2
WHERE table2.column2 IS NULL;
```

При этом предполагается, что *table2.column2* объявлена как NOT NULL.

Выборка из нескольких таблиц с помощью объединения UNION

Запрос по выборке данных из нескольких таблиц, имеющих одинаковую структуру, во многих базах данных осуществляется оператором UNION. В СУБД MySQL этот оператор впервые появился в версии 4, хотя прежде его отсутствие можно было компенсировать различными обходными приемами (о них мы поговорим позднее).

Для примера воспользуемся тремя таблицами, t1, t2 и t3, которые имеют следующий вид:

```
mysql> SELECT * FROM t1;
+------+-------+
| i    | c     |
+------+-------+
|    1 | red   |
|    2 | blue  |
|    3 | green |
+------+-------+
```

```
mysql> SELECT * FROM t2;
+------+------+
| i    | c    |
+------+------+
|   -1 | tan  |
|    1 | red  |
+------+------+
mysql> SELECT * FROM t3;
+------------+------+
| d          | i    |
+------------+------+
| 1904-01-01 |  100 |
| 2004-01-01 |  200 |
| 2004-01-01 |  200 |
+------------+------+
```

Таблицы t1 и t2 имеют столбцы целого и символьного типа, а таблица t3 имеет целые столбцы и столбцы календарного типа. Для того чтобы создать оператор UNION, который объединяет несколько выборок, достаточно написать несколько отдельных операторов SELECT и поместить ключевое слово UNION между ними. Например, для выборки целого столбца из всех таблиц понадобится такой оператор:

```
mysql> SELECT i FROM t1 UNION SELECT i FROM t2 UNION SELECT i FROM t3;
+------+
| i    |
+------+
|    1 |
|    2 |
|    3 |
|   -1 |
|  100 |
|  200 |
+------+
```

Оператор UNION имеет следующие свойства.

- Имена и типы данных столбцов для результирующего набора в результате работы оператора UNION устанавливаются по имени и типу первого оператора SELECT. Второй и последующий операторы SELECT в объединении UNION должны иметь одинаковое количество столбцов, но им совсем не обязательно иметь один и тот же тип или имя. Столбцы подбираются по порядку следования, а не по имени. Вот почему эти два запроса возвращают различные результаты:

```
mysql> SELECT i, c FROM t1 UNION SELECT i, d FROM t3;
+------+------------+
| i    | c          |
+------+------------+
|    1 | red        |
|    2 | blue       |
|    3 | green      |
|  100 | 1904-01-01 |
|  200 | 2004-01-01 |
+------+------------+
mysql> SELECT i, c FROM t1 UNION SELECT d, i FROM t3;
```

```
+------+-------+
| i    | c     |
+------+-------+
|    1 | red   |
|    2 | blue  |
|    3 | green |
| 1904 | 100   |
| 2004 | 200   |
+------+-------+
```

В обоих случаях столбцы, выбранные из таблицы `t1` (`i` и `c`), задают типы, полученные в результате работы оператора `UNION`. Это столбцы целого и символьного типа, поэтому при выборке значений из таблицы `t3` происходит преобразование. В первом запросе значение столбца `d` преобразуется из даты в строку, чтобы не произошло потери информации. Во втором запросе `d` преобразуется из даты в целое (при этом *происходит* потеря информации), а `i` преобразуется из целого числа в строку.

- По умолчанию оператор `UNION` в результирующем наборе не содержит повторяющихся строк:

```
mysql> SELECT * FROM t1 UNION SELECT * FROM t2 UNION SELECT *
FROM t3;
+------+-------+
| i    | c     |
+------+-------+
|    1 | red   |
|    2 | blue  |
|    3 | green |
|   -1 | tan   |
| 1904 | 100   |
| 2004 | 200   |
+------+-------+
```

Таблицы `t1` и `t2` имеют строки, содержащие значения `1` и `'red'`, но такая строка в результирующем наборе появляется только однажды. Кроме того, таблица `t3` имеет в своем составе строки `'2004-01-01'` и `200`, одна из которых была отброшена.

Для того чтобы появились повторения, необходимо после первого ключевого слова `UNION` указать также ключевое слово `ALL`:

```
mysql> SELECT * FROM t1 UNION ALL SELECT * FROM t2 UNION SELECT *
FROM t3;
+------+-------+
| i    | c     |
+------+-------+
|    1 | red   |
|    2 | blue  |
|    3 | green |
|   -1 | tan   |
|    1 | red   |
| 1904 | 100   |
| 2004 | 200   |
| 2004 | 200   |
+------+-------+
```

- Для того чтобы отсортировать результирующий набор, добавьте в последний оператор `SELECT` предложение `ORDER BY`; оно будет относиться к результату запроса в целом. Однако, вследствие того, что оператор `UNION` ис-

пользует имена столбцов из первого оператора SELECT, предложение ORDER BY будет иметь отношение к этим именам, а не к именам из последнего оператора SELECT, даже если они различаются.

```
mysql> SELECT i, c FROM t1 UNION SELECT i, d FROM t3
    -> ORDER BY c;
+------+------------+
| i    | c          |
+------+------------+
|  100 | 1904-01-01 |
|  200 | 2004-01-01 |
|    2 | blue       |
|    3 | green      |
|    1 | red        |
+------+------------+
```

Кроме того, предложение ORDER BY может задаваться для отдельного оператора SELECT в объединении UNION. Для этого необходимо заключить отдельный оператор SELECT (включая и ORDER BY, имеющий к нему отношение) в скобки:

```
mysql> (SELECT i, c FROM t1 ORDER BY i DESC)
    -> UNION (SELECT i, c FROM t2 ORDER BY i);
+------+-------+
| i    | c     |
+------+-------+
|    3 | green |
|    2 | blue  |
|    1 | red   |
|   -1 | tan   |
+------+-------+
```

Подобно предложению ORDER BY, в пределах объединения UNION может использоваться предложение LIMIT. При добавлении в конец оператора он относится к результирующему набору данных:

```
mysql> SELECT * FROM t1 UNION SELECT * FROM t2 UNION SELECT * FROM t3
    -> LIMIT 1;
+------+------+
| i    | c    |
+------+------+
|    1 | red  |
+------+------+
```

Заключив его в скобки как часть отдельного оператора SELECT, вы ограничите область его действия этим оператором SELECT:

```
mysql> (SELECT * FROM t1 LIMIT 1)
    -> UNION (SELECT * FROM t2 LIMIT 1)
    -> UNION (SELECT * FROM t3 LIMIT 1);
+------+------+
| i    | c    |
+------+------+
|    1 | red  |
|   -1 | tan  |
| 1904 | 100  |
+------+------+
```

- Выборку различных наборов, удовлетворяющих разным условиям, можно осуществлять и из одной таблицы. Это может пригодиться в качестве альтернати-

вы отработке нескольких разных выборок SELECT, так как объединение позволяет получить все результирующие строки в одном результирующем наборе.

До версии 4 оператор UNION отсутствовал, но эту проблему легко можно было обойти. Для этого достаточно было произвести выборку из каждой таблицы во временную таблицу, а затем сделать выборку из временной таблицы. Таким образом, можно отсортировать строки в нужном порядке. В СУБД MySQL версии 3.23 и выше эта проблема решается довольно просто благодаря тому, что сервер позволяет легко создавать временные таблицы. Такие таблицы автоматически уничтожаются после завершения текущего сеанса. В примере, представленном ниже, таблица удаляется автоматически после завершения вашего сеанса. Для ускорения быстродействия используется таблица в формате HEAP (хранящаяся в памяти).

```
CREATE TEMPORARY TABLE tmp TYPE = HEAP SELECT ... FROM t1 WHERE ... ;
INSERT INTO tmp SELECT ... FROM t2 WHERE ... ;
INSERT INTO tmp SELECT ... FROM t3 WHERE ... ;
...
SELECT * FROM tmp ORDER BY ... ;
```

Поскольку таблица tmp объявлена как TEMPORARY, сервер удалит ее автоматически после завершения сеанса работы клиента. (Конечно же, таблицу можно удалить и явным образом, как только работа с ней завершится, и это позволит освободить на сервере ресурсы, задействованные ею. Это будет очень кстати, если другие запросы продолжают выполняться, особенно для таблиц типа HEAP.)

Подобная идея может быть реализована в версиях до 3.23. Вся разница заключается в том, что столбцы таблицы tmp необходимо объявлять явным образом, так как таблицы типа TEMPORARY еще не существуют и напрочь отсутствует формат HEAP (имеется один-единственный тип таблиц – тип ISAM). Для того чтобы эта таблица прекратила свое существование после завершения сеанса работы, в конце всей процедуры необходимо задать оператор DROP TABLE.

```
CREATE TABLE tmp (column1, column2, ...);
INSERT INTO tmp SELECT ... FROM t1 WHERE ... ;
INSERT INTO tmp SELECT ... FROM t2 WHERE ... ;
INSERT INTO tmp SELECT ... FROM t3 WHERE ... ;
SELECT * FROM tmp ORDER BY ... ;
DROP TABLE tmp;
```

Если необходимо создать запросы UNION к таблицам типа MyISAM, имеющим одинаковую структуру, в качестве обходного приема можно воспользоваться таблицей типа MERGE. (Действительно, этот прием может оказаться как нельзя кстати, даже если в вашем распоряжении есть оператор UNION, так как таблица MERGE упрощает процедуру создания запросов.) Запрос к таблице типа MERGE проще аналогичного запроса, использующего конструкцию UNION к таблицам, составляющим таблицы MERGE. Запрос SELECT по таблице типа MERGE работает как UNION ALL (повторяющиеся строки не удаляются), а SELECT DISTINCT работает как UNION (повторяющиеся строки удаляются).

Удаление и обновление нескольких таблиц

До СУБД MySQL версии 4 единственное ограничение оператора DELETE заключалось в том, что ссылаться можно было только на таблицу, из которой удаля-

лись записи. Но иногда полезнее другой механизм, позволяющий удалять определенные записи, опираясь на их соответствие или несоответствие записям в другой таблице. Эта возможность была добавлена в версии 4.0.0. Аналогичная возможность обновления определенных записей на основании анализа содержимого записей в другой таблице была добавлена в версии 4.0.2. В этой главе описано, каким образом осуществляются операции DELETE и UPDATE в нескольких таблицах. Этот тип операторов опирается на концепцию, используемую в объединениях, поэтому для восприятия данного материала необходимо ознакомиться с разделом "Выборка записей из нескольких таблиц".

Если при выполнении операции DELETE или UPDATE ссылка делается на столбцы только из одной таблицы, то квалифицировать каким-то образом имена столбцов не требуется. Например, чтобы удалить все записи в таблице `t`, значения столбца `id` в которых больше 100, необходимо создать такой оператор:

```
DELETE FROM t WHERE id > 100;
```

Но как быть, когда требуется удалить записи, не опираясь на свойства, присущие самим записям, а опираясь на значения в связанных с ними записях в другой таблице? Например, предположим, что требуется удалить из таблицы `t` записи с такими значениями `id`, которые одновременно присутствуют и в таблице `t2`?

Для того чтобы написать оператор DELETE по нескольким таблицам, их необходимо перечислить в предложении FROM и предложением WHERE задать условия соответствия записей в таблицах. Этот оператор удаляет записи из таблицы `t1`, имеющие соответствующие значения столбца `id` в таблице `t2`:

```
DELETE t1 FROM t1, t2 WHERE t1.id = t2.id;
```

Обратите внимание, что предложение FROM дает перечисление всех таблиц, участвующих в операции удаления, аналогично тому, как это делается при объединениях. Кроме того, если имя столбца появляется более чем в одной таблице, оно становится неоднозначным и должно сопровождаться именем таблицы. По стилю это напоминает объединения.

Синтаксис также позволяет удалять записи из нескольких таблиц одновременно. Для того чтобы удалить строки из *обеих* таблиц, там где обнаружено соответствие значений столбцов `id`, необходимо указать их имена после ключевого слова DELETE:

```
DELETE t1, t2 FROM t1, t2 WHERE t1.id = t2.id;
```

А что требуется для удаления несоответствующих записей? Можно прибегнуть к той же стратегии, которая используется при создании оператора SELECT, отбирающего несоответствующие записи, т.е. можно воспользоваться конструкцией LEFT JOIN или RIGHT JOIN. Например, идентифицировать строки в таблице `t1`, которые не имеют соответствия в таблице `t2`, можно с помощью следующего оператора SELECT:

```
SELECT t1.* FROM t1 LEFT JOIN t2 ON t1.id = t2.id WHERE t2.id IS NULL;
```

Аналогичный оператор DELETE, предназначенный для поиска и выборки этих записей из таблицы `t1`, также использует конструкцию LEFT JOIN:

```
DELETE t1 FROM t1 LEFT JOIN t2 ON t1.id = t2.id WHERE t2.id IS NULL;
```

Немного другой синтаксис операции DELETE по нескольким таблицам поддерживается начиная с версии 4.0.2. В соответствии с ним, для того, чтобы показать, из каких таблиц будет производиться удаление, используется предложение

FROM, а также предложение USING, задающее удаляемые записи. Предыдущие операторы DELETE можно переписать следующим образом:

```
DELETE FROM t1 USING t1, t2 WHERE t1.id = t2.id;
DELETE FROM t1, t2 USING t1, t2 WHERE t1.id = t2.id;
DELETE FROM t1 USING t1 LEFT JOIN t2 ON t1.id = t2.id WHERE t2.id IS NULL;
```

Другой тип оператора DELETE, предназначенный для удаления из нескольких таблиц, можно получить при использовании связей, установленных через внешние ключи, которые создаются с помощью ограничения ON DELETE CASCADE. Более подробно об этом речь пойдет ниже, в разделе "Внешние ключи и ссылочная целостность".

Принципы работы операторов UPDATE для нескольких таблиц аналогичны выполнению операций DELETE: перечислите все таблицы, участвующие в операции, и квалифицируйте их столбцы. Предположим, что викторина, которая проводилась 23 сентября 2002 года, содержала вопрос, на который все дали неправильный ответ. Так случилось потому, что в подсказке содержалась ошибка. Поэтому к полученной всеми оценке необходимо добавить очки. Не имея возможности обновления по нескольким таблицам, такую операцию можно выполнить в несколько действий. Сначала выберем идентификатор события для викторины, состоявшейся в определенное время:

```
SELECT @id := event_id FROM event WHERE date = '2002-09-23' AND type =
'Q';
```

После этого воспользуемся полученным идентификатором события для выявления соответствующих записей:

```
UPDATE score SET score = score + 1 WHERE event_id = @id;
```

При выполнении операции UPDATE на нескольких таблицах все это можно сделать с помощью одного оператора:

```
UPDATE score, event SET score.score = score.score + 1
WHERE score.event_id = event.event_id
AND event.date = '2002-09-23' AND event.type = 'Q';
```

При этом есть возможность идентифицировать записи для их модификации. Основываясь на содержимом другой таблицы, вы можете копировать значения, хранящиеся в таблицах, из одной таблицы в другую. Следующий оператор копирует столбец t1.a в t2.a для записей, которые имеют соответствующие значения в столбцах id:

```
UPDATE t1, t2 SET t2.a = t1.a WHERE t2.id = t1.id;
```

Выполнение транзакций

Транзакция – это последовательность операторов SQL, которые будут выполняться как единая операция, не прерываемая другими клиентами. Одним из предназначений транзакции является обеспечение гарантии неприкосновенности данных со стороны других клиентов во время вашей работы с ними. СУБД MySQL автоматически блокирует отдельные операторы SQL, чтобы клиенты не могли влиять на работу друг друга. (Например, два клиента не должны иметь возможности модифицировать одну и ту же запись одновременно.) Но автоматиче-

ского блокирования одного оператора не всегда будет достаточно для гарантии того, что операция с базой данных сможет решит возложенную на нее задачу. Некоторые операции можно осуществить только с помощью нескольких операторов. В этом случае различные операции могут оказывать взаимное влияние друг на друга. Транзакции позволяют группировать операторы в единую исполняемую единицу, что позволяет решать проблемы параллельной работы, которые всегда возникают в среде, где работает много клиентов.

Поддержка транзакций также включает поддержку операций выполнения (commit) и операций отката (rollback), которые и определяют группу операторов, выполняющихся как единое целое.

Другими словами, если транзакция выполняется успешно, то это является гарантией того, что все операции внутри группы будут выполнены успешно. Если же часть транзакции выполняется со сбоем, результаты выполнения всех операторов транзакций до места сбоя отменяются, оставляя базу данных в том виде, в котором она была до выполнения транзакции.

Системы, поддерживающие возможность транзакций, часто еще характеризуются как обеспечивающие свойства ACID. (ACID обозначает Atomic, Consistent, Isolated, Durable (атомарный, целостный, изолированный, длительный) и соответствует четырем свойствам, которые должна иметь транзакция.)

- **Атомарность.** Операторы, составляющие транзакцию, формируют единый логический блок. Ни один из этих операторов нельзя выполнять отдельно.
- **Целостность.** База данных является целостной до и после выполнения транзакции. Другими словами, транзакция не вносит путаницы в базу данных.
- **Изоляция.** Одна транзакция никак не влияет на другую.
- **Длительность.** При выполнении транзакции все ее результаты сохраняются в базе данных.

Одни форматы таблиц СУБД MySQL не поддерживают транзакций (ISAM, MyISAM и HEAP), а другие поддерживают (BDB и InnoDB). Данный раздел посвящен проблемам, возникающим, когда обеспечению транзакций не уделяется должного внимания, и тому, как их можно решить с применением транзакций и без них.

Для чего нужны транзакции

Посмотрим, какую пользу можно извлечь от применения механизма транзакций, на примере ведения дел в магазине готовой одежды. В складские запасы изменения вносятся в тот момент, когда продавцы продают товары. Ниже приведен пример, демонстрирующий ситуацию, возникающую при продаже одного и того же товара несколькими продавцами и одновременной модификации ими базы данных. Предположим, что текущий остаток на складе составляет 47 рубашек.

1. Продавец А продает три рубашки и регистрирует продажу. Регистрирующее программное обеспечение начинает модификацию содержимого базы данных с выборки текущего состояния счетчика остатка рубашек на складе (47):
   ```
   SELECT quantity FROM inventory WHERE item = 'shirt';
   ```

2. В это же время продавец В продает две рубашки и тоже регистрирует свою продажу. Программное обеспечение второго кассового аппарата также начинает модификацию базы данных:
   ```
   SELECT quantity FROM inventory WHERE item = 'shirt';
   ```
3. Первый кассовый аппарат вычисляет новый остаток, 47–3 = 44, и в соответствии с этим модифицирует остаток рубашек:
   ```
   UPDATE inventory SET quantity = 44 WHERE item = 'shirt';
   ```
4. Второй кассовый аппарат вычисляет новый остаток, 47–2 = 45, и тоже модифицирует остаток рубашек:
   ```
   UPDATE inventory SET quantity = 45 WHERE item = 'shirt';
   ```

Отлично. В результате этой последовательности событий было продано пять рубашек. Но получен остаток, равный 45 рубашкам, а не 42. Проблема заключается в том, что если остаток на складе просматривается в одном операторе, а модифицируется в другом, то мы имеем дело с транзакцией, включающей несколько операторов. Действие, выполненное во втором операторе, связано с результатом, полученным в первом операторе. Если отдельные транзакции выполняются в одно и то же время, то операторы из одной транзакции влияют на операторы из другой транзакции. Вот почему в СУБД, поддерживающих транзакции, запросы продавцов будут действовать как транзакции, и запрос продавца В не сможет отработать до тех пор, пока не отработает запрос продавца А.

Другая ситуация, которая может возникнуть при обработке базы данных с помощью нескольких операций, заключается в том, что при возникновении ошибки на полпути база данных останется в наполовину модифицированном состоянии (нецелостном). Типичным примером такого случая может служить финансовая операция по переводу денег с одного счета на другой. Предположим, что Билл подписал чек Бобу на 100 долларов и Боб хочет получить деньги по этому чеку. Счет Билла должен уменьшиться на 100 долларов, а счет Боба увеличиться на аналогичную сумму:

```
UPDATE account SET balance = balance - 100 WHERE name = 'Bill';
UPDATE account SET balance = balance + 100 WHERE name = 'Bob';
```

Если в момент завершения первого запроса и перед самым выполнением второго произойдет аварийный сбой, транзакция окажется незавершенной. В СУБД, где транзакции не поддерживаются, остается возможность проверить состояние операций, которые выполнялись в момент сбоя, и отменить их или довести операции до конца вручную. СУБД с возможностью отработки транзакций и отката будет в состоянии справиться с этой ситуацией. (По крайней мере теоретически. Можно будет определить, какая из транзакций не отработала, и повторить ее, не беспокоясь о незавершенности транзакции.)

Решение проблемы транзакций без применения механизма транзакций

В среде, где не поддерживаются транзакции, некоторые задачи обеспечения транзакций разрешимы, а некоторые — нет. Рассмотрим, что можно сделать без транзакций, а что сделать никогда не удастся. Еще раз рассмотрим сценарий учета продаж ру-

башек, описанный выше. Справиться с проблемами параллельной работы, которые присущи этой задаче, можно с помощью следующих приемов.

- **Заблокировать таблицы явным образом.** Операторы можно сгруппировать и выполнить как единое целое, объединив их операторами LOCK TABLES и UNLOCK TABLES. Все необходимые для операции таблицы блокируются, выполняются необходимые действия, после чего блокировка снимается. В результате никто не сможет работать с таблицами, пока они заблокированы. При использовании блокировки таблиц ситуация со складским остатком рубашек будет выглядеть следующим образом.

  1. Продавец А продает три рубашки и регистрирует продажу. Регистрирующее программное обеспечение начинает модифицировать содержимое базы данных с выборки текущего состояния счетчика остатка рубашек на складе (47):

     ```
     LOCK TABLES inventory WRITE;
     SELECT quantity FROM inventory WHERE item = 'shirt';
     ```

     Блокирование с параметром WRITE здесь необходимо из-за того, что конечной целью этой операции является модификация таблицы учета складских остатков, при которой используется запись в нее.

  2. В это время продавец В тоже продает две рубашки и регистрирует свою продажу. Программное обеспечение второго кассового аппарата также начинает модифицировать базу данных, для чего сначала ее необходимо заблокировать:

     ```
     LOCK TABLES inventory WRITE;
     ```

     В этом случае заблокировать таблицу не удается потому, что продавец А уже заблокировал таблицу.

  3. Продавец А вычисляет новый складской остаток как 47 – 3 = 44, записывает его и разблокирует таблицу:

     ```
     UPDATE inventory SET quantity = 44 WHERE item = 'shirt';
     UNLOCK TABLES;
     ```

  4. Теперь вступает в силу запрос на блокировку от продавца В, и он получает новый текущий остаток на складе (44 рубашки):

     ```
     SELECT quantity FROM inventory WHERE item = 'shirt';
     ```

  5. Второй кассовый аппарат вычисляет новый остаток как 44 – 2 = 42, записывает его и разблокирует таблицу:

     ```
     UPDATE inventory SET quantity = 42 WHERE item = 'shirt';
     UNLOCK TABLES;
     ```

     В этом случае оба оператора, выполняющие обе операции, не пересекаются, и остаток рубашек на складе вычисляется правильно.

     При работе с несколькими таблицами перед выполнением групповых запросов их все необходимо заблокировать. Однако в случае чтения из определенной таблицы необходимо использовать блокировку READ. (Это позволит другим пользователям читать из этих таблиц, но не позволит им производить запись в них.) Предположим, что в таблице inventory блок запросов производит некие изменения, а в таблице customer — ограничивается операциями чтения. В этом случае таблица inventory блокируется блокировкой WRITE, таблица customer блокируется блокировкой READ:

```
LOCK TABLES inventory WRITE, customer READ;
... здесь осуществляется работа с таблицами..
UNLOCK TABLES;
```

- **Производить не абсолютные, а относительные изменения.** Для метода внесения изменений, при котором используется явное блокирование таблиц, эта операция предполагает просмотр текущего остатка с помощью одного оператора, вычисление нового значения на основании количества проданных рубашек и запоминание нового складского остатка с помощью другого оператора. Вот второй вариант решения, обеспечивающий исключение взаимного влияния пользователей при транзакциях, содержащих много операторов. Такой метод не всегда может быть реализован, но мы проиллюстрируем его опять на примере складского остатка продаваемых рубашек. Так, первый вариант основан на выяснении текущего складского остатка, вычислении нового с учетом проданных рубашек и внесения изменений в складской остаток на основании полученного результата. Для этого можно выполнить только одно действие, изменив остаток рубашек относительно их текущего количества.

1. Продавец А продает три рубашки и регистрирует уменьшение складского остатка на три единицы:
   ```
   UPDATE inventory SET quantity = quantity - 3 WHERE item = 'shirt';
   ```
2. Продавец В продает две рубашки и регистрирует уменьшение складского остатка на две единицы:
   ```
   UPDATE inventory SET quantity = quantity - 2 WHERE item = 'shirt';
   ```
   При таком подходе к решению видно, что для выполнения модификации базы данных не требуется несколько операторов, а это в свою очередь снимает проблему параллельной работы.

Таким образом, вполне можно обойтись без явных операторов блокирования и без транзакций вообще. Такие подходы, не предусматривающие транзакций, могут успешно применяться при решении самых различных задач, но они имеют определенные ограничения.

- Не все операции можно описать в терминах относительных обновлений. Иногда просто *не* обойтись без применения последовательности нескольких операторов, и в этом случае обязательно должны приниматься во внимание проблемы параллельной работы.
- Необходимо оградить клиентов от взаимного влияния. Для этого таблицы блокируются на время выполнения операции, но что происходит, если где-то посередине операции случается сбой или ошибка? В таком случае необходимо отменить результаты работы первых операторов из этой группы, отработавших до возникновения сбойной ситуации, с тем чтобы база не осталась в наполовину модифицированном или в нецелостном состоянии.

  К сожалению, несмотря на то, что блокирование таблиц позволяет решать проблемы параллельной работы, оно оказывается совершенно беспомощным при восстановлении после ошибок.

- Стратегия блокирования требует от вас самостоятельного блокирования и разблокирования таблиц. При внесении изменений в набор модифицируемых таблиц необходимо соответствующим образом менять оператор LOCK TABLES. СУБД, имеющая поддержку транзакций, сама решает, какие блокировки ей необходимы, и активизирует их автоматически.

Возможности транзакций помогают решать все эти задачи. Обработчик транзакций выполняет группу операторов как единый блок и решает проблемы параллельной работы таким образом, что клиенты при этом не мешают друг другу. Они также позволяют выполнять откат в случае сбоя, что предохраняет базу данных от ситуаций, при которых операции остаются наполовину выполненными, и автоматически блокируют таблицы везде, где это необходимо.

Использование транзакций для обеспечения безопасного выполнения операций

Для того чтобы получить возможность использования транзакций, необходимо использовать таблицы того типа, который поддерживает работу с транзакциями. Таблицы типа ISAM, MyISAM и HEAP для этих целей совершенно не подходят, но можно воспользоваться таблицами типа BDB или InnoDB, которые впервые появились в двоичных дистрибутивах для версий 3.23.12 и 3.23.29 соответственно и в дистрибутивах с исходными текстами начиная с версии 3.23.34. Однако лучше всего будет воспользоваться самыми последними дистрибутивами. Это позволит вам воспользоваться всеми преимуществами последних нововведений. Когда нет полной уверенности в поддержке вашим сервером форматов BDB и InnoDB, обратитесь к разделу "Определение типов, поддерживаемых сервером" этой главы.

По умолчанию СУБД MySQL работает в режиме `auto-commit` (автовыполнения), т.е. изменения, которые вносятся отдельными операторами, производятся с базой данных немедленно. Это означает, что каждый оператор как бы выполняет свою собственную транзакцию. Для того чтобы выполнять транзакции явным образом, необходимо отключить режим автовыполнения и задать операции выполнения (commit) и отката (rollback) явно.

Одним из способов выполнения транзакции является задание оператора BEGIN для отключения режима автовыполнения, выполнения операторов, составляющих транзакцию, и завершение транзакции с помощью оператора COMMIT, чтобы зафиксировать операции. Если во время операции происходит сбой, ее результаты можно отменить с помощью оператора ROLLBACK. Оператор BEGIN приостанавливает режим автовыполнения. А после выполнения или отката транзакции режим восстанавливается таким, каким он был до выполнения оператора BEGIN. (Если был активизирован режим автовыполнения, после завершения транзакции он восстанавливается, если он был отключен, после завершения одной транзакции начинается другая.)

Этот подход можно проиллюстрировать на следующем примере. Создадим таблицу:

```
mysql> CREATE TABLE t (name CHAR(20), UNIQUE (name)) TYPE = INNODB;
```

Этот оператор создает таблицу InnoDB. Обозначим начало транзакции оператором BEGIN, добавим в эту таблицу несколько строк, выполним транзакцию и посмотрим, что из этого получится:

```
mysql> BEGIN;
mysql> INSERT INTO t SET name = 'William';
mysql> INSERT INTO t SET name = 'Wallace';
mysql> COMMIT;
mysql> SELECT * FROM t;
+---------+
| name    |
+---------+
| Wallace |
| William |
+---------+
```

Строки добавились в таблицу. Работу транзакции легко проверить следующим образом: запустите для этого другой экземпляр `mysql`, выборка с его помощью содержимого таблицы `t` после отработки оператора `INSERT` и до отработки оператора `COMMIT` ничего не покажет. Новые добавляемые в таблицу строки нельзя увидеть с помощью других работающих экземпляров `mysql` до тех пор, пока в первом процессе `mysql` не отработает оператор `COMMIT`.

Если во время отработки транзакции происходит сбой, ее работу можно отменить посредством оператора `ROLLBACK`. Воспользуемся таблицей `t` еще раз:

```
mysql> BEGIN;
mysql> INSERT INTO t SET name = 'Gromit';
mysql> INSERT INTO t SET name = 'Wallace';
ERROR 1062: Duplicate entry 'Wallace' for key 1
ОШИБКА 1062: Дублирование записи 'Wallace' для ключа 1
mysql> ROLLBACK;
mysql> SELECT * FROM t;
+---------+
| name    |
+---------+
| Wallace |
| William |
+---------+
```

Здесь второй оператор `INSERT` пытается добавить в таблицу строку, в которой дублируется уже существующее значение в столбце `name`. Эта операция дает сбой, так как столбец `name` входит в состав индекса `UNIQUE`. После запуска команды `ROLLBACK` в таблице останутся те же две строки, которые и были в ней до сбойной транзакции. В данном случае результат работы операторов `INSERT`, который расположен перед местом сбоя, отменяется и не сохраняется в таблице.

Указание оператора `BEGIN` во время выполнения транзакции является командой завершения текущей транзакции и немедленного начала следующей транзакции.

Существует еще один способ выполнения транзакций — это манипулирование режимом автовыполнения с помощью оператора `SET`:

```
SET AUTOCOMMIT = 0;
SET AUTOCOMMIT = 1;
```

После установки значения параметра `AUTOCOMMIT`, равным нулю, режим автовыполнения отключается. Результат любого последующего оператора является частью текущей транзакции, которая должна завершаться оператором `COMMIT` или `ROLLBACK`, чтобы выполнить транзакцию или отменить результаты ее выполнения. В соответствии с этим методом, режим автовыполнения будет отклю-

чен до тех пор, пока он не будет включен явным образом. Таким образом, завершение одной транзакции будет означать автоматическое начало другой. Выполнить транзакцию можно, включив режим автовыполнения. Как работает этот метод, покажем на предыдущем примере:

```
mysql> DROP TABLE t;
mysql> CREATE TABLE t (name CHAR(20), UNIQUE (name)) TYPE = INNODB;
```

Отключим режим автовыполнения, добавим несколько строк и выполним транзакцию:

```
mysql> SET AUTOCOMMIT = 0;
mysql> INSERT INTO t SET name = 'William';
mysql> INSERT INTO t SET name = 'Wallace';
mysql> COMMIT;
mysql> SELECT * FROM t;
+----------+
| name     |
+----------+
| Wallace  |
| William  |
+----------+
```

Начиная с этой точки в таблицу были добавлены две записи, но режим автовыполнения оставался отключенным. Добавление новых операторов сделает их частью новой транзакции, которая может быть выполнена или отменена независимо от первой транзакции. Для того чтобы вновь убедиться в том, что режим автовыполнения отключен и оператор ROLLBACK отменяет результат работы всех невыполненных операторов, выполним следующие запросы:

```
mysql> INSERT INTO t SET name = 'Gromit';
mysql> INSERT INTO t SET name = 'Wallace';
ERROR 1062: Duplicate entry 'Wallace' for key 1
ОШИБКА 1062: Дублирование записи 'Wallace' для ключа 1
mysql> ROLLBACK;
mysql> SELECT * FROM t;
+----------+
| name     |
+----------+
| Wallace  |
| William  |
+----------+
```

Для восстановления режима автовыполнения можно воспользоваться следующим оператором:

```
SET AUTOCOMMIT = 1;
```

Транзакции выполняются также при следующих условиях.

- Кроме операторов SET AUTOCOMMIT, BEGIN, COMMIT и ROLLBACK, которые управляют транзакциями явным образом, есть еще ряд других операторов, которые позволяют управлять выполнением транзакций неявным образом, так как сами они не могут быть частью транзакций. Запуск одного из перечисленных ниже операторов во время выполнения транзакции вынуждает сервер сначала выполнить транзакцию, а потом этот оператор.

```
ALTER TABLE
CREATE INDEX
DROP DATABASE
DROP INDEX
DROP TABLE
LOAD MASTER DATA
LOCK TABLES
RENAME TABLE TRUNCATE TABLE
UNLOCK TABLES (если таблица в данный момент заблокирована)
```

- Если соединение клиента завершается или разрывается до выполнения транзакции, сервер автоматически делает откат такой транзакции.

Транзакции могут пригодиться во многих ситуациях. Предположим, что вы работаете с таблицей score, которая является частью проекта "Учет успеваемости", и обнаружили, что оценки двух учащихся перемешались и их необходимо поменять местами. Вот неверно введенные оценки:

```
mysql> SELECT * FROM score WHERE event_id = 5 AND student_id IN (8,9);
+------------+----------+-------+
| student_id | event_id | score |
+------------+----------+-------+
|          8 |        5 |    18 |
|          9 |        5 |    13 |
+------------+----------+-------+
```

Для того чтобы исправить замеченную ошибку, учащемуся с идентификатором 8 должна быть присвоена оценка 13, а учащемуся 9 — 18. Это можно сделать с помощью двух операторов:

```
UPDATE score SET score = 13 WHERE event_id = 5 AND student_id = 8;
UPDATE score SET score = 18 WHERE event_id = 5 AND student_id = 9;
```

При этом необходимо убедиться в том, что оба оператора выполняются одним блоком. Для решения этой задачи можно прибегнуть к традиционным решениям. С помощью оператора BEGIN можно выполнить следующее:

```
mysql> BEGIN;
mysql> UPDATE score SET score = 13 WHERE event_id = 5 AND student_id = 8;
mysql> UPDATE score SET score = 18 WHERE event_id = 5 AND student_id = 9;
mysql> COMMIT;
```

То же самое можно сделать, манипулируя режимом автовыполнения:

```
mysql> SET AUTOCOMMIT = 0;
mysql> UPDATE score SET score = 13 WHERE event_id = 5 AND student_id = 8;
mysql> UPDATE score SET score = 18 WHERE event_id = 5 AND student_id = 9;
mysql> COMMIT;
mysql> SET AUTOCOMMIT = 1;
```

И в обоих случаях произойдет обмен оценками:

```
mysql> SELECT * FROM score WHERE event_id = 5 AND student_id IN (8,9);
+------------+----------+-------+
| student_id | event_id | score |
+------------+----------+-------+
|          8 |        5 |    13 |
|          9 |        5 |    18 |
+------------+----------+-------+
```

Внешние ключи и ссылочная целостность

Внешний ключ позволяет объявить зависимость ключа одной таблицы от ключа другой таблицы, а ссылочная целостность позволяет наложить ограничения на операции над таблицами, имеющими внешний ключ. Например, таблица `score` базы данных `sampdb` имеет столбец `student_id`, который можно использовать для связывания данных по результатам с результатами из таблицы `student`. В базах данных, поддерживающих концепцию внешних ключей, столбец `score.student_id` можно назвать внешним ключом. Это позволит наложить на этот внешний ключ в таблице `score` следующее ограничение: в таблицу не может быть добавлен результат учащегося, идентификатор которого отсутствует в таблице `student`. (Другими словами, внешний ключ не разрешает вводить оценки несуществующих учащихся.) Кроме того, появляется возможность *удаления каскадного типа*, т.е. при удалении учащегося из таблицы `student` будут удалены все записи с его результатами из таблицы `score`.

Внешние ключи обеспечивают целостность данных и удобны для работы с базами данных. Без внешних ключей вся ответственность за отслеживание взаимосвязей между таблицами и соблюдение их целостности ложится на разработчиков программных приложений. В большинстве случаев эта работа не требует больших затрат и заключается в добавлении нескольких операторов DELETE, чтобы при удалении записи из таблицы удалялись соответствующие ей записи из связанных с нею таблиц. Но если между таблицами установлены достаточно сложные связи, программист может отказаться нести ответственность за отслеживание всех связей в разрабатываемых им приложениях. И кроме того, почему бы не возложить эти заботы на систему управления базой данных?

Поддержка внешних ключей в СУБД MySQL обеспечивается дескриптором таблиц InnoDB. В этом разделе описывается, как задаются внешние ключи для данного типа таблиц и как они влияют на использование таблиц. Но сначала необходимо разобраться с терминами.

- Порождающей является таблица, содержащая оригинальные значения ключей.
- Порожденной является таблица, которая ссылается на ключевые значения в порождающей таблице.
- Ключевые значения из порождающей таблицы используются для связывания двух таблиц. В частности, индекс в порожденной таблице ссылается на индекс порождающей таблицы. Его значения должны соответствовать значениям в порождающей таблице или иметь значение NULL, чтобы показать, что в порождающей таблице нет связанной записи. Такой индекс порожденной таблицы называется внешним ключом, т.е. ключ является внешним для порождающей таблицы и содержит значения, указывающие на порождающую таблицу. Связь, созданная внешним ключом, может быть установлена в режиме запрещения значений NULL, в таком случае значения внешнего ключа должны точно соответствовать значениям из порождающей таблицы.

Формат InnoDB обеспечивает соблюдение всех этих правил, позволяющих соблюдать отношения внешних ключей. Это называется *ссылочной целостностью*.

Синтаксис объявления внешнего ключа в порожденной таблице имеет следующий вид, где необязательные элементы указываются в квадратных скобках:

```
FOREIGN KEY [index_name] (index_columns)
REFERENCES tbl_name (index_columns)
[ON DELETE action]
[ON UPDATE action]
[MATCH FULL | MATCH PARTIAL]
```

Необходимо отметить, что, несмотря на то, что все части этого синтаксиса анализируются, формат InnoDB не использует семантику всех предложений. Так, не поддерживаются и игнорируются предложения ON UPDATE и MATCH.[1] Обращать внимание следует только на следующие части этого оператора.

- Предложение FOREIGN KEY задает столбцы, составляющие индекс в порожденной таблице, которые должны соответствовать значению индекса в порождающей таблице. Заданное имя индекса *index_name* будет проигнорировано.
- Предложение REFERENCES объявляет порождающую таблицу и столбцы, содержащиеся в индексе, которые имеют отношение к внешнему ключу в порожденной таблице. Часть предложения *index_columns* должна иметь количество столбцов, равное количеству столбцов, перечисленных в *index_columns* в предложении FOREIGN KEY.
- Предложение ON DELETE позволяет задавать действия, выполняемые с порожденной таблицей при удалении записей в порождающей. Допустимы следующие действия.
  - ON DELETE CASCADE позволяет удалять записи в порожденной таблице при удалении соответствующих записей в порождающей таблице. В этом случае результат удаления передается по каскаду, начиная от порождающей таблицы и заканчивая порожденной. Это позволяет выполнять удаление сразу в нескольких таблицах при удалении строк только из одной порождающей таблицы.
  - ON DELETE SET NULL при удалении порождающей записи присваивает индексным столбцам в соответствующих порожденных записях значение NULL.

При использовании этой возможности все столбцы порожденной таблицы, перечисленные в объявлении внешнего ключа, должны объявляться с возможностью присвоения им пустых значений. (Единственное затруднение, с которым можно столкнуться при объявлении внешних ключей, заключается в том, что внешний ключ не может быть объявлен как PRIMARY KEY, поскольку индексы такого типа не могут хранить пустых значений.)

При объявлении внешних ключей необходимо придерживаться следующих правил.

- Порожденная таблица имеет индекс, в котором столбцы, входящие во внешний ключ, задаются первыми. Порождающая таблица также имеет индекс, столбцы которого перечислены в начале предложения REFERENCES.

---

[1] Для таблиц другого типа все объявления FOREIGN KEY игнорируются, но это не является ошибкой.

(Другими словами, столбцы, находящиеся в ключе, должны быть проиндексированы в обеих связанных с помощью внешних ключей таблицах.) Эти индексы должны быть заданы явным образом в порождающих и порожденных таблицах. Таблицы формата InnoDB такой возможности не имеют.

- Соответствующие столбцы в порождающих и порожденных индексах должны иметь сравнимые типы. Например, нельзя устанавливать соответствие между столбцами типа INT и CHAR. Соответствующие столбцы должны иметь одну длину. Соответствующие целочисленные столбцы должны иметь одинаковый размер и должны быть одновременно SIGNED или UNSIGNED.

Посмотрим на конкретном примере, как это работает. Начнем с создания таблиц parent и child, причем таблица child имеет внешний ключ, который ссылается на столбец par_id из таблицы parent:

```
CREATE TABLE parent
(
  par_id INT NOT NULL,
  PRIMARY KEY (par_id)
) TYPE = INNODB;
CREATE TABLE child
(
  par_id INT NOT NULL,
  child_id INT NOT NULL,
  PRIMARY KEY (par_id, child_id),
  FOREIGN KEY (par_id) REFERENCES parent (par_id) ON  DELETE CASCADE
) TYPE = INNODB;
```

В этом примере внешний ключ объявляется как ON DELETE CASCADE. Это говорит о том, что удаление какой-либо записи из таблицы parent будет означать автоматическое удаление записей с соответствующими значениями столбца par_id из таблицы child.

Теперь добавим новые записи в таблицу parent и добавим новые записи с соответствующими ключевыми значениями в таблицу child:

```
mysql> INSERT INTO parent (par_id) VALUES(1),(2),(3);
mysql> INSERT INTO child (par_id,child_id) VALUES(1,1),(1,2);
mysql> INSERT INTO child (par_id,child_id) VALUES(2,1),(2,2),(2,3);
mysql> INSERT INTO child (par_id,child_id) VALUES(3,1);
```

В результате отработки этих операторов получим следующее содержимое таблиц, где каждое значение поля par_id из таблицы child имеет соответствующее значение столбца par_id из таблицы parent:

```
mysql> SELECT * FROM parent;
+--------+
| par_id |
+--------+
|      1 |
|      2 |
|      3 |
+--------+
mysql> SELECT * FROM child;
+--------+----------+
| par_id | child_id |
+--------+----------+
```

```
|       1 |         1 |
|       1 |         2 |
|       2 |         1 |
|       2 |         2 |
|       2 |         3 |
|       3 |         1 |
+---------+-----------+
```

Для того чтобы проверить, что таблицы InnoDB проверяют связь между таблицами, добавим в таблицу child запись, не имеющую соответствия в столбце par_id таблицы parent:

```
mysql> INSERT INTO child (par_id,child_id) VALUES(4,1);
ERROR 1216: Cannot add a child row: a foreign key constraint fails
ОШИБКА 1216: Невозможно добавить порожденную строку: ограничение по
внешнему ключу не позволяет это сделать
```

А теперь посмотрим, что происходит при удалении записи:

```
mysql> DELETE FROM parent where par_id = 1;
```

СУБД MySQL удаляет запись из таблицы parent:

```
mysql> SELECT * FROM parent;
+---------+
| par_id |
+---------+
|       2 |
|       3 |
+---------+
```

Но при этом каскадно удаляются соответствующие записи в таблице child:

```
mysql> SELECT * FROM child;
+---------+-----------+
| par_id | child_id |
+---------+-----------+
|       2 |         1 |
|       2 |         2 |
|       2 |         3 |
|       3 |         1 |
+---------+-----------+
```

В этом примере наглядно показано, как организовывается каскадное удаление записей в порожденных таблицах при удалении записи в порождающей таблице. Существует и другая возможность, при которой порожденные записи сохраняются в таблице, но столбцы, соответствующие внешнему ключу, устанавливаются в значение NULL. Для этого в объявление таблицы child необходимо внести следующие изменения.

- Вместо ON DELETE CASCADE задается ON DELETE SET NULL. Это указывает таблице типа InnoDB вместо удаления записи устанавливать столбец, входящий во внешний ключ (par_id), равным NULL.
- Первоначально столбец par_id таблицы child объявлен как NOT NULL. Это, конечно, никак не совместимо с объявлением ON DELETE SET NULL, поэтому столбцы должны объявляться как NULL.
- Первоначально в объявлении таблицы child столбец par_id также объявляется как часть индекса PRIMARY KEY. Однако индекс PRIMARY KEY не

может содержать пустых значений NULL. Поэтому изменение объявления столбца par_id таким образом, чтобы он мог содержать NULL, также потребует замены индекса PRIMARY KEY индексом UNIQUE. Индексы UNIQUE в таблицах типа InnoDB обеспечивают уникальность для всех значений, кроме пустых, которые могут дублироваться в индексе.

Чтобы посмотреть, к чему приведут все эти нововведения, вернемся к таблице parent, использующей первоначальное объявление, и загрузим в нее те же первоначальные объявления. После этого создадим таблицу child с соблюдением новых условий:

```
CREATE TABLE child
(
  par_id INT NULL,
  child_id INT NOT NULL,
  UNIQUE (par_id, child_id),
  FOREIGN KEY (par_id) REFERENCES parent (par_id) ON  DELETE SET NULL
) TYPE = INNODB;
```

С точки зрения добавления новых записей порожденная таблица ведет себя по-прежнему, т.е. она позволяет вставлять записи со значениями столбца par_id, которые имеются в порождающей таблице, но блокирует добавление значений, которые в ней отсутствуют:[2]

```
mysql> INSERT INTO child (par_id,child_id) VALUES(1,1),(1,2);
mysql> INSERT INTO child (par_id,child_id) VALUES(2,1),(2,2),(2,3);
mysql> INSERT INTO child (par_id,child_id) VALUES(3,1);
mysql> INSERT INTO child (par_id,child_id) VALUES(4,1);
ERROR 1216: Cannot add a child row: a foreign key constraint fails
ОШИБКА 1216: Невозможно добавить порожденную строку: ограничение по
внешнему ключу не позволяет это сделать
```

Различие в поведении проявляется при удалении порождающей записи. Попробуйте удалить порождающую запись, а затем проверьте содержимое таблицы child:

```
mysql> DELETE FROM parent where par_id = 1;
mysql> SELECT * FROM child;
+--------+----------+
| par_id | child_id |
+--------+----------+
|   NULL |        1 |
|   NULL |        2 |
|      2 |        1 |
|      2 |        2 |
|      2 |        3 |
|      3 |        1 |
+--------+----------+
```

В этом случае порожденные записи, которые содержат в столбце par_id значения, равные 1, не удаляются. Столбцу par_id присваивается пустое значение, заданное в соответствии с ограничением ON DELETE SET NULL.

---

[2] В действительности относительно процедуры добавления записей имеет место одна особенность. Поскольку столбец par_id сейчас объявлен как NULL, можно явным образом вставить в порожденную таблицу записи, содержащие пустые значения, и при этом никаких ошибок не произойдет.

Различные возможности работы с внешними ключами появились не одновременно, что и отражает следующая таблица. Первоначальная поддержка внешних ключей не допускает добавления или удаления порожденных записей, нарушающих ограничения, накладываемые на ключи. Другие возможности появились позднее.

Возможность	Версия
Базовая поддержка внешних ключей	3.23.44
`ON DELETE CASCADE`	3.23.50
`ON DELETE SET NULL`	3.23.50

Из этой таблицы хорошо видно, что лучше всего воспользоваться возможностями версии 3.23.50, чтобы получить максимальную поддержку механизма внешних ключей. Другая причина того, почему необходимо использовать новейшие версии, заключается в том, что до версии 3.23.50 не были решены следующие проблемы.

- Использование оператора `ALTER TABLE` или `CREATE INDEX` при изменении таблиц InnoDB, участвующих в связях внешних ключей в роли потомка или в роли родителя, довольно опасно. При использовании этих операторов удаляются ограничения, установленные внешними ключами.
- Оператор `SHOW CREATE TABLE` не позволяет просматривать установку внешних ключей. Это же относится и к программе `mysqldump`, для которой представляется довольно затруднительным корректно восстановить архивные таблицы с внешними ключами.

Как обходиться без внешних ключей

Что можно предпринять для того, чтобы обеспечить целостность связей между вашими таблицами, если ваша версия СУБД MySQL не поддерживает формат InnoDB (а следовательно, внешние ключи)?

Обычно ограничения, накладываемые внешними ключами, нетрудно запрограммировать в приложениях. Это можно сделать на этапе ввода данных. Например, очень маловероятно, что при вводе новых записей в таблицу `score` будут введены данные о результатах несуществующих учащихся. Естественно начинать ввод результатов со списка учащихся, формирующегося из таблицы `student`. В соответствии с этой процедурой невозможно ввести данные об учащемся, имя которого отсутствует в списке.

Эффекта, подобного каскадному удалению, можно добиться и программным путем. Предположим, что удаляется запись об учащемся номер 13. Это значит, что также должны быть удалены все оценки данного учащегося. В базе данных с поддержкой каскадного удаления удаление записи из таблицы `student` удалит также все записи, соответствующие оценкам данного учащегося, из таблицы `score`:

```
DELETE FROM student WHERE student_id = 13
```

Без механизма внешних ключей придется удалять все логически связанные записи из соответствующих таблиц явным образом. Это позволит воспроизвести эффект каскадного удаления:

```
DELETE FROM student WHERE student_id = 13;
DELETE FROM score WHERE student_id = 13;
```

Другой способ выполнения этой операции появился в версии 4 и заключается в использовании приема множественного удаления, имеющего эффект, аналогичный каскадному удалению. Но будьте осторожны, так как здесь можно попасть в своеобразную ловушку. Следующий оператор, с первого взгляда выполняющий эту операцию, содержит ошибку:

```
DELETE student, score FROM student, score
WHERE student.student_id = 13 AND student.student_id =
score.student_id;
```

Загвоздка заключается в том, что этот оператор не сработает в случае, когда учащийся вообще не имеет никакой оценки, и предложение WHERE не найдет никакого соответствия и, таким образом, не удалит ни одной записи из таблицы student. В этом случае будет более уместна операция LEFT JOIN, так как она обнаружит запись в таблице student, даже при отсутствии любой соответствующей записи в таблице score:

```
DELETE student, score FROM student LEFT JOIN score USING (student_id)
WHERE student.student_id = 13;
```

Полнотекстный поиск типа FULLTEXT

В версии СУБД MySQL 3.23.23 появилась возможность выполнения полнотекстного поиска, благодаря которому при поиске слов и фраз нет необходимости применять шаблоны. Эта возможность активизируется для таблицы с помощью создания специального вида индекса, который имеет следующие характеристики.

- Полнотекстный поиск опирается на индексы FULLTEXT, которые создаются только для таблиц типа MyISAM и только для столбцов типа TEXT, не-BINARY CHAR и VARCHAR.
- Полнотекстный поиск не чувствителен к регистру. Это следует из типов столбцов, для которых могут быть использованы индексы типа FULLTEXT.
- При полнотекстном поиске общеупотребительные слова игнорируются. Причем "общеупотребительный" здесь обозначает "встречающийся по крайней мере в половине записей". Это особенно важно помнить тогда, когда вы экспериментируете с возможностью полнотекстного поиска на тестовой таблице. (Обязательно введите в свою таблицу не меньше трех записей, если в ней содержится одна или две записи, а каждое слово "встречается по меньшей мере в 50% слов", и вы не получите вообще никаких результатов!) Отдельные наиболее общеупотребительные слова, такие как "the", "after" и "other", и слишком короткие слова всегда игнорируются. По умолчанию "слишком коротким" является слово, которое имеет длину меньше четырех символов, но в скором будущем это значение можно будет уменьшить.
- Слова здесь определяются как последовательности символов, включающие буквы, цифры, одинарные кавычки и символ подчеркивания. Это значит, что такая строка, как "full-blooded", будет рассматриваться как

строка, содержащая два слова — "full" и "blooded." Обычно полнотекстный поиск определяет соответствие целых слов, а не частей слов, и считает, что запись соответствует полной строке, если она включает любое слово из искомой строки. Существует еще один вариант полнотекстного поиска, который называется логическим полнотекстным поиском и позволяет накладывать дополнительные ограничения, которым должны удовлетворять все слова (в произвольном порядке или для поиска фраз точно в соответствии с порядком, приведенным в строке поиска). Логический поиск позволяет производить поиск записей, которые *не* содержат определенных слов, или добавлять групповой символ для поиска всех слов, соответствующих всем словам, начинающимся с заданного префикса.

- Индекс типа FULLTEXT может создаваться для одного или нескольких столбцов. Если в нем задействовано сразу несколько столбцов, поиск, опирающийся на индекс, просматривает одновременно все столбцы. Обратной стороной такого подхода является то, что во время поиска необходимо задавать список столбцов, точно соответствующий набору столбцов, которые соответствуют определенному индексу FULLTEXT. Например, если время от времени необходимо просматривать столбец col1, иногда col2, а иногда сразу и col1, и col2, необходимо иметь в распоряжении три индекса: по два для каждого столбца в отдельности и один, который включает сразу два столбца.

Эти возможности появились не одновременно, одни — раньше, а другие — позднее. В следующей таблице приведены номера версий, в которых появились определенные возможности полнотекстного поиска.

Возможность	Версия
Базовый полнотекстный поиск	3.23.23
Настраиваемые параметры	4.0.0
Логический поиск	4.0.1
Поиск фраз	4.0.2

В следующих примерах показано использование полнотекстного поиска. Для этого сначала создается индекс FULLTEXT, а затем выполняется поиск с использованием оператора MATCH.

Индекс FULLTEXT аналогичен другим индексам, т.е. его можно задать при создании таблицы с помощью оператора CREATE TABLE или создать с помощью оператора ALTER TABLE или CREATE INDEX позднее. Поскольку операторы FULLTEXT требуют использования таблиц MyISAM, можно использовать преимущества формата MyISAM при создании новой таблицы для использования их при полнотекстных поисках. Загрузка таблицы происходит значительно быстрее, если таблица сначала загружается без индекса, а затем создается индекс, чем при загрузке данных в уже проиндексированную таблицу. Предположим, что в нашем распоряжении имеется таблица apothegm.txt, содержащая высказывания известных людей и имена авторов этих высказываний.

Aeschylus	Time as he grows old teaches many lessons
Alexander Graham Bell	Mr. Watson, come here. I want you!
Benjamin Franklin	It is hard for an empty bag to stand upright
Benjamin Franklin	Little strokes fell great oaks
Benjamin Franklin	Remember that time is money
Miguel de Cervantes	Bell, book, and candle
Proverbs 15:1	A soft answer turneth away wrath
Theodore Roosevelt	Speak softly and carry a big stick
William Shakespeare	But, soft! what light through yonder window breaks?

Если необходимо осуществлять поиск по фразам и авторству одновременно и в отдельности, следует индексировать столбцы отдельно и создать индекс, который содержит оба столбца. Таблицу apothegm можно создать, заполнить и проиндексировать следующим образом:

```
CREATE TABLE apothegm (attribution VARCHAR(40), phrase TEXT);
LOAD DATA LOCAL INFILE 'apothegm.txt' INTO TABLE apothegm;
ALTER TABLE apothegm
  ADD FULLTEXT (phrase),
  ADD FULLTEXT (attribution),
  ADD FULLTEXT (phrase, attribution);
```

После создания таблицы проведем поиск с использованием оператора MATCH определения столбцов, по которым будет производиться поиск, и AGAINST() для определения искомой строки. Например:

```
mysql> SELECT * FROM apothegm WHERE MATCH(attribution)
AGAINST('roosevelt');
+--------------------+-----------------------------------+
| attribution        | phrase                            |
+--------------------+-----------------------------------+
| Theodore Roosevelt | Speak softly and carry a big stick |
+--------------------+-----------------------------------+
mysql> SELECT * FROM apothegm WHERE MATCH(phrase) AGAINST('time');
+------------------+-----------------------------------------+
| attribution      | phrase                                  |
+------------------+-----------------------------------------+
| Benjamin Franklin | Remember that time is money            |
| Aeschylus        | Time as he grows old teaches many lessons |
+------------------+-----------------------------------------+
mysql> SELECT * FROM apothegm WHERE MATCH(attribution,phrase)
    -> AGAINST('bell');
+----------------------+-----------------------------------+
| attribution          | phrase                            |
+----------------------+-----------------------------------+
| Alexander Graham Bell | Mr. Watson, come here. I want you! |
| Miguel de Cervantes  | Bell, book, and candle            |
+----------------------+-----------------------------------+
```

Обратите внимание в последнем примере на то, что этот запрос выполняет поиск искомого слова в различных столбцах, что демонстрирует способность полнотекстного поиска осуществлять поиск в нескольких столбцах сразу. Обратите также внимание на то, что столбцы перечислены в порядке attribution,

phrase. Этот порядок отличается от порядка, в котором они заданы при создании индекса (phrase, attribution), что отражает тот факт, что порядок ничего не значит. Здесь значение имеет сам факт существования индекса типа FULLTEXT, точно состоящего из перечисленных столбцов.

Для того чтобы определить, сколько записей удовлетворяет условию поиска, можно воспользоваться функцией COUNT(*):

```
mysql> SELECT COUNT(*) FROM apothegm WHERE MATCH(phrase)
AGAINST('time');
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
```

По умолчанию при использовании выражения MATCH в предложении WHERE в полнотекстном поиске выводимые строки сортируются в убывающем порядке по критерию уместности. Значения уместности являются неотрицательными значениями, где нулевое значение соответствует "неуместности". Для того чтобы просмотреть эти значения, рассмотрим выражение MATCH в перечне выводимых столбцов:

```
mysql> SELECT phrase, MATCH(phrase) AGAINST('time') AS relevance
    -> FROM apothegm;
+-----------------------------------------------+-----------------+
| phrase                                        | relevance       |
+-----------------------------------------------+-----------------+
| Time as he grows old teaches many lessons     | 1.1976701021194 |
| Mr. Watson, come here. I want you!            | 0               |
| It is hard for an empty bag to stand upright  | 0               |
| Little strokes fell great oaks                | 0               |
| Remember that time is money                   | 1.2109839916229 |
| Bell, book, and candle                        |               0 |
| A soft answer turneth away wrath              |               0 |
| Speak softly and carry a big stick            |               0 |
| But, soft! what light through yonder window breaks? |         0 |
+-----------------------------------------------+-----------------+
```

По умолчанию операция поиска находит записи, которые содержат любое из искомых слов. Таким образом, в результате такого запроса будет найдено любое упоминание слов "hard" и "soft":

```
mysql> SELECT * FROM apothegm WHERE MATCH(phrase)
    -> AGAINST('hard soft');
+---------------------+-----------------------------------------+
| attribution         | phrase                                  |
+---------------------+-----------------------------------------+
| Benjamin Franklin   | It is hard for an empty bag to stand upright |
| Proverbs 15:1       | A soft answer turneth away wrath        |
| William Shakespeare | But, soft! what light through yonder window |
|                     | breaks?                                 |
+---------------------+-----------------------------------------+
```

Более полное управление процессом множественного поиска слов, а также поддержка логического режима поиска были добавлены в MySQL 4.0.1. Этот режим задается добавлением ключевых слов IN BOOLEAN MODE после искомой строки в функции AGAINST(). Логический поиск имеет следующие характеристики.

- Игнорируется правило 50%; поиск найдет слова, даже если они встречаются реже, чем в половине всех записей.
- Результаты не сортируются по коэффициенту уместности.
- В искомой строке можно применять групповые модификаторы. Ведущий символ "плюс" или "минус" означает наличие или отсутствие искомого слова в соответствующих записях. Строка `'bell'` ищет строку, в которой содержится слово "bell," но результатом поиска `'+bell -candle'` в логическом режиме будет строка, содержащая слово "bell" и не содержащая слово "candle".

```
mysql> SELECT * FROM apothegm
    -> WHERE MATCH(attribution,phrase)
    -> AGAINST('bell');
+----------------------+----------------------------------------+
| attribution          | phrase                                 |
+----------------------+----------------------------------------+
| Alexander Graham Bell | Mr. Watson, come here. I want you!     |
| Miguel de Cervantes  | Bell, book, and candle                 |
+----------------------+----------------------------------------+
mysql> SELECT * FROM apothegm
    -> WHERE MATCH(attribution,phrase)
    -> AGAINST('+bell -candle' IN BOOLEAN MODE);
+----------------------+----------------------------------------+
| attribution          | phrase                                 |
+----------------------+----------------------------------------+
| Alexander Graham Bell | Mr. Watson, come here. I want you!     |
+----------------------+----------------------------------------+
```

Замыкающая звездочка действует как групповой символ, в результате чего все слова, начинающиеся с искомого слова, будут совпадать. Например, в результате поиска по `'soft*'` будут найдены слова "soft", "softly", "softness" и т.д.:

```
mysql> SELECT * FROM apothegm WHERE MATCH(phrase)
    -> AGAINST('soft*' IN BOOLEAN MODE);
+----------------------+----------------------------------------+
| attribution          | phrase                                 |
+----------------------+----------------------------------------+
| Proverbs 15:1        | A soft answer turneth away wrath       |
| William Shakespeare  | But, soft! what light through yonder   |
|                      | window breaks?                         |
| Theodore Roosevelt   | Speak softly and carry a big stick     |
+----------------------+----------------------------------------+
```

Однако возможность указания групповых символов не может использоваться для поиска слов, длина которых меньше минимальной длины слова индекса. Полный перечень модификаторов перечислен в разделе, посвященном оператору MATCH, приложения В, "Операторы и функции".

- Общеупотребительные слова игнорируются, как и при нелогическом поиске, даже если они обозначены в соответствии с требованиями. При поиске, заданном с помощью строки `'+Alexander +the +great'`, будут найдены записи, содержащие "Alexander" и "great", но будет пропущено слово "the".
- При поиске фраз выполняется поиск слов, установленных в определенном порядке. Поиск фраз возможен только в версии 4.0.2. Поисковая строка заключается в кавычки и должна содержать все знаки пунктуации, как это указано в таблице. Другими словами, задается *точная* фраза.

```
mysql> SELECT * FROM apothegm
    -> WHERE MATCH(attribution,phrase)
    -> AGAINST('"bell book and candle"' IN BOOLEAN MODE);
Empty set (0.00 sec)
mysql> SELECT * FROM apothegm
    -> WHERE MATCH(attribution,phrase)
    -> AGAINST('"bell, book, and candle"' IN BOOLEAN MODE);
+---------------------+-------------------------+
| attribution         | phrase                  |
+---------------------+-------------------------+
| Miguel de Cervantes | Bell, book, and candle  |
+---------------------+-------------------------+
```

- Есть возможность выполнить логический полнотекстный поиск по столбцам, которые не являются частью индекса FULLTEXT, хотя такой поиск осуществляется медленнее, чем для индексированных столбцов.

До MySQL 4 параметры полнотекстного поиска можно было изменять, только изменив исходный код с последующей перекомпиляцией сервера. В версии 4 появляются перенастраиваемые параметры, которые могут изменяться при установке переменных сервера. Наиболее интересными являются параметры `ft_min_word_len` и `ft_max_word_len`, которые определяют самые короткие и самые длинные индексируемые слова. По умолчанию задаются значения 4 и 254, при этом при создании индексов FULLTEXT слова, длина которых выходит за эти пределы, игнорируются.

Предположим, что потребовалось уменьшить минимальную длину слов с 4 до 3. Что для этого требуется?

1. Запустить сервер с предварительно установленным значением параметра `ft_min_word_len`, равным 3. Для того чтобы гарантировать это, установку данного параметра лучше всего поместить в конфигурационный файл `/etc/my.cnf`:

   ```
   [mysqld]
   set-variable = ft_min_word_len=3
   ```

2. Для всех существующих таблиц, которые уже имеют индексы типа FULLTEXT, индексы необходимо перестроить. Их можно удалить и создать вновь, но лучше всего поступить следующим образом:

   ```
   REPAIR TABLE tbl_name USE_FRM;
   ```

3. Любой новый индекс типа FULLTEXT, который будет создан после изменения этого параметра, автоматически будет использовать новый параметр.

Более детально о конфигурационных файлах и установке переменных сервера можно узнать в приложении Г, "Синтаксис SQL".

Комментарии

SQL-код можно сопровождать комментариями, что удобно при сохранении запросов в файлах. В СУБД MySQL есть два способа комментирования. Комментарием считается любая строка, начинающаяся с символа '#'. Кроме того, можно делать комментирование в стиле языка программирования C. Это значит, что все

строки, расположенные между символами '/*' и '*/', рассматриваются как комментарий. Действие комментария такого типа распространяется на несколько строк.

```
# это однострочный комментарий
/* это тоже однострочный комментарий */
/* а это
   многострочный
   комментарий
*/
```

Третий стиль комментирования появился в версии 3.23.3. Начиная с версии 3.23 комментарий можно начинать с двойного тире и пробела после них; все символы, которые стоят между такими тире и концом строки, рассматриваются как комментарий, т.е. не будут анализироваться интерпретатором. Некоторые другие базы данных тоже считают двойное тире началом комментария. СУБД MySQL это тоже позволяет, но для снятия неопределенности требует дополнительного пробела. Операторы, содержащие выражения наподобие `5--7`, могут быть распознаны как начало комментария. А выражение наподобие `5-- 7` вряд ли можно будет спутать с выражением. Но все же можно порекомендовать воспользоваться одним из традиционных комментирующих символов и обращаться к помощи двойного тире только в кодах, заимствованных из других баз данных.

В СУБД MySQL есть интересная возможность "прятать" ключевые слова, присущие только ей, за символы `'/*! '`. СУБД MySQL "заглядывает" за такие ограничители, в то время когда любая другая СУБД их просто проигнорирует как любой другой комментарий. Это позволит создавать запросы, которые можно применить как для работы с СУБД MySQL, извлекая при этом все преимущества применения дополнительных возможностей MySQL, так и для работы с другими базами данных. Вот пример двух абсолютно идентичных для обычных СУБД запросов. Но MySQL во втором случае выполнит операцию `INSERT DELAYED`:

```
INSERT INTO absence (student_id, date) VALUES(13, "2002-09-28");
INSERT /*! DELAYED */ INTO absence (student_id, date) VALUES(13,
"2002-09-28");
```

Начиная версии 3.22.26 комментарии С-типа могут быть сделаны для учета специфики версии СУБД MySQL, для которой предназначается программный код. Для этого после символов `'/*! '` нужно задать номер версии, и сервер будет игнорировать этот код, если его версия не ниже версии, указанной в комментарии. Комментарий в следующем операторе `CREATE TABLE` игнорируется, если сервер имеет версию 3.23.0 или ниже:

```
CREATE TABLE t (i INT) /*!32300 TYPE = HEAP */;
```

Возможности, которые не поддерживает СУБД MySQL

Этот раздел посвящен описанию возможностей, которые широко представлены во многих СУБД, но отсутствуют в СУБД MySQL. Некоторые из возможностей появятся в недалеком будущем и в СУБД MySQL.

- **Хранимые процедуры и триггеры.** Хранимая процедура представляет собой SQL-код, хранящийся непосредственно на сервере. При выполнении ее не надо повторно посылать запрос от клиента и анализировать его синтаксис. Редактирование хранимой процедуры сразу же влияет на все клиентские приложения, использующие ее в своей работе. Триггер активизирует хранимую процедуру в тот момент, когда с таблицей происходит какое-то событие, например удаление записи. Такая возможность может потребоваться для повторного генерирования какого-нибудь сложного отчета, который всегда должен содержать самые актуальные данные. Обработка хранимых процедур будет реализована в пятой версии СУБД MySQL.
- **Курсоры.** *Курсор* — это логическая сущность, которая ведет себя, как таблица, но таковой не является. Курсор является способом представления столбцов из различных таблиц, как если бы это была одна таблица. Курсоры иногда называют *виртуальными таблицами*. Курсоры будут реализованы в следующих версиях СУБД MySQL.
- **Привилегии уровня записи**. СУБД MySQL поддерживает различные уровни привилегий, начиная от глобальных, распространяющихся только на базы данных, таблицы и только на столбцы. СУБД MySQL не поддерживает привилегии уровня записи. Однако для блокировки записей в приложениях можно воспользоваться функциями `GET_LOCK()` и `RELEASE_LOCK()`. Эта процедура изложена в описании функции `GET_LOCK()` в приложении B, "Операторы и функции".
- **Последовательность '--' в качестве комментария**. Этот стиль комментирования в большинстве версий не поддерживается из-за его неоднозначности. Исключение составляет версия 3.23.3, где такой комментарий приемлем, но с одним нововведением — при наличии пробела в конце. За справкой обратитесь к разделу "Комментарии" данной главы.

4

Оптимизация запросов

В этой главе...

Индексирование
Оптимизатор MySQL
Выбор типа столбцов и эффективность запросов
Эффективная загрузка данных
Проблемы планирования и блокировки
Оптимизация для администраторов

В мире реляционных баз данных доминируют базы данных, наборы данных и операции над ними. База данных — это набор таблиц, а таблица представляет собой набор строк и столбцов. Запросы, создаваемые с помощью оператора SELECT, тоже возвращают наборы строк и столбцов. Это простая абстракция, которая никак не связана с внутренним представлением данных в СУБД. Еще одним абстрактным допущением является то, что операции над таблицами производятся одновременно. Это вполне допустимо, поскольку запросы базируются на концепции операций над множествами, а в теории множеств концепция времени отсутствует.

В действительности все иначе. СУБД реализуют абстрактные концепции на базе реального оборудования. В результате выполнение запросов требует много (иногда чрезвычайно много) времени. А человек — создание нетерпеливое, он не хочет ждать. Таким образом, мы живем в абстрактном мире мгновенных математических операций над множествами, но постоянно ищем пути ускорения работы наших запросов. К счастью, они существуют. Это — индексирование таблиц, позволяющее ускорить их просмотр. Опытный пользователь может извлечь максимальную выгоду из индексов. Это создание запросов, оказывающих влияние на механизм планирования, позволяющий запросам различных клиентов взаимодействовать более эффективно. Это методы настройки "железа", позволяющие

повысить производительность, обойдя физические ограничения, присущие тому или иному типу оборудования.

Именно на этих проблемах и акцентируется внимание в данной главе с единственной целью: помочь вам оптимизировать производительность СУБД, чтобы она обрабатывала запросы как можно быстрее. СУБД MySQL работает достаточно быстро, но даже быстрые системы можно заставить работать еще эффективнее.

Индексирование

Сначала обсудим проблемы индексирования, так как это наиболее важный инструмент ускорения работы запросов. Несмотря на то что есть другие методы, самым эффективным методом было, есть и будет индексирование. В списке рассылки СУБД MySQL пользователи чаще всего спрашивают о том, как можно ускорить выполнение запросов. Как ни странно, в большинстве случаев причина этого кроется в том, что таблицы не индексированы, и их индексирование сразу же решает проблему. Но это не всегда применимо, потому что оптимизация не всегда достигается столь простым способом. Тем не менее, если у таблицы нет индексов, в большинстве случаев попытки улучшить производительность другими методами являются потерей времени. Все-таки сначала следует воспользоваться индексированием, а затем другими вариантами.

В этой главе вы узнаете, что такое индекс, каким образом он может повысить или понизить производительность, а также познакомитесь с принципами проектирования столбцов для таблиц. В следующем разделе мы рассмотрим работу оптимизатора запросов. Для того чтобы можно было использовать преимущества индексирования как можно более эффективно, необходимо не только уметь создавать индексы, но и понимать принципы работы оптимизатора. Дело в том, что можно создать индекс, который не принесет никакой выгоды. Поэтому таких ситуаций необходимо избегать. Иногда, хотя и редко, приходится нейтрализовать действия оптимизатора. Мы коснемся этой темы позднее.

Преимущества индексирования

Рассмотрим принцип работы индекса. Возьмем таблицу, не имеющую индекса. Неиндексированная таблица представляет собой беспорядочный набор строк. Рассмотрим, например, таблицу на рис. 4.1, которая уже упоминалась в главе 1, "Знакомство с СУБД MySQL и SQL". Эта таблица не имеет индексов, так что при поиске определенной компании необходимо будет проверить все ее строки на совпадение искомому значению. Для этого потребуется отсканировать всю таблицу от начала до конца. Это займет очень много времени и будет ужасно неэффективно, особенно если таблица содержит только несколько записей, удовлетворяющих критерию поиска.

На рис. 4.2 показана эта же таблица, но уже проиндексированная по столбцу company_num. Индекс содержит запись о каждой строке из таблицы, и записи индекса отсортированы по значению company_num. Теперь вместо просмотра всех записей таблицы мы можем воспользоваться индексом. Предположим, что требуется найти все строки, содержащие записи о компании с кодом 13. Сканирование возвращает три строки. Значения в индексе отсортированы, поэтому при

достижении значения 14 можно смело завершать сканирование. После 14 мы уже не найдем искомых значений. При поиске значений, которые лежат где-то посередине, с помощью специальных позиционирующих алгоритмов можно перейти к нужной строке без длительного линейного сканирования таблицы (например, методом бинарного поиска). Таким образом, можно быстро позиционироваться на первую совпадающую запись таблицы, сэкономив при этом массу времени. Базы данных используют различные методы быстрого позиционирования индексных значений, и неважно, что эти методы из себя представляют. Важно, что они работают, и то, что индексирование — отличный инструмент оптимизации.

Возникает вопрос: "Почему бы не отсортировать данные и не назвать это индексным файлом? Не ускорит ли это процесс?" Да, может быть, если у вас всего один индекс. Но если потребуется создать второй индекс, отсортировать одни и те же данные двумя различными способами одновременно будет совершенно невозможно. (Например, один индекс накладывается на имена потребителей, а второй — на идентификационные номера потребителей или на телефонные номера.) Рассмотрение индексов отдельно от файлов помогает решить проблему нескольких индексов. При добавлении или удалении новых значений намного проще просматривать более короткие индексные значения, чем продвигаться вдоль длинных строк с данными.

Таблица ad

company_num	ad_num	hit_fee
14	48	0.01
23	49	0.02
17	52	0.01
13	55	0.03
23	62	0.02
23	63	0.01
23	64	0.02
13	77	0.03
23	99	0.03
14	101	0.01
13	102	0.01
17	119	0.02

Рис. 4.1. Неиндексированная таблица

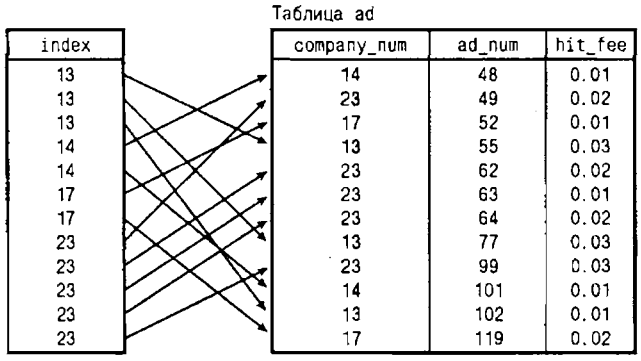


Рис. 4.2. Индексированная таблица

Этот пример демонстрирует приемы индексирования таблиц СУБД MySQL, хотя специфические детали для таблиц определенных типов варьируются. Например, в таблицах MyISAM или ISAM табличные данные хранятся в файлах данных, а значения индексов — в отдельных индексных файлах. Одна таблица может иметь несколько индексов, которые хранятся в одном и том же индексном файле. Каждый индекс в индексном файле состоит из отсортированного массива ключевых записей, которые используются для быстрого доступа к файлу данных. Для сравнения, дескрипторы таблиц BDB и InnoDB также не разделяют табличные данные и индексные значения, хотя они оба содержат индексы в качестве набора отсортированных значений. Дескриптор BDB использует по одному файлу на таблицу для хранения данных и индексных значений. Дескриптор InnoDB использует одно табличное пространство, в пределах которого он управляет данными и индексами для всех таблиц InnoDB.

Предыдущий пример демонстрирует преимущества индексов в контексте запроса к одной таблице, что резко уменьшает необходимость полного просмотра таблицы. Однако индексы еще более удобны при создании запросов, в которых задействовано сразу же несколько таблиц. В однотабличном запросе количество значений, анализируемых во время запроса, равно количеству строк в таблице. В запросах, затрагивающих сразу несколько таблиц, количество комбинаций резко возрастает и равно произведению количества строк в таблицах.

Предположим, у нас есть три неиндексированные таблицы, `t1`, `t2` и `t3`, содержащие столбцы `c1`, `c2` и `c3` соответственно, содержащие по 1000 строк, пронумерованных от 1 до 1000. (Это достаточно простой пример, но на нем хорошо видно решение реальных проблем.) Запрос на выборку всех комбинаций табличных строк выглядит следующим образом:

```
SELECT t1.c1, t2.c2, t3.c3
FROM t1, t2, t3
WHERE t1.c1 = t2.c2 AND t1.c1 = t3.c3
```

Этот запрос возвратит 1000 строк, содержащих три равных значения. Если при обработке этого запроса отсутствуют индексы, совершенно неизвестно, какая строка содержит какое значение. Необходимо перебрать последовательно все комбинации, чтобы найти те, которые удовлетворят условию в предложении `WHERE`. Количество вероятных комбинаций составляет 1000×1000×1000 (1 миллиард!), что в миллион раз больше числа совпадений. Это явно излишние усилия, и совершенно справедливо можно ожидать, что этот запрос будет работать достаточно медленно, даже для такой быстродействующей базы данных, как СУБД MySQL. И это только в таблице, содержащей 1000 строк. Что же будет, если таблицы будут насчитывать миллионы строк? Нетрудно заметить, что это быстро приведет к резкому ухудшению производительности. Индексирование этих таблиц сразу же приведет к ускорению обработки запросов, потому что при наличии индекса запрос обрабатывается по следующему алгоритму.

1. Делается выборка и просматривается содержимое первой строки таблицы `t1`.
2. Непосредственно в индексе таблицы `t2` отыскивается строка в таблице, удовлетворяющая условию выборки из таблицы `t1`. Аналогично, непосредственно в индексе таблицы `t3` отыскивается строка в таблице, удовлетворяющая условию выборки из таблицы `t1`.

3. Переходим к следующей строке в таблице `t1` и повторяем предыдущую процедуру. И так до тех пор, пока все строки таблицы `t1` не будут просмотрены.

В этом случае по-прежнему осуществляется полный просмотр таблицы `t1`, но можно производить прямые просмотры таблиц `t2` и `t3` для прямой выборки строк из этих таблиц. Запрос будет работать "всего лишь" в миллион раз быстрее. (Это обстоятельство, безусловно, следует взять на заметку. Но несмотря на то что описанные в этом примере проблемы вполне реальны, добавление индексов к таблицам обычно не приносит таких поразительных результатов.)

Так СУБД MySQL использует индексы для ускорения поиска строк, удовлетворяющих условиям выборки из предложения `WHERE`. Но это не единственное применение индексов. Они могут улучшить производительность и для следующих типов операций.

- Для ускорения поиска максимального или минимального значения индексируемого столбца при работе с функциями `MIN()` или `MAX()`.
- Для ускорения сортировки с помощью конструкций `ORDER BY` и `GROUP BY`.
- Иногда СУБД MySQL может избежать чтения из файла данных вообще. Предположим, что производится выборка значений только индексированного столбца в MyISAM и другие столбцы из этой таблицы не выбираются. В этом случае достаточно данных, полученных из индекса, чтения данных из файла данных не требуется. Нет смысла читать значения дважды, так что обращаться за справкой к файлу данных при этом не потребуется.

Недостатки индексирования

В целом, можно научиться эффективно использовать индекс. Это значит, что в большинстве случаев, если вы не индексируете свои таблицы, вы наносите вред самому себе. Может показаться, что я рисую картину в розовых красках, которая будет наблюдаться, если индексировать все таблицы во всех базах данных. А есть ли какие-нибудь изъяны в индексах? Да, есть. На практике они незначительны и многократно перекрываются преимуществами от использования индексов. Но о них надо знать.

Во-первых, индексный файл занимает определенное место на диске. При создании большого количества индексов размер такого индексного файла может быстро достичь максимального размера. Это может стать причиной достижения предельного размера таблицами быстрее, чем это было бы без использования индексов.

- Для таблиц ISAM и MyISAM излишнее индексирование таблиц может привести к достижению индексным файлом его максимального размера быстрее, чем файлом данных.
- Таблицы BDB хранят данные и индексы файлы в одном файле. Это, безусловно, является причиной более быстрого достижения максимального размера табличным файлом.
- Таблицы InnoDB размещаются в едином табличном пространстве. При добавлении индексов дисковое пространство, отведенное под табличное пространство, будет исчерпано быстрее. Тем не менее, пока у вас есть свободное пространство на диске, вы можете расширять табличную область и до-

бавлять в нее новые компоненты. (В отличие от файлов, используемых для ISAM, MyISAM, таблиц BDB, размеры табличного пространства InnoDB не ограничены предельными размерами файлов, на которые рассчитана файловая система, так как она рассчитана на хранение множества файлов.)

Во-вторых, индексы ускоряют поиск данных, но замедляют операции добавления, удаления и модификации в индексируемых столбцах. Взаимосвязь тут простая: чем больше индексов имеет таблица, тем больше замедление операций над записями. Ниже, в разделе "Эффективная загрузка данных", мы детально рассмотрим проблемы производительности.

Выбор индекса

Синтаксис оператора создания индексов был рассмотрен в разделе "Создание и удаление индексов" в главе 3, "Синтаксис и использование языка SQL". Но знание синтаксиса само по себе не может помочь в индексировании таблицы. Это зависит от того, как будут использоваться таблицы. Данную главу можно рассматривать как руководство по идентификации и выбору столбцов — кандидатов на индексирование.

- **Необходимо индексировать искомые, сортируемые и группируемые, а не выбираемые столбцы.** Другими словами, лучшей кандидатурой будет столбец, который упоминается в предложении `WHERE` или в предложениях объединения `ORDER BY` или `GROUP BY`, а не столбцы, которые появляются в списке выбора, следующем после ключевого слова `SELECT`.

```
SELECT
    col_a                          <— не подходит на роль
                                   индексируемого столбца
FROM
    tbl1 LEFT JOIN tbl2
    ON tbl1.col_b = tbl2.col_c     <— подходит на роль
                                   индексируемого столбца
WHERE
    col_d = expr;                  <— подходит на роль
                                   индексируемого столбца
```

  Конечно, столбцы, указанные в предложении `WHERE`, могут повторяться в списке выборки. Но смысл сказанного заключается в том, что появление столбца в списке выборки — не лучший показатель того, что должно быть проиндексировано.

  Столбцы, которые упоминаются в предложениях объединения или в выражениях вида *col1 = col2*, можно назвать наиболее удачными кандидатами на индексирование. В качестве такого примера можно привести столбцы `col_b` и `col_c` из только что приведенного запроса. Если СУБД MySQL оптимизирует запросы с помощью объединения, число потенциальных комбинаций строк в таблицах будет уменьшаться из-за уменьшения числа полных сканирований таблицы.

- **Уникальные индексы.** Рассмотрим диапазон значений в столбце. Индексы работают достаточно хорошо со столбцами, имеющими уникальное значение, и хуже со столбцами, содержащими повторяющиеся значения. Например, если столбец, хранящий данные о возрасте, содержит несколько раз-

личных величин, индекс будет работать достаточно быстро. Индекс окажется совершенно бесполезным, когда речь идет о столбце, хранящем данные по полу учащихся и содержащем только два значения, "M" и "F". (Независимо от того, какое значение будет выбираться, будет выбрана половина строк.) При данных обстоятельствах индекс может не использоваться вообще, поскольку оптимизатор вообще не просматривает индекс при необходимости сканирования всей таблицы. (Если определено, что значение встречается более чем в 30 процентах столбцов таблицы.)

- **Используйте короткие индексы.** При индексировании строкового столбца постарайтесь индексировать только часть данных, если в этом есть какой-то смысл. Например, если столбец имеет длину `CHAR(280)`, нет необходимости индексировать по всей длине столбца, так как в большинстве случаев для обеспечения уникальности ключа достаточно будет первых 10–20 символов. Индексирование по префиксу из первых 10–20 символов сэкономит пространство в индексе и, вероятно, также ускорит выполнение запросов. Обработка небольшого индекса требует меньше операций ввода-вывода на диск (меньшие значения сравниваются быстрее). Важно также то, что при меньших объемах ключей в кэш-памяти будет храниться больше ключевых значений, таким образом, СУБД MySQL сможет хранить больше значений одновременно. Это увеличивает вероятность того, что СУБД MySQL сможет осуществить выборку, не подчитывая дополнительные блоки индекса с диска. (При этом не следует терять чувство здравого смысла: индексирование только по первому символу вряд ли будет эффективно.)

- **Как можно эффективнее используйте "левый крайний префикс".** При создании индекса по *n*-столбцам создается *n* индексов. Индекс по нескольким столбцам обрабатывается как несколько отдельных индексов, и любой крайний слева набор столбцов, задействованный в индексе, может быть использован для выборки строк из таблицы. Такой набор называется "левый крайний префикс".

  Предположим, что у нас есть таблица, индексированная по столбцам `state`, `city` и `zip`. Строки в индексе отсортированы в порядке `state/city/zip`. Это значит, что они также будут автоматически отсортированы в порядке `state/city` и только `state`. Это означает, что СУБД MySQL получит преимущество, если в запросе будут определены только значения `state` или только значения `state` и `city`. Таким образом, индекс можно использовать для выборки по следующей комбинации столбцов:

```
state, city, zip
state, city
state
```

  Индекс не будет задействован при поиске значений, которые не удовлетворяют правилу "левого крайнего", например при поиске по столбцу `city` или `zip`. Индекс не может также быть задействован для поиска по указанному штату и почтовому индексу (столбцы 1 и 3 индекса). Однако поиск данных по определенному штату будет работать очень быстро.

- **Не злоупотребляйте индексированием.** Не индексируйте все, что попало, руководствуясь правилом "кашу маслом не испортишь". Это довольно традицион-

ная ошибка. Каждый новый индекс требует дополнительного пространства и, как это было замечено ранее, замедляет операции записи. При модификации содержимого таблицы модифицируется, или реорганизуется, индекс, и чем больше у вас индексов, тем больше времени это займет. Если вы редко или никогда не пользуетесь индексами, то необоснованно замедляете модификацию таблиц. Кроме того, СУБД MySQL просматривает индексы при генерировании плана выполнения выборки. Создание излишних запросов добавляет работы оптимизатору. Это также возможно (хотя и маловероятно) в ситуации, когда у вас слишком много индексов. Создание только тех индексов, в которых есть непосредственная необходимость, — золотое правило, которое помогает избежать таких ошибок.

При добавлении нового индекса в таблицу проверьте, не является ли новый индекс "левым крайним префиксом" уже существующих индексов. Если это так, то не стоит беспокоиться о добавлении нового индекса, потому что в действительности он у вас уже есть. (Например, если уже существует индекс по `state`, `city` и `zip`, то нет смысла добавлять отдельный индекс по `state`.)

- **Тип операции сравнения, которая производится со столбцом.** Индексы применяются при выполнении операций `'<'`, `'<='`, `'='`, `'>='`, `'>'` и `BETWEEN`. Индексы также используются в операциях `LIKE`, когда образец имеет литеральный префикс. Если же значения столбца используются в других операциях, таких как `STRCMP()`, их индексировать нет смысла. Они используются только для сравнения, и, если производится поиск внутри диапазона (такой как `a < b`) с таблицей `HEAP`, индекс не поможет.
- **Для классификации таких запросов используйте журнал регистрации медленных запросов.** Эта регистрация поможет найти запросы, которые выиграют от индексирования. Для просмотра этого журнала можно использовать программу `mysqldumpslow`. (Подробнее об этом речь пойдет в главе 11.) Если определенный запрос появляется вновь и вновь в журнале медленных запросов, это означает, что найден запрос, который может не быть оптимизирован. Его необходимо переписать. При оценке журнала медленных запросов обязательно руководствуйтесь следующим.
  - Скорость выполнения запросов измеряется в реальном времени. Поэтому чем больше загружен сервер, тем больше запросов будет квалифицировано как "медленные" и занесены в журнал медленных запросов регистрации. Это следует принимать в расчет.
  - При использовании параметра `--log-long-format`, кроме медленных запросов, в журнал медленных запросов будут занесены запросы, выполненные без индексирования. Эти запросы не обязательно должны квалифицироваться как "медленные". (К примеру, индекс не нужен для маленьких таблиц.)

Оптимизатор MySQL

После запуска запроса интерпретатор MySQL анализирует запрос на предмет его оптимизации. Оптимизация может привести к ускорению обработки запроса. Рассмотрим основные принципы работы оптимизатора. Дополнительная ин-

формация по этой теме изложена в справочном руководстве по СУБД MySQL, в ней также приведены разные оценки производительности.

Оптимизатор запросов СУБД MySQL позволяет максимально эффективно использовать индексы, но при этом в своей работе он использует и другую информацию. Например, такой запрос будет выполнен очень быстро, независимо от размера таблицы:

```
SELECT * FROM tbl_name WHERE 1 = 0
```

В этом случае СУБД MySQL анализирует предложение WHERE, "понимает", что ни одна строка не удовлетворяет условию, и даже не запускает операцию выборки *из таблицы*. Это можно увидеть в операторе EXPLAIN, который запрашивает у СУБД MySQL информацию о том, каким образом будет выполняться запрос SELECT без его реального выполнения. Для этого перед оператором SELECT необходимо указать ключевое слово EXPLAIN:

```
EXPLAIN SELECT * FROM tbl_name WHERE 1 = 0
```

Этот оператор возвратит следующее:

```
+------------------+
| Comment          |
+------------------+
| Impossible WHERE |
+------------------+
```

Обычно EXPLAIN возвращает больше информации, чем приведено в этом примере. Эта информация включает описание индексов, которые будут использованы для сканирования таблиц, типы используемых объединений, данные о количестве строк, которые будут просканированы в таблице.

Как работает оптимизатор

Оптимизатор запросов СУБД MySQL преследует несколько целей, но главная из них — применять индексы везде, где это возможно, и использовать наиболее строгий индекс для того, чтобы максимально минимизировать количество просматриваемых строк. Это звучит тем более удивительно, что основной задачей при задании оператора SELECT является поиск строк, а не исключение их из рассмотрения. Причина такой работы заключается в том, что чем быстрее оптимизатор исключает строки из рассмотрения, тем быстрее будут найдены строки, удовлетворяющие условию. Запросы будут обрабатываться быстрее, если более строгие тесты будут отработаны в первую очередь. Предположим, что наш запрос проверяет два проиндексированных столбца:

```
SELECT col3 FROM mytable
WHERE col1 = 'some value' AND col2 = 'some other value';
```

Предположим также, что проверка по столбцу col1 дала 900 строк, проверка по столбцу col2 — 300 строк, а объединение — 30 строк. Если протестировать сначала col1, необходимо просмотреть 900 строк, а затем найти из них 30 строк, которые будут совпадать со значениями из столбца col2. Это составляет 870 ошибочных тестов. Если проверку начать со столбца col2, предстоит проверить только 300 строк, чтобы найти 30, которые будут совпадать со столбцом col1. Это всего 270 ошибочных тестов. Очевидно, что второй вариант требует меньше вычислений и операций ввода-вывода.

Кроме того, можно упростить работу оптимизатора, придерживаясь следующих правил.

Сравнивайте только однотипные столбцы. При сравнении проиндексированных столбцов следует сравнивать столбцы, имеющие одинаковые типы. Например, тип CHAR(10) можно рассматривать как тип, аналогичный типам CHAR(10) и VARCHAR(10), но отличный от типов CHAR(12) и VARCHAR(12). Типы INT и BIGINT тоже различаются. Использование столбцов одного и того же типа является требованием всех версий СУБД MySQL до версии 3.23. В противном случае индексы будут проигнорированы. Начиная с версии 3.23 этого правила придерживаться не обязательно, но использование в операциях сравнения идентичных столбцов по сравнению с использованием различных типов повысит производительность. Если столбцы, которые вы сравниваете, являются столбцами различного типа, их тип можно предварительно изменить с помощью оператора ALTER TABLE.

Старайтесь выбирать для сравнений независимые проиндексированные столбцы. Нельзя индексировать столбцы, которые используются в качестве параметров функций или операндов арифметических выражений. Это вынудит СУБД MySQL вычислять выражение для каждой строки таблицы. Иногда это неизбежно, но во многих случаях есть возможность переписать запрос, чтобы он обращался к проиндексированному столбцу непосредственно.

Следующее предложение WHERE продемонстрирует это. В первой строке оптимизатор упростит выражение 4/2 до 2, а затем использует индекс по столбцу my_col, чтобы быстро найти значения my_col меньше 2. Во втором выражении СУБД MySQL будет искать значение my_col для каждой строки, умноженной на 2, и затем сравнит полученный результат с 4. Индекс не используется потому, что все значения в левой части выражения вычисляются:

```
WHERE my_col < 4 / 2
WHERE my_col * 2 < 4
```

Рассмотрим другой пример. Предположим, что столбец date_col проиндексирован. При запуске такого запроса индекс не нужен:

```
SELECT * FROM my_tbl WHERE YEAR(date_col) < 1990;
```

Это выражение не сравнивает непосредственно проиндексированный столбец со значением 1990, оно сравнивает значение, полученное из столбца, и это значение вычисляется для каждого столбца. В результате индекс по столбцу date_col не используется. Как исправить эту ситуацию? Воспользуйтесь литеральной датой, и индекс date_col "заработает".

```
WHERE date_col < '1990-01-01'
```

Но предположим, что у нас нет конкретной даты. Вместо определения конкретной даты может понадобиться определить записи, содержащие даты, которые находятся через определенное количество дней от текущей даты. Эти запросы можно написать тремя разными способами, которые никак нельзя назвать равноценными:

```
WHERE TO_DAYS(date_col) - TO_DAYS(CURRENT_DATE) < cutoff
WHERE TO_DAYS(date_col) < cutoff + TO_DAYS(CURRENT_DATE)
WHERE date_col < DATE_ADD(CURRENT_DATE, INTERVAL cutoff DAY)
```

В первой строке индекс не используется потому, что значение TO_DAYS(date_col) вычисляется для каждой строки. Вторая строка лучше. Как *cutoff*, так и TO_DAYS(CURRENT_DATE) являются константами; таким образом, правая часть выражения может вычисляться оптимизатором до обработки запроса, а не один раз на строку. Но столбец date_col упоминается здесь в качестве параметра функции, таким образом, индекс в этом случае не используется. Наконец, третья строка предоставляет возможность использовать индекс. Опять правая часть выражения вычисляется один раз как константа до выполнения запроса, но на этот раз значение является календарной датой. Значение сравнивается непосредственно со значением столбца date_col, которое больше не надо преобразовывать в дни. В этом случае индекс используется.

Не используйте групповые символы в начале шаблона LIKE. Иногда выборку задают следующим образом:

```
WHERE col_name LIKE '%string%'
```

Это приемлемо в том случае, если целью поиска является строка string, которая может находиться в любом месте в проверяемой строке. Но не следует возводить такой метод в ранг обычного приема. Если в действительности производится выборка строки только тогда, когда подстрока находится в начале столбца, первый символ '%' можно опустить. Например, при поиске строк в столбце, содержащем фамилии, начинающиеся с 'Mac', нужно написать предложение WHERE такого типа:

```
where last_name LIKE 'Mac%'
```

Оптимизатор анализирует начальную литеральную часть образца, и в зависимости от этого индекс будет или не будет использоваться, как если бы было задано выражение в форме, позволяющей использовать индекс по столбцу last_name:

```
where last_name >= 'Mac' AND last name < 'Mad'
```

Это правило не распространяется на образцы, которые используются с оператором REGEXP.

Помогайте оптимизатору производить более эффективную оценку эффективности индекса. По умолчанию при сравнении значений из проиндексированного столбца с константой оптимизатор предполагает, что ключевые значения располагаются равномерно по индексу. Оптимизатор быстро проверит индекс на предмет того, сколько записей будет использоваться. Это поможет в определении возможности использования индекса для сравнения с константами. Оптимизатор можно снабдить более полной информацией, если задать параметр --analyze программе myisamchk или isamchk для анализа распределения значений ключа. Для таблиц MyISAM необходимо воспользоваться программой myisamchk, а для таблиц ISAM – программой isamchk.

- Для анализа ключей необходимо получить право на регистрацию на узле сервера MySQL и право доступа к табличным файлам.
- Необходимо взаимодействовать с сервером на предмет доступа к табличным файлам, так как доступ к файлам во время работы с его файлами может оказаться нежелательным. (Протоколы для координирования доступа к таблице с сервером описаны в главе 13. При этом необходимо использовать протокол, который соответствует доступу к файлам.)

Для проверки работы оптимизатора воспользуйтесь ключевым словом EXPLAIN. Проверьте задействованные в запросе индексы, чтобы быстро исключить из рассмотрения все ненужные строки. Или попробуйте объединить тип STRAIGHT_JOIN для того, чтобы обеспечить определенный порядок объединения таблиц. Посмотрите, как работают различные варианты запроса. Возможно, что лучше будет индексами вовсе не пользоваться. Для MySQL 3.23.12 используйте USE INDEX или IGNORE INDEX, предоставляя возможность выбора индексов серверу.

Несколько раз проверьте, как работают различные варианты запросов. При проверке альтернативных вариантов запросов запускайте их несколько раз. Специфика этой ситуации состоит в том, что второй запрос работает всегда быстрее первого. Это происходит из-за того, что информация от первого запроса остается в кэш-памяти диска и вторично не будет считываться с диска. Поэтому результат такого теста не дает реальной картины. Кроме того, проверку нужно выполнять тогда, когда система имеет относительно стабильную нагрузку, чтобы избежать влияния других процессов.

Не злоупотребляйте автоматическим преобразованием типов данных MySQL. MySQL предоставляет возможность автоматического преобразования типов данных, но если есть возможность избежать этого преобразования, то лучше ею воспользоваться, так как это повышает производительность. Например, если num_col — столбец целого типа, следующие два запроса возвращают одинаковый результат:

```
SELECT * FROM mytbl WHERE num_col = 4;
SELECT * FROM mytbl WHERE num_col = '4';
```

Но второй запрос предполагает преобразование типов данных. Сама по себе операция преобразования имеет маленькие неудобства для превращения целого и строкового типа данных в вещественный тип данных для сравнения. Более серьезной проблемой является то, что, если таблица индексирована по столбцу num_col, преобразование типов данных может помешать использованию индекса.

Подавление оптимизации

Как это ни странно, но иногда приходится подавлять механизм оптимизации СУБД MySQL. Это может понабиться при следующих обстоятельствах.

Необходимость замедления процесса удаления содержимого таблиц. Если необходимо полностью удалить содержимое таблиц, то самым удобным способом будет удаление всего содержимого таблицы с помощью оператора TRUNCATE TABLE с последующим созданием его на базе характеристик, имеющихся в файле .frm:

```
TRUNCATE TABLE tbl_name;
```

До MySQL 4 аналогичный результат получается с использованием оператора DELETE без предложения WHERE:

```
DELETE FROM tbl_name;
```

СУБД MySQL оптимизирует работу этого особого случая использования оператора DELETE: она просто воссоздает пустую таблицу и индексы, пользуясь описанием структур таблиц из информационного файла таблицы. Такая оптимизация предельно ускоряет операцию DELETE, потому что в этом случае не производится

построчное удаление каждой строки. Однако такая операция имеет последствия, которые нежелательны в следующих случаях.

- До MySQL 4 оператор DELETE без предложения WHERE может вернуть количество строк в таблице, равное 0, в то время как в таблице записи есть. TRUNCATE TABLE делает это же в любой версии MySQL. В большинстве случаев это не имеет значения (правда, это может сильно озадачить в том случае, если такого результата не ждешь), а в отдельных случаях такое положение вещей может оказаться просто неприемлемым.
- Для таблиц ISAM после удаления строк значения счетчика AUTO_INCREMENT повторно не используются (см. главу 2). Тем не менее полная очистка таблицы путем воссоздания может привести к сбросу счетчика в 1.

Если вы уже имели дело с этими проблемами и хотите избежать их, используйте оптимальный оператор DELETE, включающий заведомо надежное предложение WHERE:

```
DELETE FROM tbl_name WHERE 1;
```

Это заставит СУБД MySQL произвести построчное удаление. Такой запрос выполнится значительно медленней, но вернет реальное количество удаленных строк. Кроме того, это поможет сохранить существующую последовательную нумерацию AUTO_INCREMENT (только для таблиц в формате MyISAM). К сожалению, для таблиц, хранящихся в формате ISAM, последовательность все равно сбрасывается в 1.

Подавление сортировки при объединении таблицы оптимизатором. Для отключения сортировки оптимизатором можно воспользоваться объединением STRAIGHT_JOIN. При таком объединении из первой таблицы будет выбрано минимальное количество строк. (Если нет полной уверенности в том, какая из таблиц является таковой, можно выбрать сначала и большую таблицу.) Другими словами, поэкспериментируйте с таблицами и укажите в правой части выражения самую критичную таблицу. Запрос работает тем эффективнее, чем меньше область поиска возможных вариантов. Попробуйте обработать запрос двумя способами. Вполне вероятно, что оптимизатор объединяет таблицы не так, как вам это представляется, так что STRAIGHT_JOIN может и не помочь.

Другой вариант — использовать после имени таблицы в списке таблиц модификаторы USE INDEX и IGNORE INDEX, заранее указав использование индексов. Это может пригодиться при ошибках оптимизатора.

Производите поиск результатов в произвольном порядке. Для сортировки получаемых результатов в случайном порядке в версии 3.23.3 была добавлена сортировка по функции ORDER BY RAND(). Другим методом, применимым к старым версиям СУБД MySQL, является выборка столбца, содержащего случайные числа, с последующей сортировкой по этим числам. Но если написать запрос следующего вида, оптимизатор сведет все ваши усилия на нет:

```
SELECT ..., RAND() as rand_col FROM ... ORDER BY rand_col;
```

Суть проблемы, затронутой здесь, заключается в том, что оптимизатор видит вызов функции, считает, что значение столбца является константой, и оптимизирует предложение ORDER BY прямо в запросе! Можно "обмануть" оптимизатор,

указав столбец из таблицы в функции. Например, если в вашей таблице есть столбец age, можно написать такой запрос:

```
SELECT ..., age*0+RAND() as rand_col FROM ... ORDER BY rand_col;
```

Избегайте бесконечных циклов. При модификации индексированных столбцов возможно возникновение ситуации, когда цикл модификации столбца таблицы будет производиться бесконечно, если этот столбец упоминается в предложении WHERE. Это происходит из-за того, что при этом модифицированное значение индекса перемещается в ту часть диапазона выборки, которая не была еще обработана. Предположим, что таблица my_tbl имеет проиндексированный столбец key_col. Вот запрос, вызывающий такой бесконечный цикл:

```
UPDATE iny_tbl SET key_col = key_col+1 WHERE key_col > 0;
```

Решением этой проблемы будет использование столбца key_col в предложении WHERE. Это отключит использование индекса:

```
UPDATE my_tbl SET key_col = key_col+1 WHERE key_col+0 > 0;
```

Есть и другое решение. Установите у себя СУБД MySQL версии 3.23.2 или выше, в которой эта проблема уже решена.

Выбор типа столбцов и эффективность запросов

В этом разделе даются общие рекомендации, как лучше задавать типы столбцов[1].

Не пользуйтесь длинными столбцами, когда вполне достаточно коротких. Не делайте столбцы типа CHAR необоснованно длинными. Если самая длинная строка, хранящаяся в столбце, будет иметь длину 40 символов, то необязательно объявлять ее как CHAR(255). Размеры таблицы можно уменьшить, если вместо типа BIGINT воспользоваться типом MEDIUMINT. Это уменьшит количество операций ввода-вывода, а значения при вычислениях будут обрабатываться быстрее.

Отдавайте предпочтение столбцам фиксированной длины. Это правило применимо к таблицам, которые часто модифицируются, вследствие чего они больше подвержены фрагментации. Например, тип CHAR предпочтительнее типа VARCHAR. Недостатком этого можно считать, что такая таблица займет больше места, но если пренебречь избыточным потреблением дискового пространства, строки фиксированной длины обрабатываются быстрее, чем строки переменной длины. В таблицах InnoDB внутренний формат хранения строк не считает строки фиксированной и переменной длины разными (все строки пользуются заголовком, содержащим ссылки на значение столбца), так что использование типа CHAR, в сущности, не выгоднее типа VARCHAR, поскольку CHAR в среднем займет больше пространства, чем VARCHAR. Так что предпочтительнее использовать VARCHAR, для того чтобы снизить до минимума объем используемой памяти и операций ввода-вывода на диск, необходимых для обработки строк.

В таблицах BDB обычно такой разницы нет. Можно испробовать таблицы обоими способами, запустив при этом эмпирические тесты, чтобы проверить разницу.

---

[1] В этом разделе под типом BLOB подразумеваются сразу два типа – BLOB и TEXT.

Объявляйте столбцы как `NOT NULL`. Это обеспечит быструю обработку данных и уменьшит потребление памяти. Это также упростит обработку запросов, так как отпадает необходимость проверять столбцы на пустые значения.

По возможности пользуйтесь столбцами типа `ENUM`. Строки, имеющие ограниченный набор строгоопределенных значений, можно объявлять с типом `ENUM`. Значения типа `ENUM` обрабатываются очень быстро, так как они имеют числовое внутреннее представление.

Пользуйтесь функцией PROCEDURE ANALYSE( ). Если у вас MySQL 3.23 или более новая версия, используйте функцию `PROCEDURE ANALYSE()`, с помощью которой можно просмотреть характеристики столбцов в таблицах.

```
SELECT * FROM tbl_name PROCEDURE ANALYSE()
SELECT * FROM tbl_name PROCEDURE ANALYSE(16,256)
```

Первый оператор выдает предложения по оптимальным типам для всех столбцов таблицы *tbl_name*. Второй оператор сообщает функции `PROCEDURE ANALYSE()` не предлагать тип `ENUM`, содержащий больше 16 значений или длина которых требует не больше 256 байт (при желании значения можно изменить). Без таких ограничений выводимый результат будет слишком громоздким. В таких случаях объявления `ENUM` очень трудно читаются.

На основании результата работы функции `PROCEDURE ANALYSE()` можно судить об эффективности табличной структуры. Изменить структуру таблиц можно с помощью стандартного оператора `ALTER TABLE`.

Пользуйтесь параметром OPTIMIZE TABLE для таблиц, которые склонны к фрагментации. Таблицы, над которыми часто производится операция модификации (особенно имеющие в своей структуре записи переменной длины), часто подвержены фрагментации. Явление фрагментации само по себе нежелательно потому, что при этом на диске формируется большое количество неиспользуемых блоков. Это вызывает излишнюю потерю времени, так как при выборке требуется считывание дополнительного количества блоков, что в итоге вызывает снижение производительности. Это имеет место и для других типов столбцов, но особенно ощутимо для столбцов типа `BLOB`, потому что они имеют большие размеры. Постоянное использование параметра `OPTIMIZE TABLE` позволяет предотвратить деградацию производительности.

`OPTIMIZE TABLE` работает с таблицами MyISAM и BDB, но дефрагментирует только таблицы MyISAM. Метод дефрагментации, работающий для любой таблицы, загружает таблицу `mysqldump`, после удаляет и воссоздает ее, используя файл дампа:

```
% mysqldump --opt db_name tbl_name > dump.sql
% mysql db_name < dump.sql
```

Храните данные в упакованном виде в типе BLOB. Использование типа `BLOB` для хранения данных позволяет получать данные за одну выборку. Это удобно для хранения данных, которые затруднительно хранить в стандартной табличной структуре. В описании оператора `ALTER TABLE` в главе 3, "Синтаксис и использование языка SQL", был приведен пример, в котором таблица использовалась для хранения результатов из анкеты, созданной на базе Web-узла. В том примере было показано, как с помощью оператора `ALTER TABLE` столбцы добавлялись в таблицу, когда потребовалось добавить новые вопросы в анкету.

Еще один подход к решению этой проблемы — создание приложения, основной функцией которого является преобразование данных, заносимых в Web-анкету, в некое подобие структуры данных, а затем вставка ее в один столбец типа BLOB. Это добавляет работы по кодировке данных (и раскодировке данных при последующей выборке данных из таблицы), но упрощает структуру данных и избавляет от необходимости изменять структуру таблиц всякий раз, когда изменяется структура анкеты.

Однако использование значений типа BLOB может привести к определенным проблемам, особенно если над ними совершается много операций DELETE или UPDATE. Удаление столбца BLOB может оставить большую "дырку" в таблице, которая будет заполнена позднее.

Используйте хешированный индекс. Хешированный индекс иногда помогает повысить эффективность. Можно создать хешированное значение на основании других столбцов и сохранить их в отдельном столбце. По этому значению можно будет производить выборку строк. Это подходит только в случае, когда имеет место совпадение значений. (Хешированные значения бесполезны при поиске внутри какого-то определенного диапазона с применением операторов "<" или ">=".) Хешированные значения генерируются в СУБД MySQL версии 3.23 и выше с помощью функции MD5 ( ). Начиная с версии 4.0.2 и 4.1 можно использовать функции SHA1 ( ) или CRC32 ( ) соответственно.

Хешированный индекс очень удобен при работе со столбцами типа BLOB. До появления версии 3.23.2 этот тип относился к разряду неиндексируемых, но даже в версии 3.23.2 и выше будет удобнее и быстрее найти BLOB-значения с помощью хешированного идентификатора, чем по самому столбцу BLOB.

По возможности избегайте выборки больших значений BLOB и TEXT. Например, запрос на выборку типа SELECT * нельзя считать оптимальным решением, когда предложение WHERE не ограничивает диапазон выборки самыми необходимыми строками. В противном случае есть вероятность выборки очень большого значения типа BLOB без особой необходимости. Есть и другой случай, когда идентифицирующая информация, хранящаяся в столбце типа BLOB, хранится в другом столбце. Сначала по этому значению ищется строка или строки, удовлетворяющие критерию выборки, а по выбранным строкам выбирается само значение BLOB.

Храните значения BLOB в отдельной таблице. При определенных условиях имеет смысл для столбцов типа BLOB создать отдельную таблицу, если оставшиеся столбцы позволят преобразовать первичную таблицу в таблицу с фиксированными строками. Это поможет уменьшить фрагментацию первичной таблицы и позволит получить все преимущества от использования строк постоянной длины. Это также позволит запустить запросы SELECT * в исходной таблице, не пересылая большого количества значений типа BLOB по сети.

Эффективная загрузка данных

Большую часть времени пользователь занят оптимизацией запросов SELECT потому, что это самый популярный запрос языка SQL, и сразу бывает не совсем понятно, как его эффективно применить. (Для сравнения: загрузка данных в таблицу — более простая операция.) Но и для увеличения производительности про-

цесса прямой загрузки данных в базу имеются свои методы. Основные принципы процедуры загрузки данных можно определить следующим образом.

- Массовая загрузка данных всегда предпочтительней построчной, так как в этом случае индекс не обновляется, а заполняется в конце процедуры загрузки.
- Загрузка в таблицу, не имеющую индекса, производится быстрее, чем загрузка в индексированную таблицу. При наличии индекса (или индексов) добавление производится не только в саму таблицу, но и в индекс, модифицируемый при каждом добавлении новых записей.
- Короткие операторы SQL обрабатываются быстрее длинных операторов по двум причинам: синтаксический анализ коротких операторов занимает меньше времени, и собственно передача по сети происходит быстрее.

Некоторые из этих факторов могут показаться несущественными (особенно последний из них), но при загрузке больших объемов данных даже самый незначительный фактор может привести к очень существенному резонансу. На основании принципов, приведенных выше, можно сделать вывод о том, каким образом можно загружать данные эффективно и быстро.

- Оператор `LOAD DATA` всегда эффективнее оператора `INSERT`, поскольку он загружает данные блоками. Обновление индекса при этом производится значительно реже, а серверу нужно обрабатывать только один оператор.
- Оператор `LOAD DATA` эффективнее оператора `LOAD DATA LOCAL`. В первом случае файл должен быть расположен на сервере, а пользователь, выполняющий эту операцию, должен иметь привилегию `FILE`. В таком случае сервер сможет прочесть файл непосредственно с диска. Совсем другая картина складывается при использовании оператора `LOAD DATA LOCAL`: клиент считывает файл и отсылает его по сети на сервер, а это значительно медленней.
- Если необходимо использовать оператор `INSERT`, воспользуйтесь формой, которая позволит произвести загрузку нескольких строк с помощью одного оператора:

  ```
  INSERT INTO tbl_name VALUES(...),(...),...;
  ```

  Причем, чем больше строк определяется в операторе, тем лучше. Это уменьшит число нужных операторов и минимизирует число обновлений индекса.

  Для того чтобы файл дампа содержал операторы `INSERT` вставки множественных строк, при резервном копировании программой `mysqldump` нужно задавать ключ `--extended-insert`. Аналогичный эффект даст ключ `--opt` (оптимизировать), который включает параметр `--extended-insert`. И, наоборот, избегайте использования программы `mysqldump` с ключом `--complete-insert`, так как полученные в таком случае операторы `INSERT` будут рассчитаны на обработку одной строки, они будут длиннее и потребуют больше обработки, чем операторы, полученные без параметра `--complete-insert`.
- Если необходимо использовать несколько операторов `INSERT`, чтобы уменьшить операции с индексом, по возможности сгруппируйте их. Для ти-

пов таблиц, поддерживающих транзакции, этого эффекта можно добиться, включив операторы INSERT в одну транзакцию:

```
BEGIN;
INSERT INTO tbl_name ... ;
INSERT INTO tbl_name ... ;
INSERT INTO tbl_name ... ;
COMMIT;
```

Таблицы типов, не поддерживающих транзакции, можно заблокировать и производить операцию INSERT при заблокированных таблицах:

```
LOCK TABLES tbl_name WRITE;
INSERT INTO tbl_name ... ;
INSERT INTO tbl_name ... ;
INSERT INTO tbl_name ... ;
UNLOCK TABLES;
```

Вы получаете выигрыш в обоих случаях. Индекс обновляется один раз, а не после выполнения каждого оператора INSERT.

- Для уменьшения трафика сети можно воспользоваться сжатым протоколом передачи "клиент/сервер". Для большинства клиентов СУБД MySQL этот режим задается параметром - -compress в командной строке. Прибегать к этому стоит только в медленных сетях, потому что сама процедура сжатия потребует определенных накладных расходов.
- Максимально используйте возможность вставки значений по умолчанию. Опускайте в операторе INSERT те столбцы, которые можно ввести по умолчанию. Ваши операторы станут короче, уменьшится число символов, пересылаемых по сети на сервер. Кроме того, из-за того что операторы будут содержать меньше значений, серверу понадобится производить меньше операций лексического анализа и преобразования значений.
- При добавлении проиндексированных таблиц для уменьшения нагрузки можно воспользоваться пакетной вставкой (оператор LOAD DATA или многострочный оператор INSERT). Это уменьшит нагрузку на систему, создаваемую в результате обновления индекса. Индекс будет обновляться только после добавления всех строк, а не в построчном режиме.
- Процедуру загрузки данных в новую таблицу можно значительно ускорить. Сначала создайте таблицу без индексов, затем загрузите данные, а после этого создайте индексы. Одноразовое создание индекса будет значительно быстрее, чем построчная модификация индекса.
- Значительно эффективнее перед загрузкой данных в проиндексированную таблицу сначала удалить индекс, затем загрузить данные и создать индекс снова. Но эти стратегии не применяются к таблицам InnoDB и BDB, которые не предусматривают оптимизации при раздельном создании индексов.

Перед выбором стратегии удаления или отключения индекса для таблиц типа MyISAM или ISAM необходимо немного поэкспериментировать, чтобы проверить, насколько это резонно. (Нужно понимать, что при загрузке небольшого количества данных в большую таблицу создание нового индекса тоже может занять слишком много времени.)

Индексы удаляются и создаются с помощью операторов DROP INDEX и CREATE INDEX, или применяется работающая с индексом форма оператора ALTER TABLE. Есть два варианта для активизации и отключения индексов.

Альтернативный подход заключается в активизации и отключении индексов программой myisamchk или isamchk. Для этого пользователь должен иметь свою учетную запись на сервере MySQL и право на запись в табличные файлы. Для отключения индексов таблиц войдите в соответствующий каталог базы данных и запустите одну из следующих команд.

- Можно использовать формы DISABLE KEYS и ENABLE KEYS оператора ALTER TABLE:

  ```
  ALTER TABLE tbl_name DISABLE KEYS;
  ALTER TABLE tbl_name ENABLE KEYS;
  ```

  Эти операторы включают и отключают корректировку любого неуникального индекса в таблице.

- Альтернативный подход заключается в активизации и отключении индексов программой myisamchk или isamchk. Для этого пользователь должен иметь свою учетную запись на сервере MySQL и право на запись в табличные файлы. Также следует обратить внимание на меры предосторожности, предложенные в главе 13, "Резервное копирование и восстановление баз данных", чтобы отключить обращения сервера к таблице, пока используются файлы.

Формы DISABLE KEYS и ENABLE KEYS являются предпочтительным методом отключения и активизации потому, что основную работу выполняет сервер. Однако они доступны только при наличии MySQL 4. (Заметьте, что если используется LOAD DATA для загрузки данных в пустую таблицу, то сервер выполняет эту работу автоматически.)

Деактивизируя индексы таблицы вручную, убедитесь в том, что сервер в данный момент не работает с таблицей, и только после этого перейдите в соответствующую базу данных и запустите следующую команду:

```
% myisamchk --keys-used=0 tbl_name
```

После загрузки данных в таблицу активизируйте индексы снова:

```
% myisamchk --recover --quick --keys-used=n tbl_name
```

n в данном случае интерпретируется как шаблон, который показывает, какой индекс будет работать. Нулевой бит соответствует первому индексу. Например, если таблица имеет три индекса, значение *n* должно быть равно 7 (двоичный 111). Номера индексов вычисляются параметром - -description:

```
% myisamchk --description tbl_name
```

Команды для таблиц ISAM аналогичны, кроме того, что предпочтительнее использовать isamchk, а не myisamchk, значение --keys-used для isamchk указывает на использование индекса с наибольшим номером. (Для таблиц с тремя индексами число *n* будет равно 3.)

Такие принципы загрузки данных применимы и в параллельных средах, когда различные клиенты выполняют запросы параллельно. Например, пользователь стремится максимально исключить "долгоиграющие" запросы SELECT к таблицам, которые часто модифицируются. Это всегда вызывает много конфликтов и

снижает производительность системы. Возможным выходом из такой ситуации (если ваши операции записи представляют собой в основном операции INSERT) является сохранение этих записей во вспомогательной таблице и периодическое добавление их из вспомогательной таблицы в основную таблицу. Такая стратегия не подходит, если новые записи необходимо видеть немедленно, но если такая стратегия подходит и новые записи могут некоторое время быть недоступны, использование временного файла может помочь в двух аспектах. Во-первых, это ограничит возникновение конфликтов при запросах SELECT из основной таблицы, что ускорит их выполнение. Во-вторых, массовая загрузка данных из вторичной таблицы в основную займет меньше времени, чем построчная загрузка. Одно обновление индекса потребует гораздо меньше времени, чем обновление индекса с построчным добавлением.

Одним из применений этой стратегии является доступ к базе данных с Web-сервера при регистрации на Web-странице. В этом случае не будет критичным решением размещение новых записей в главной таблице.

Еще одной стратегией уменьшения количества обращений к таблице является использование параметра DELAYED_KEY_WRITE во время создания таблиц в формате MyISAM. Это можно делать тогда, когда потеря одной записи при сбое системы не будет критичной. (Это может быть случай использования СУБД MySQL для регистрации в системе.) Этот параметр делает обновление индекса событием достаточно редким.

По необходимости можно запустить сервер в режиме задержки обновления индекса. Для этого запустите программу mysqld с ключом - -delayed-key-write. В таком случае запись в индексный блок задерживается до тех пор, пока не потребуется обновление индекса для освобождения пространства под другие индексные значения при заполнении таблицы до выполнения команды flush-tables или закрытия индексируемой таблицы.

Для подчиненного сервера репликации для того, чтобы отложить обновление индексов для всех таблиц формата MyISAM, невзирая на то, как они были изначально созданы на главном сервере, используйте параметр --delay-keywrite=ALL.

Проблемы планирования и блокировки

Все предыдущие разделы посвящались прежде всего методам ускорения работы запросов отдельных клиентов. СУБД MySQL предоставляет возможность изменять индивидуальные приоритеты выполнения запросов, что позволяет разным клиентам эффективно взаимодействовать таким образом, чтобы не возникали длительные блокировки. Изменение приоритетов может привести к повышению быстродействия отдельных типов запросов. Сначала познакомимся со стратегией планирования, принятой в СУБД MySQL по умолчанию, а затем посмотрим, каким образом можно на нее повлиять. Сначала определимся с терминами. Назовем читателем клиента, осуществляющего выборку (оператор SELECT). Клиента, осуществляющего операции модификации записей таблицы (операторы DELETE, INSERT, REPLACE или UPDATE), назовем писателем.

Вкратце стратегию планирования СУБД MySQL можно определить так.

- Запросы записи должны выполняться в порядке их поступления.
- Запросы, выполняющие операцию записи, имеют более высокий приоритет, чем запросы, выполняющие операцию чтения.

Для таблиц типа MyISAM и ISAM стратегия планирования осуществляется с применением блокировок таблиц. Перед тем как клиент получает доступ к таблице, ее необходимо заблокировать. Это можно сделать явным образом с помощью оператора LOCK TABLES. Но обычно менеджер блокировки сервера блокирует таблицы автоматически. После того как клиент завершит работу с таблицей, блокировка снимается. Это можно сделать как явным способом с помощью оператора UNLOCK TABLES, так и воспользоваться автоматической разблокировкой сервера.

Клиент, производящий операцию записи, должен осуществлять блокировку исключительного доступа к таблице. В процессе записи таблица находится в неопределенном состоянии. Данные в этот момент могут удаляться, добавляться или модифицироваться, и все индексы, имеющие отношение к данной таблице, также могут изменяться. Предоставить пользователям доступ к таблице в этот момент будет очевидной ошибкой. Совершенно понятно, что возможность записывать данные в одну таблицу сразу же нескольким пользователям быстро превратит таблицу в хаотичных данных. Аналогичным образом, чтение данных из обновляемой таблицы приведет к неточным результатам.

Клиент, осуществляющий операцию чтения, должен блокировать таблицу от записи в нее другими пользователями для того, чтобы в момент чтения никто не мог изменить содержимое таблицы. Однако при этом блокировка не должна давать исключительные права на чтение. Эта блокировка должна предоставлять доступ для одновременного чтения другим клиентам. Чтение не изменяет содержимого таблицы, поэтому нет особого повода запрещать другим пользователям доступ к таблице для чтения в тот момент, когда читатель производит чтение сам.

СУБД MySQL позволяет оказывать влияние на стратегию планирования с помощью специфических модификаторов запросов. Одним из таких модификаторов является ключевое слово LOW_PRIORITY в операторах DELETE, INSERT, LOAD DATA, REPLACE и UPDATE. Вторым модификатором является ключевое слово HIGH_PRIORITY для операторов SELECT. Третьим модификатором является ключевое слово DELAYED для операторов INSERT и REPLACE.

Ключевое слово LOW_PRIORITY влияет на процесс планирования следующим образом. Если запрос на операцию записи в таблицу поступает во время чтения, таблица блокируется до тех пор, пока запрос на чтение не будет обработан полностью. Если в момент ожидания запроса на запись поступает еще один запрос на чтение, последний тоже будет заблокирован, так как приоритет запросов на чтение меньше, чем запросов на запись. Последовательность действий тут будет следующей. В соответствии со стратегией планирования, принятой по умолчанию, после завершения работы первого читателя активизируется писатель, затем, после завершения работы писателя, запускается второй читатель.

Если запрос на запись объявлен как LOW_PRIORITY, операция записи будет иметь приоритет ниже, чем операция чтения. В этом случае, если второй читатель обращается во время ожидания операции записи, его операция будет выполнена до операции записи. Только в случае, когда читателей, ожидающих своей очереди на выполнение, больше нет, писатель сможет продолжить свою работу.

Одним довольно существенным осложнением, которого можно ожидать от такого изменения стратегии планирования, является то, что теоретически вероятно, что писатель с модификатором LOW_PRIORITY будет заблокирован навсегда. Если во время, когда запрос писателя уже ожидает выполнения запросов на чтение, будут поступать новые, очередь писателя не наступит никогда.

Действие ключевого слова HIGH_PRIORITY в запросах SELECT абсолютно аналогично. Оно позволяет оператору SELECT обходить операторы записи, ожидающие своей очереди, даже если они имеют обычный приоритет.

Модификатор DELAYED для операции INSERT действует следующим образом. Предположим, что поступает запрос INSERT DELAYED. Сервер ставит его в очередь и немедленно возвращает статус клиенту, чтобы последний мог продолжить работу до того, как факт добавления строк будет иметь место. Если в этот момент есть читатели, производящие операцию чтения из таблицы, строки будут поставлены в очередь. При отсутствии читателей сервер начинает добавлять строки в порядке очередности. В процессе добавления сервер постоянно будет делать паузу и проверять, есть ли новые запросы на чтение, ожидающие своей очереди. Если это так, выполнение заданий из очереди задержанных операций приостанавливается и начинают обрабатываться запросы читателей. Когда все запросы на чтение будут обработаны, сервер опять возвращается к процедуре добавления строк. Этот процесс будет продолжаться до тех пор, пока очередь не очистится.

Ключевые слова LOW_PRIORITY и DELAYED в этом смысле похожи, поскольку оба замедляют добавление строк, но они совершенно по-разному влияют на операции клиента. Ключевое слово LOW_PRIORITY заставляет клиента ждать, пока строки не будут вставлены. Ключевое слово DELAYED позволяет клиенту продолжать работу, а сервер откладывает операцию добавления, пока у него не появится возможность обработать строки.

Ключевое слово INSERT DELAYED может очень пригодиться в случае, когда клиенты запускают длительные запросы SELECT, а администратор не хочет, чтобы возникла длительная блокировка операций добавления. Клиент, выдающий запрос INSERT DELAYED, сможет продолжить работу, потому что сервер ставит запросы на добавление записей в очередь.

Однако нужно понимать и другие различия в поведении обычного оператора INSERT и оператора INSERT DELAYED. Клиент получает сообщение об ошибке, когда оператор содержит синтаксическую ошибку, и больше никакой информации, которая сообщается при нормальном завершении работы оператора. Например, нельзя будет доверять возвращаемому значению AUTO_INCREMENT. Не будет возвращено количество повторений в уникальных индексах, если таковые имеют место. Это происходит из-за того, что операция вставки возвращает статус до реального завершения операции. В качестве другого "подводного камня" можно назвать ситуацию, когда строки из операторов INSERT DELAYED ставятся в очередь на добавление в тот момент, когда произошел сбой сервера или он был остановлен командой `kill -9`. В этом случае сервер добавляет строки до наступления полного останова.

Дескриптор MyISAM позволяет сделать исключение из основных принципов, в соответствии с которыми читатели препятствуют работе писателей. Это происходит при условии, что таблицы в формате MyISAM не имеют "дыр" (т.е. в них

нет удаленных строк). В этом случае любой оператор INSERT обязательно добавляет строки в конец таблицы, а не в середину. При таких обстоятельствах клиентам позволяется добавлять строки к таблицам даже тогда, когда другие клиенты читают из них. Такая процедура называется параллельной вставкой, так как она позволяет продолжить работу одновременно с поиском без блокировки. Если вы используете эту возможность, обратите внимание на следующее.

- Не используйте параметр LOW_PRIORITY с оператором INSERT. Это всегда блокирует операцию INSERT для читателей и мешает делать параллельные вставки.
- Читатели, которым требуется явно заблокировать таблицу, но которые намерены делать параллельные вставки, должны использовать LOCK TABLES ... READ LOCAL, а не LOCK TABLES ... READ. С помощью ключевого слова LOCAL можно выполнить блокировку, позволяющую делать параллельные вставки, потому что она применяется только для существующих строк и не блокирует новых строк, добавленных с конца.

Не все модификаторы планирования появились в СУБД MySQL одновременно. Ниже приведена таблица, где упоминаются все модификаторы и указана версия, в которой они появились впервые. Этот список можно использовать для определения возможностей версии СУБД MySQL.

Тип оператора	Версия СУБД MySQL, в которой впервые появилась такая возможность
DELETE LOW_PRIORITY	3.22.5
INSERT LOW_PRIORITY	3.22.5
INSERT DELAYED	3.22.15
LOAD DATA LOW_PRIORITY	3.23.0
LOAD DATA CONCURRENT	3.23.38
LOCK TABLES ... LOW_PRIORITY	3.22.8
REPLACE LOW_PRIORITY	3.22.5
REPLACE DELAYED	3.22.15
SELECT ... HIGH_PRIORITY	3.22.9
UPDATE LOW_PRIORITY	3.22.5
SET SQL LOW PRIORITY UPDATES	3.22.5

Уровни блокировки и параллельность

Модификаторы планирования, которые только что обсуждались, позволяют влиять на стандартную политику планирования. Они были добавлены прежде всего для того, чтобы справляться с проблемами, возникающими при использовании блокировок на уровне таблиц, которые налагаются дескрипторами MyISAM и ISAM для сохранения целостности баз данных.

Сейчас СУБД MySQL может работать с таблицами типов BDB и InnoDB, которые предусматривают блокирование на различных уровнях и имеют различные характеристики производительности с точки зрения управления содержимым.

Дескриптор BDB использует блокировки страничного уровня. Дескриптор InnoDB использует блокировки на уровне строк, но только по мере необходимости. (В большинстве случаев, например когда выполняются только операции чтения, дескриптор InnoDB может не применять блокировки вообще.)

Уровень блокировки, используемый дескриптором таблицы, существенно влияет на параллельность работы клиентов. Предположим, что два клиента хотят обновить одну и ту же строку в заданной таблице. Для выполнения операций обновления каждым клиентом потребуется блокировка записи. Для таблиц MyISAM дескриптор заблокирует таблицу для первого клиента, что приведет к остановке работы с этой таблицей другого клиента до тех пор, пока первый клиент не завершит свою работу. Для таблиц BDB может быть достигнута значительно более высокая степень параллельности работы различных клиентов, если обе строки не размещаются на одной и той же странице. Для таблиц типа InnoDB параллельность будет даже больше; оба обновления могут производиться одновременно, если оба клиента не обновляют одну и ту же строку.

Основной принцип заключается в том, что блокировка таблиц на более высоком уровне улучшает параллельность работы, так как при этом таблицу может использовать одновременно большее количество клиентов, при условии, что они работают с различными ее частями. Сложность реализации этого заключается в том, что для смешивания различных типов запросов подходят разные типы таблиц.

- Типы ISAM и MyISAM характеризуются большой скоростью выполнения операций поиска. Однако использование блокировок уровня таблиц может создавать определенные сложности для смешивания операций выборки и обновлений, особенно если операции поиска имеют тенденцию к увеличению времени выполнения. При таких условиях обновлениям, возможно, придется ожидать определенное время, перед тем как продолжить свое действие.
- Таблицы типа BDB и InnoDB обеспечивают лучшую производительность при большом количестве обновлений. Так как здесь блокировка производится не на уровне целой таблицы, а на уровне страницы или строки, уровень блокируемой таблицы будет меньше. Это уменьшает влияние блокировки и улучшает возможности параллельной обработки.

Блокировка на уровне таблиц имеет преимущество перед другими, более точными методами блокировки с точки зрения предотвращения возникновения клинчей. При блокировке на уровне таблиц сервер при просмотре очереди запросов может определить, какие таблицы требуются, и предварительно их заблокировать. Клинч в работе с таблицами типа InnoDB и BDB может возникнуть из-за того, что дескрипторы не получают все необходимые блокировки в начале транзакции. Вместо этого блокировки происходят по мере возникновения в процессе обработки транзакции. Вполне вероятна ситуация, когда два запроса, уже заблокировавшие таблицу, затем будут пытаться установить дальнейшие блокировки на основе снятия предыдущих. В результате каждый клиент устанавливает блокировку, которая будет сдерживать активность других. Это приводит к клинчу, в результате которого сервер будет вынужден прервать одну из транзакций. Для таблиц типа BDB возникновение такого клинча можно предотвратить, используя параметр `LOCK TABLES`, позволяющий накладывать блокировки явным

образом, так как дескриптор BDB осуществляет поиск таких блокировок. Это не сработает для таблиц типа InnoDB, поскольку в этом случае дескриптор не получает информации о блокировках, установленных LOCK TABLES.

Оптимизация для администраторов

Все предыдущие разделы этой главы посвящались методам оптимизации операций создания, индексирования таблиц и написания запросов, которые доступны обычным пользователям СУБД MySQL. Но оптимизация оставляет широкое поле деятельности и для системных администраторов, которые управляют сервером СУБД MySQL или машиной, на которой он работает. Есть параметры сервера, которые имеют непосредственное отношение к обработке запросов. Они поддаются настройке. Основные моменты, которые необходимо принимать во внимание в процессе оптимизации, выполняемой администратором, приведены ниже.

- Доступ к данным, находящимся в памяти производится быстрее, чем к данным, находящимся на диске.
- Хранение данных в памяти настолько долго, насколько это возможно. Это позволяет снизить активность дисков.
- Сохранение информации, хранящейся в индексе, важнее, чем сохранение содержимого записей данных.

Ниже приводятся способы реализации этих принципов на практике.

Увеличение размера кэша сервера. У сервера есть очень много параметров (переменных), которые можно изменять для управления его работой; некоторые из них непосредственно влияют на скорость обработки запросов. Наиболее важными изменяемыми параметрами являются размеры кэша таблицы и кэшей, использованных дескрипторами таблиц при выполнении операций индексирования. При наличии достаточного объема памяти распределение ее под буферы кэша позволяет дольше хранить информацию в памяти и уменьшить операции с диском. Благодаря этому доступ к информации, хранящейся в оперативной памяти, можно получить быстрее, чем считывать их с диска.

- Кэш таблиц используется для хранения информации об открытых таблицах. Его размер контролируется серверной переменной `table_cache`. Если сервер получает множество таблиц, этот кэш заполняется и сервер должен закрыть таблицы, которые не использовались, чтобы освободить пространство под новые таблицы. Кроме того, просмотрев индикатор состояния `Opened_tables`, можно узнать, насколько эффективно используется кэш:

  ```
  SHOW STATUS LIKE 'Opened_tables';
  ```

  `Opened_tables`, отражает сколько раз таблица может быть открыта, если она еще не была открыта ни разу. (Это значение также отображается как значение `Opens` в выводе команды `mysqladmin status`.) Если это число становится стабильным или медленно возрастает, вероятно, что установлено достаточно правильное значение. Если значение возрастает достаточно быстро, это значит, что кэш переполнен и таблицы должны быть закрыты, чтобы освободить пространство для других таблиц. При наличии дескрип-

торов файлов увеличение размера кэша таблиц позволяет уменьшить количество операций открытия таблиц.

- Ключевой буфер используется дескрипторами таблиц MyISAM и ISAM для хранения блоков индексов при выполнении операций, использующих и индексы. Его размер контролируется серверной переменной `key_buffer_size`. Чем больше значение, тем больше индексных блоков одновременно сможет хранить сервер MySQL в оперативной памяти, что увеличивает вероятность обнаружения ключевых значений непосредственно в памяти, не читая информационные блоки с диска. Объем буфера, принятый по умолчанию, составляет 8 Мбайт. При наличии больших объемов свободной памяти этот объем становится сдерживающим фактором, но его можно увеличить, наблюдая при этом существенное увеличение скорости выполнения выборок с использованием индекса и операций создания и модификации индексов.
- Дескрипторы InnoDB и BDB имеют свои собственные кэши для буферизации данных и индексных значений. Их размеры контролируются переменными `innodb_buffer_pool_size` и `bdb_cache_size`.
- Дескриптор InnoDB также содержит буфер журнала, размер которого управляется переменной `innodb_log_buffer_size`.
- Другим кэшем специального назначения является кэш, описанный в разделе "Кэш запросов".

Процедура для переменных сервера описана в главе 11, "Общее администрирование MySQL". При изменении значений параметров необходимо придерживаться следующих рекомендаций.

- Изменяйте по одному параметру за один раз. В противном случае при одновременном изменении сразу нескольких переменных трудно оценить эффект, полученный от каждого изменения.
- Увеличивайте значения параметров постепенно. При значительном увеличении переменной за один раз система очень быстро может израсходовать все ресурсы, что немедленно скажется на ее производительности.
- Чтобы получить представление о порядке величин, которые соответствуют потребностям вашей системы, посмотрите на образцы конфигурационных файлов `my-small.cnf`, `my-medium.cnf`, `my-large.cnf` и `my-huge.cnf`, имеющиеся в дистрибутиве MySQL. (Их можно найти в каталоге `support-files` исходного дистрибутива и в каталоге `share` в двоичных дистрибутивах.) Эти файлы помогут понять, с помощью каких параметров можно изменять настройки серверов, которые обеспечивают различные уровни использования, а также некоторые примерные значения для этих параметров.

Ниже перечислены другие стратегии, которые позволяют увеличить эффективность работы сервера.

Отключите ненужные дескрипторы таблиц. Сервер не будет отводить память ни под какие дескрипторы, что позволит использовать эту память в любых других целях. Дескрипторы ISAM, InnoDB и BDB можно отключить полностью на этапе построения сервера, а дескрипторы InnoDB и BDB – во время запуска сервера. Подробнее об этом рассказывается в главе 11.

Упрощайте таблицу разрешений. Несмотря на то что сервер кэширует таблицы разрешения в памяти, при существовании хоть одной строки в таблице `tables_priv` или `columns_priv` сервер будет вынужден проверять привилегии уровня таблиц и столбцов для каждого запроса. Если эти таблицы пусты, сервер оптимизирует свои проверки привилегий и пропустит проверку на этих уровнях.

При построении MySQL из исходного текста постарайтесь использовать статические библиотеки, а не распределенные. Динамические программы, использующие разделенные библиотеки, позволяют сэкономить дисковое пространство, но статические программы работают быстрее. (Однако статические программы нельзя использовать, если вы используете пользовательские функции, так как механизм UDF в своей работе опирается на динамические ссылки.)

Кэш запросов

Начиная с MySQL 4.0.1 для ускорения обработки операторов `SELECT`, которые обрабатываются повторно, сервер может использовать в работе кэш запросов, что позволяет значительно повысить производительность. Кэш запросов работает следующим образом.

- После первой обработки оператора `SELECT` сервер запоминает текст запроса и возвращаемые им результаты.
- В следующий раз сервер просматривает запрос и не делает ничего для того, чтобы выполнить его повторно. Вместо этого сервер выбирает результат такого запроса непосредственно из кэша и возвращает его клиенту.
- Кэширование запросов производится в текстовом виде, в котором они принимаются сервером. Запросы рассматриваются также, если текст запросов совпадает. Запросы считаются различными, если они отличаются регистром или приходят от клиентов, которые используют разные кодировки или коммуникационные протоколы. Запросы также считаются различными, даже если они не относятся к одним и тем же таблицам (например, если они ссылаются на идентичные таблицы, но в различных базах данных).
- При обновлении таблицы любые кэшированные запросы, в которых сделана ссылка на эту таблицу, становятся ошибочными и удаляются. Это предотвращает возвращение устаревших результатов.

Поддержка кэша запросов встроена по умолчанию. Если нет желания использовать кэш и вы хотите избежать даже минимальной перегрузки, которую он может вызвать, можно построить сервер без него, задав сценарию построения параметр `--without-query-cache`.

Для серверов, которые имеют поддержку кэша, работа кэша основывается на значениях трех переменных.

- Переменная `query_cache_size` задает размер кэша запросов. Значение 0 отключает кэш, что и является стандартной установкой. (Другими словами, кэш не используется, если вы не включите его явным образом.) Чтобы включить кэш, задайте значение `query_cache_size`, определяющее в байтах желательный размер кэша. Например, для того чтобы отвести под кэш 16 Мбайт, воспользуйтесь в конфигурационном файле следующей установкой:

```
[mysqld]
set-variable = query_cache_size=16M
```

- Переменная `query_cache_limit` устанавливает максимальный кэшируемый размер результирующего набора; результаты запросов, превышающие этот размер, никогда не кэшируются.
- Переменная `query_cache_type` задает режим работы кэша запросов. Ниже приведены допустимые значения режимов.

Режим	Назначение
0	Не кэшировать
1	Кэшировать все запросы, за исключением таблицы, которые начинаются с `SELECT SQL_NO_CACHE`
2	Кэшировать по запросу только те запросы, которые начинаются с `SELECT SQL_CACHE`

Отдельные клиенты начинают работу с возможностями кэширования, обозначенными как режим кэширования по умолчанию. Клиент может изменить режим кэширования запросов сервером с помощью следующего оператора:

```
SET SQL_QUERY_CACHE_TYPE = val;
```

Переменная `val` может принимать значения 0, 1, 2, которые имеют то же значение, что таблицы значения переменной `query_cache_type`. Символические значения `OFF`, `ON` и `DEMAND` являются синонимами `0`, `1` и `2`.

Клиент также может управлять кэшированием отдельных запросов. Для этого после ключевого слова SELECT задается специальный модификатор. Конструкция `SELECT SQL_CACHE` приводит к кэшированию результата запроса, если кэш работает в режиме запроса. Использование конструкции `SELECT SQL_NO_CACHE` приводит к тому, что результат не будет кэширован.

Подавление кэширования удобно при обработке запросов на выборку из постоянно изменяющейся таблицы. В таком случае кэш не будет использоваться часто. Предположим, что в обычной таблице MySQL протоколируются запросы к Web-серверу, кроме того, периодически запускаются суммирующие запросы по этой таблице. Для существенно загруженного Web-сервера новые строки вставляются в таблицу довольно часто, таким образом, результаты кэшированных запросов для данной таблицы довольно быстро станут неактуальными. Затруднение заключается в том, что даже при условии, что итоговые запросы делаются достаточно часто, кэш может как-либо в этом помочь. При таких условиях имеет смысл делать запросы с применением модификатора `SQL_NO_CACHE`, чтобы дать понять серверу, что результаты его кэширования никому не нужны.

Проблемы, связанные с используемой аппаратной платформой

В начале этой главы обсуждались методы, направленные на улучшение производительности сервера, независимо от конфигурации аппаратной части. Конечно, можно подобрать компьютеры получше, чтобы ваш сервер заработал быстрее. Но далеко не всякая модернизация оборудования части одинаково помогает

в этом. Принципы оценки доработок аппаратного оборудования во многом совпадают с принципами настройки параметров сервера. Необходимо разместить как можно больше информации в самую быструю среду хранения и хранить ее там как можно дольше.

Производительность сервера можно увеличить, выполнив следующие настройки аппаратной платформы.

- **Добавьте оперативную память.** Это позволит увеличить размеры кэша и буфера сервера. Таким образом, можно добиться более частого использования информации, хранящейся в памяти, при уменьшении необходимости подкачки информации с диска.
- **Если достаточно оперативной памяти для свопинга, переконфигурируйте свою систему таким образом, чтобы убрать все другие устройства свопинга.** В противном случае некоторые системы будут продолжать свопинг на диск, даже если при этом будет достаточно оперативной памяти.
- **Чтобы сократить задержки операций ввода-вывода, установите более быстродействующие диски.** Время поиска является наиболее критическим фактором, влияющим на производительность. Однако, если есть выбор между добавлением памяти или установкой более быстрых дисков, то первое позволит увеличить размер кэша, который в свою очередь снижает активность дисков.
- **Старайтесь распределить дисковые операции между различными физическими устройствами.** Если можно распределить чтение или запись между различными физическими устройствами, это позволит увеличить производительность по сравнению с чтением и записью с одного и того же устройства. Например, можно хранить базы данных на одном устройстве, а журналы — на другом. Запись на два разные устройства одновременно будет происходить быстрее, чем на одно и то же устройство. Следует заметить, что использование различных логических разделов, находящихся на одном и том же физическом устройстве, будет недостаточно. Это не поможет, так как они по-прежнему будут конкурировать за один и тот же физический ресурс (головки диска). Процедура переноса баз данных описана в главе 10, "Каталог данных MySQL".

  Перед тем как перенести данные на другое устройство, убедитесь в том, что вам понятны характеристики загрузки вашей системы. И если новый диск уже занят каким-то загруженным ресурсом, такой перенос не принесет никакой пользы. Например, вы не получите никакой выгоды от переноса базы данных на устройство, где располагается дерево документов сервера. (Конечно, при наличии одного физического диска какое-то существенное перераспределение загрузки диска вряд ли возможно.)

  Кроме того, использование устройств RAID может дать вам определенные выгоды, которые несет в себе параллельная работа.
- **Используйте многопроцессорную аппаратуру.** Для многопотоковых приложений, каким является и сервер MySQL, многопроцессорное оборудование позволяет выполнять несколько процессов одновременно.

II

Программные интерфейсы СУБД MySQL

В этой части...

Глава 5. Введение в программирование в СУБД MySQL
Глава 6. Программный интерфейс MySQL C API
Глава 7. Программный интерфейс Perl API
Глава 8. Интерфейс API для языка написания сценариев PHP

5

Введение в программирование в СУБД MySQL

В этой главе...

Написание программ, работающих с СУБД MySQL
Интерфейсы API для СУБД MySQL
Выбор API-интерфейса

В этой главе рассказывается, в чем заключается преимущество написания собственных программ, работающих с СУБД MySQL, перед использованием стандартных клиентских программ, включенных в дистрибутив СУБД MySQL. Кроме того, представлен концептуальный обзор интерфейсов для трех языков, сущность которых будет изложена в следующих главах (C, Perl и PHP), и обсуждаются некоторые факторы, которые необходимо принимать во внимание при выборе языка программирования.

Написание программ, работающих с СУБД MySQL

СУБД MySQL поставляется с набором программных программ. Например, программа `mysqldump` экспортирует содержимое и структуры таблиц, программа `mysqlimport` загружает данные из файлов в таблицы, программа `mysqladmin` выполняет операции администрирования, программа `mysql` позволяет взаимодействовать с сервером и выполнять произвольные запросы. Каждая программа была задумана как маленькая программа с узкоспециализированными функциями. Это справедливо и для более гибкой программы `mysql` в том смысле, что с ее по-

мощью можно выполнить любое количество различных запросов. Цель ее – делать SQL-запросы и просматривать полученные результаты.

Ограниченная природа клиентской программы СУБД MySQL не является недостатком – так было задумано при проектировании. Эта программа является программой общего назначения и не должна быть универсальным механизмом. Разработчики СУБД MySQL не могут подписаться под философией написания громадных и неповоротливых программ, выполняющих любой ваш каприз (это обычно заканчивается тем, что программируется множество функций, которые никому никогда не потребуются).

Стандартные клиентские программы помогают справиться с большинством общих задач, с которыми приходится сталкиваться пользователям MySQL. Тем не менее к приложениям иногда выдвигаются требования, которые нельзя удовлетворить средствами стандартного клиента. Этим целям служит библиотека программирования клиентов. Она позволяет создавать программы и предоставляет программисту достаточно гибкий инструмент, позволяющий удовлетворить любое требование заказчика. Библиотека клиента открывает широкие возможности, ограниченные только вашим воображением.

Какие же специальные возможности вы получите, написав свои собственные программы? Давайте исследуем этот вопрос, сравнивая возможности клиента `mysql` и его интерфейса с возможностями библиотеки.

- **Настройка процедуры ввода данных.** Клиент `mysql` позволяет вводить только обычные операторы SQL. Программа "собственного изготовления" позволит создать более понятный и простой интерфейс для ввода данных. Такая программа снимает необходимость знать язык SQL и роль базы данных в выполнении задачи. Процедура ввода может представлять собой нечто "рудиментарное", такое как цикл "подсказка–ответ" или что-то более изощренное. Например, ввод в экранной форме, реализованный с помощью таких пакетов управления экранами, как "curses" или "S-Lang", "X-Window", с использованием Tcl/Tk или формы Web-броузера.

  Большинство пользователей операторов `SELECT` предпочтут обычному вводу выбор параметров поиска из формы. Например, агент по продаже недвижимости хочет найти дома определенного проекта, цена которых лежит в определенном диапазоне, удовлетворяющие определенному местоположению, задав при этом параметры поиска и получив в ответ только искомые варианты. Аналогичные соображения применимы и при добавлении новых записей или изменении существующих. Наборщица, вводящая данные в отделе ввода данных, не должна ничего знать о синтаксисе операторов `INSERT`, `REPLACE` и `UPDATE`.

  Дополнительным доводом в пользу создания промежуточного этапа ввода между конечным пользователем и сервером СУБД MySQL является то, что на него можно возложить задачу проверки данных, введенных пользователем. Например, можно сделать проверку введенных календарных дат, чтобы убедиться, что они удовлетворяют формату календарных данных, принятому в СУБД MySQL.

  Некоторые приложения могут вообще работать без вмешательства пользователей, как это бывает в случае, когда ввод в СУБД MySQL генерируется другой программой. Может потребоваться настроить Web-сервер таким об-

разом, чтобы он производил запись регистрационных данных не в файл, а базу данных MySQL. Программа мониторинга системы может быть установлена таким образом, чтобы периодически заносить информацию о статусе в базу данных.

- **Возможность форматирования вывода.** Программа `mysql` выводит результаты в неформатированном виде, если таковым не считать имеющийся формат из столбцов, разделенных символом табуляции, или табличную форму. Есть еще один случай, когда требуется специальный формат, — полученные данные можно отформатировать самостоятельно. Здесь диапазон новаторских решений лежит от такого простого решения, как напечатать "отсутствует" вместо NULL, до более сложных требований к отчетам. Рассмотрим следующий отчет:

```
State       City          Sales
-------------------------------------
AZ          Mesa          $94,384.24
                Phoenix       $17,328.28
                ------------------------
                Subtotal      $117,712.52
-------------------------------------
CA          Los Angeles   $118,198.18
                Oakland       $38,838.36
                ------------------------
                Subtotal      $157,036.54
=====================================
                TOTAL         $274,749.06
```

Этот отчет содержит несколько специализированных элементов.

- Настроенные заголовки.
- Подавление вывода повторяющихся значений с таким расчетом, чтобы значение печаталось только при его изменении.
- Вычисление промежуточных и общих итогов.
- Форматирование чисел, таких как 94384.24, в виде денежных сумм $94,384.24.

Другим общим типом задач, включающих сложное форматирование, является создание счетов-фактур, где заголовок каждого счета-фактуры должен быть связан с информацией о покупателе, а деталировка — с информацией о каждом заказанном наименовании. Такой тип отчета может однозначно выходить за рамки возможностей форматирования, имеющихся у `mysql`.

При решении задач определенного типа вывод результатов может вообще не потребоваться. Предположим, что в одной таблице требуется найти данные, которые будут использоваться при вычислении результата. В свою очередь, полученный результат будет добавлен в другую таблицу. Может понадобиться перенаправить вывод результатов куда-то на другое устройство, отличное от экрана пользователя, запустившего запрос. Например, задача выборки имен и адресов электронной почты для автоматического процесса генерации писем для рассылки по почте генерирует вывод, содержащий сообщения, рассылаемые по электронной почте, а не тому, кто непосредственно работает с программой.

- **Возможность обойти ограничения, вытекающие из самой природы SQL.** Язык SQL не является процедурным языком и не имеет набора структур управления выполнением программ, таких как условные операторы, циклы и подпрограммы. Сценарии SQL состоят из набора операторов, выполняемых одновременно от начала до конца с минимальной проверкой на ошибки.

  При запуске запросов SQL с помощью программы `mysql` в пакетном режиме `mysql` завершит выполнение после первой же ошибки или, если `mysql` была запущена с ключом `--force`, выполнит все запросы, сколько бы ошибок не появилось. Программа может обеспечить управление выполнением операторов и, таким образом, выполняться избирательно в зависимости от успешного или неуспешного выполнения запроса. Более того, в зависимости от успешности или ошибочности выполнения одних операторов SQL можно передавать управление другим операторам SQL.

  Язык SQL не имеет достаточно хорошего механизма передачи данных, что унаследовано и программой `mysql`. Очень затруднительно использовать результаты одного запроса в другом запросе или объединять результаты, полученные из нескольких запросов.

  В качестве редкого исключения можно назвать функцию `LAST_INSERT_ID()`, которая применяется для получения последнего сгенерированного значения в столбце с атрибутом `AUTO_INCREMENT`.

  В общем, можно сказать, что очень сложно задать выборку записей, когда каждая из них лежит в основе сложных последующих операций. Например, поиск списка покупателей с последующим просмотром детальной кредитной истории каждого может потребовать несколько запросов о покупателе. Программа `mysql` с выполнением задач такого типа не справится, потому что для их решения необходимо создавать ряд запросов, которые зависят от результатов предыдущих запросов и выходят за пределы форматирующих возможностей `mysql`.

  В целом с определенной долей уверенности можно сказать, что для решения задач, заключающихся в выяснении отношений "заголовок-деталировка", и задач, требующих форматированного вывода, требуется инструментарий, предоставляющий большие возможности, чем `mysql`. Это должна быть программа, "склеивающая" запросы вместе и позволяющая использовать результаты одного запроса в качестве входных данных другого.

- **СУБД MySQL можно интегрировать в любое приложение.** Многие программы сильно выиграют от возможности использования данных из баз данных. Приложение, которому требуется проверить идентификационный номер покупателя или наличие товара на складе, может это сделать с помощью короткого запроса. Web-приложение, предназначенное для выборки книг определенного автора, может сделать это посредством запроса из базы данных и затем предоставить полученные результаты на экран броузера клиента.

  Этого можно добиться с помощью "интеграции" с использованием возможностей сценария оболочки, вызывающего `mysql` с файлом, содержащим операторы SQL, а затем обрабатывающего полученные результаты с помощью других утилит ОС UNIX. Но результат может получиться не очень кра-

сивым, особенно по мере усложнения задачи. При доработке приложения может создаться впечатление, что оно "работает хорошо, но на самом деле что-то не так". Кроме того, сами проблемы создания сценария оболочки, работающей для запуска других команд, могут вас мало интересовать. Вполне может оказаться, что непосредственное взаимодействие с сервером СУБД MySQL и вызов того, что требуется в каждой фазе выполнения, будет гораздо эффективнее.

В главе 1, "Знакомство с СУБД MySQL и SQL", по отношению к тестовой базе данных `sampdb` были намечены некие цели, которые могут оказаться причиной того, что для взаимодействия с базой данных необходимо написать специальную программу. Некоторые из этих целей указаны в следующем списке.

- Получить список членов "Исторической Лиги" в формате, пригодном для печати.
- Сделать доступным список на Web-узле.
- Рассылать уведомления о необходимости обновления членства по электронной почте.
- Упростить ввод оценок с помощью Web-броузера.

Возможность интегрирования СУБД MySQL в Internet будет детально обсуждена в дальнейшем. Сама СУБД MySQL не предоставляет прямой поддержки Web-приложений, но с помощью определенного инструментария базы данных могут стать легкодоступными из Internet. Запросы можно формировать с помощью Web-сервера и возвращать результаты на броузер клиента.

Существуют две дополнительные перспективы объединения СУБД MySQL и Web-сети.

- **Использование Web-сервера для обеспечения полного доступа к СУБД MySQL.** В фокусе внимания разработчика лежит его база данных, а Internet предоставляет собой лишь инструмент, обеспечивающий простой и надежный доступ к данным. Место базы данных в таком сценарии очевидно потому, что она лежит в фокусе ваших интересов. Например, можно создать Web-страницы, которые позволят просмотреть содержимое и структуру базы данных.
- **Применение СУБД MySQL для расширения возможностей Web-сервера.** В фокусе ваших интересов лежит Web-узел, и вы хотите расширить его возможности с помощью СУБД MySQL. Воспользоваться возможностями СУБД MySQL можно, например, в качестве инструмента ведения доски объявлений или дискуссионного стола. В этом случае роль базы данных достаточно скрыта, и посетители узла могут даже не знать о том, что СУБД MySQL вообще играет какую-то роль в предоставляемых узлом услугах. В таком случае СУБД MySQL используется для расширения возможностей Web-сервера.

Эти два подхода нельзя назвать взаимоисключающими. Например, в сценарии "Историческая Лига" для предоставления ее членам простого доступа к содержимому списка членов "Исторической Лиги" через Internet планируется использо-

вать Web-сервер. Здесь налицо использование возможностей Web-сети для предоставления доступа к базе данных. Добавление возможности просмотра списка повысит популярность узла у членов "Исторической Лиги". Это значит, что использование базы данных расширит перечень услуг, предоставляемых узлом.

Независимо от того, как вам видится интеграция своей СУБД MySQL с Web, техническое решение будет выглядеть следующим образом. Соединение внешнего интерфейса вашего Web-узла с СУБД MySQL осуществляется через Web-сервер. Web-сервер посылает запрос от пользователя на сервер СУБД MySQL, получает результаты и посылает их клиенту обратно на его броузер.

При этом совсем не обязательно делать данные общедоступными, но в этом есть свои преимущества, особенно по сравнению с доступом к данным через стандартную клиентскую программу СУБД MySQL.

- Пользователи, вводящие данные через Internet, могут это делать с помощью любого броузера и на любой платформе. В своем выборе они не ограничены только системами, на которых работает СУБД MySQL. Независимо от того, как распространены клиенты СУБД MySQL и на каких платформах они могут работать, броузеры все равно имеют более широкое распространение.
- Web-интерфейс может быть проще, чем автономный клиент СУБД MySQL с управлением с помощью командной строки.
- Web-интерфейс адаптируется к требованиям любого приложения. Клиенты СУБД MySQL являются инструментарием общего назначения с фиксированным интерфейсом.
- Динамические Web-страницы расширяют возможности СУБД MySQL до пределов, которые реализовать с помощью обычного клиента СУБД MySQL уже трудно, а порой невозможно. Например, нельзя совместить приложение, которое обрабатывает карточку покупателя, только с помощью клиента MySQL.

Для создания Web-приложений можно воспользоваться любым языком программирования, но некоторые из них подходят для этих целей больше других. И об этом вы узнаете ниже, в разделе "Выбор API-интерфейса".

Интерфейсы API для СУБД MySQL

Для облегчения разработки программных приложений СУБД MySQL имеет библиотеку, написанную на языке программирования C, позволяющую производить доступ к базам данных из любой программы, написанной на языке C. Клиентская библиотека предоставляет программный интерфейс для приложений (application programming interface – API), который обеспечивает соединение и обмен данными между клиентом и сервером.

Однако разработчик в качестве средства разработки программного обеспечения не ограничен одним только языком C. Многие языковые процессоры или сами написаны на языке C, или имеют возможность подключения библиотек C. Таким образом, клиентская библиотека СУБД MySQL предоставляет вышеупомянутое соединение и для таких языков. Это предоставляет широкое поле деятельности для создания приложений, способных работать с сервером MySQL. Существуют клиентские API для таких языков, как Perl, PHP, Java, Python, C++, Tcl и др. Также име-

ются интерфейсы для языка Java (хотя при этом не используется библиотека C, а непосредственно используется протокол клиент/сервер). Время от времени создаются новые API, поэтому действительный на данный момент перечень интерфейсов можно найти на страницах справочного руководства по СУБД MySQL или на Web-узле MySQL (`http://www.mysql.com/portal/development/html/`).

Подключение каждого языка характеризуется своим собственным интерфейсом, определяющим правила доступа к базам данных СУБД MySQL. Объем данного издания не позволяет детально рассмотреть все имеющиеся интерфейсы, поэтому мы остановимся на трех из них, которые являются самыми популярными.

- **Библиотека для клиента C.** Это самый первый и основной интерфейс СУБД MySQL. Например, он использовался для создания стандартных клиентов дистрибутива СУБД MySQL: `mysql`, `mysqladmin` и `mysqldump`.
- **DBI (Database Interface — интерфейс базы данных) API для универсального языка написания сценариев Perl.** DBI реализован как модуль Perl, который осуществляет интерфейс с другими модулями на уровне DBD (драйверов баз данных), обеспечивающем доступ к определенному механизму базы данных. (Остановимся, в частности, на модуле DBD, который поддерживает доступ к СУБД MySQL.) Наиболее традиционным использованием DBI с СУБД MySQL является создание автономных клиентов, активизируемых командной строкой, и сценариев, которые запускаются Web-сервером для того, чтобы обеспечить доступ к СУБД MySQL из Internet.
- **PHP API.** PHP представляет собой язык написания сценариев на стороне сервера, который является удобным способом включения программ в тело Web-страницы. Такая страница обрабатывается PHP до отсылки клиенту, что позволяет этим сценариям создавать динамические Web-страницы, включающие результат SQL-запроса к базе данных. Термин "PHP" изначально означает Personal Home Page (персональная домашняя страница), но функционально PHP уже давно перерос свое первоначально назначение. Web-узел типа PHP сейчас называется "PHP: гипертекстовый препроцессор". Как и DBI, PHP, кроме доступа к СУБД MySQL, обеспечивает поддержку доступа к нескольким другим СУБД.

Каждый из этих API будет подробно описан в соответствующих главах. В настоящей главе приведен сравнительный обзор этих интерфейсов с описанием их общих характеристик, чтобы дать общее представление о них.

Конечно, совсем не обязательно замыкаться на каком-то одном API. Изучив плюсы и минусы каждого API, мы вооружаем себя знанием, которое позволит сделать осознанный выбор. Более того, при разработке большого проекта, содержащего несколько компонентов, можно воспользоваться несколькими API и создать разные части проекта на различных языках программирования, в зависимости от того, какой больше всего подходит на данном этапе. Если у разработчика есть время, может оказаться полезным реализовать приложение в нескольких вариантах одновременно. Таким образом, можно приобрести бесценный опыт работы с различными API.

В приложении A, "Получение и инсталляция программного обеспечения", вы узнаете, где можно приобрести программное обеспечение, необходимое для про-

ерки на практике любого API. Кроме того, вы можете обратиться к другим кни-ам, в которых описывается СУБД MySQL.

Вот две из них, с которыми автор знаком ближе всего (прежде всего потому, то он их написал сам), — это *MySQL and Perl for the Web* (New Riders, 2001) и *MySQL ookbook* (O'Reilly, 2002). В первой из них есть достаточно полное описание использования СУБД MySQL в Web-среде. Во второй обсуждаются Perl и PHP, а акже показано, как написать программы MySQL, пользуясь интерфейсом DB-API зыка Python и интерфейсом языка Java JDBC.

Предшественники DBI и PHP

Предшественником интерфейса DBI для языка Perl является модуль интерфейса между СУБД MySQL и языком Perl, `Mysqlperl`. Этот модуль больше не поддерживается, и в новых разработках на базе СУБД MySQL им пользоваться не следует. По одной простой причине: `Mysqlperl` рассчитан на работу с Mysql, а DBI — нет. Если разработчик решает использовать Perl-приложения, созданные для СУБД MySQL, в работе с другими СУБД, значительно проще перенести сценарии DBI, чем сценарии `Mysqlperl`, потому что они менее зависимы от конкретного механизма базы данных. (DBI можно пользоваться даже в том случае, если у разработчика в распоряжении имеется сценарий Perl, разработанный для доступа к базам данных СУБД MySQL, и оказывается, что он разработан для `Mysqlperl`, а не для DBI. DBI может быть настроен таким образом, чтобы включать поддержку эмуляции `Mysqlperl`. Поэтому нет необходимости устанавливать оба пакета.)

Предшественником PHP является PHP/FI 2.0 (FI происходит от термина "интерпретатор форм"). Как и `Mysqlperl`, интерфейс PHP/FI уже является достоянием истории, и я не хочу возвращаться больше к его обсуждению.

Интерфейс API для языка C

Интерфейс API для языка C используется в контексте компилированных -программ. Эта клиентская библиотека обеспечивает интерфейс самого низкого з имеющихся уровней обмена данных с сервером СУБД MySQL, предоставляю-ий все возможности для соединения и обмена данными с сервером.

Все клиенты языка C, поставляемые в дистрибутиве СУБД MySQL, основаны а этом интерфейсе. Клиентская библиотека для языка C служит основой для одключения СУБД MySQL к другим языкам программирования, за исключением нтерфейса для языка Java. Например, драйверы СУБД MySQL для модуля Perl)BI и процессора PHP могут работать с СУБД MySQL только благодаря связям, :тановливаемым с помощью клиентской библиотеки СУБД MySQL языка C. Например, драйвер СУБД MySQL для модуля Perl DBI и PHP-кода "знает" о подключении к СУБД MySQL, если к коду подключена клиентская C-библиотека для :УБД MySQL.

Происхождение API-интерфейса C для СУБД Mysql

Если у вас имеется опыт написания программ для СУБД mSQL, вы обязательно обратите внимание, что интерфейс API C для СУБД MySQL во многом аналогичен интерфейсу API C для СУБД mSQL. Разработчики приложения для СУБД MySQL могут воспользоваться массой бесплатных утилит для СУБД mSQL, имеющихся в наличии. Для того чтобы перенести эти утилиты с минимальными затратами, интерфейс API для СУБД MySQL был преднамеренно разработан подобным интерфейсу API C для СУБД mSQL. (СУБД MySQL даже поставляется со сценарием `msql2mysql`, который производит простейшие текстовые преобразования имен функций интерфейса API mSQL в соответствующие имена функций интерфейса API MySQL. Эта операция относительно тривиальная, но очень помогает в процессе преобразования программ, созданных для работы с СУБД mSQL, для продления их "жизни" на платформе СУБД MySQL.)

Интерфейс Perl DBI API

Интерфейс DBI API используется в контексте приложений, написанных на языке Perl. Этот интерфейс является наиболее универсальным интерфейсом изо всех трех, рассматриваемых нами, поскольку он может работать со многими базами данных и в то же время скрывает (насколько это возможно для инструмента написания сценариев) детали, характерные для конкретных баз данных. DBI реализован с помощью модулей Perl, в которых используется двухуровневая архитектура (рис. 5.1).

- Уровень DBD (драйверов баз данных) обеспечивает поддержку различных баз данных. Он реализован с помощью драйверов, которые настроены на механизм определенной базы данных.
- Уровень DBI (интерфейса баз данных) обеспечивает интерфейс для сценариев клиента. Этот уровень абстрагирован от конкретного механизма базы данных. Модуль уровня DBD, который реализует поддержку DBI для СУБД MySQL, называется `DBD::mysql`. Этот модуль изначально назывался `Msql-Mysql-modules`, так как сначала он был создан для работы с СУБД mSQL, а затем его функциональность была расширена до СУБД MySQL. Настоящее название `DBD::mysql` свидетельствует о том, что СУБД MySQL уже "затмил" по своей популярности СУБД mSQL.

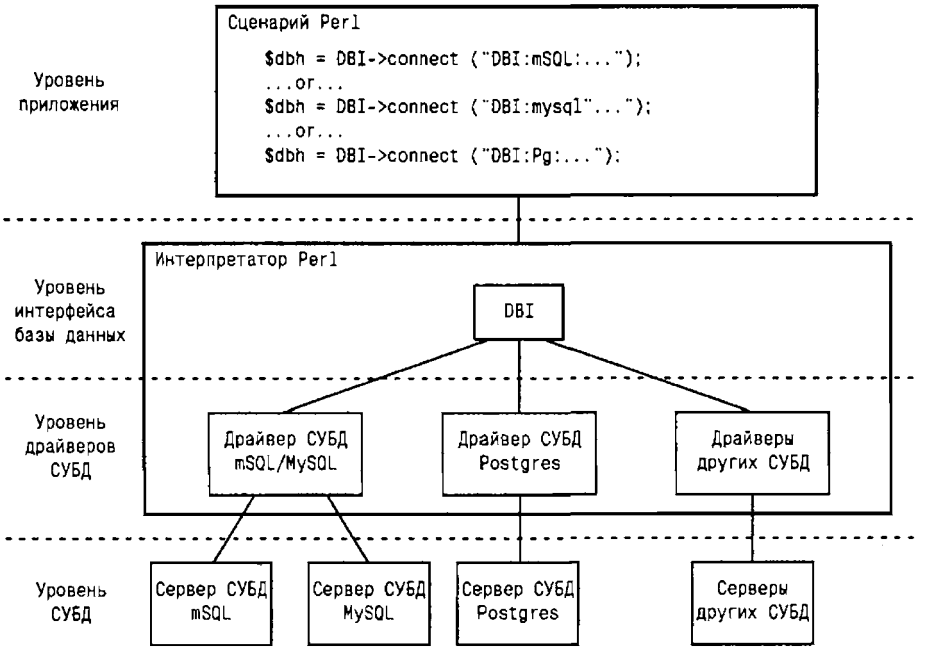


Рис. 5.1. Архитектура интерфейса DBI

Архитектура DBI-интерфейса позволяет создавать приложения в относительно традиционной манере. При написании сценария DBI используется стандартный набор вызовов. Уровень DBI вызывает соответствующий драйвер уровня DBD для обработки запросов, а уже драйвер возлагает на себя решение всех про-

блем, связанных с соединением с сервером определенной базы данных. Данные, полученные от сервера, возвращаются обратно на уровень DBD, а затем через уровень DBI возвращаются приложению. Целостность данных не зависит от того, из какой базы данных они были получены.

В результате имеем интерфейс, скрывающий от разработчика приложения различия между механизмами баз данных, но работающий с широким диапазоном баз данных (настолько большим, насколько хватает драйверов). Интерфейс DBI обеспечивает надежный интерфейс клиентского рабочего места и улучшает переносимость, позволяющую осуществлять доступ к базам данных в унифицированном формате.

Единственная особенность, отражающая специфику базы данных, проявляется в момент открытия базы данных. Во время установки связи необходимо указывать, с помощью какого драйвера она будет осуществляться. Например, при работе с СУБД MySQL подключение необходимо производить следующим образом:

```
$dbh = DBI -> connect ("DBI:mysql:... ");
```

Для работы с СУБД Postgres или mSQL при подключении необходимо использовать следующие команды:

```
$dbh = DBI -> connect ("DBI:Pg:... ");
$dbh = DBI -> connect ("DBI:mSQL:... ");
```

После подключения какого-либо упоминания о драйвере не нужно. Вся обработка специфических нюансов конкретной базы данных полностью возложена на интерфейс DBI и драйвер.

Однако это все теория. На практике против переносимости сценариев DBI работают два фактора.

- Различные механизмы баз данных работают с различными диалектами языка SQL. И велика вероятность того, что SQL-запрос, написанный для одной базы данных, не будет "понятен" другой. Если вы придерживаетесь традиционного стиля написания SQL-запросов, то существует высокий процент вероятности, что переносимость будет обеспечена. Но если определенный диалект SQL зависит от механизма базы данных, то сценарии тоже будут зависеть от него. Например, сценарий, использующий оператор SHOW TABLES и присущий только диалекту языка SQL MySQL, не будет работать с другими базами данных.
- Модули драйверов баз данных содержат информацию, зависящую от конкретной базы данных. Это позволяет разработчикам эффективно использовать особенности конкретной базы данных. Например, драйвер СУБД MySQL обеспечивает метод доступа к столбцам в результатах запросов, таких как максимальная длина каждого столбца, являются ли столбцы числовыми, и т.д. Общеизвестно, что эти возможности имеют какой-либо аналог в других базах данных. Такие возможности драйверов являются существенной преградой самой идеологии переносимости. Используя такие возможности при разработке сценария, разработчик должен четко осознавать, что он при этом теряет возможность использовать этот сценарий при работе с другими СУБД[1].

[1] Тем не менее автор продемонстрирует в главе 7, что вполне возможно избежать конструкций, характерных только для конкретной СУБД MySQL. Так что выбор целиком лежит на совести разработчика – решать ему, использовать такие конструкции в своей работе или нет.

Несмотря на потенциал этих двух факторов, при создании сценариев, зависимых от конкретных СУБД, механизм DBI обеспечения доступа к базе данных абстрактного типа является существенным средством достижения переносимости. Это уже задача разработчика, решать, какие преимущества вы хотите извлечь из такого механизма. Как вы узнаете в главе 7, "Программный интерфейс Perl API", можно достаточно легко избежать конструкций, присущих только СУБД MySQL. Это связано с тем, что природа данных конструкций позволяет принять решение использовать их или окончательно отказаться от их использования.

Семантика терминов DBI и DBD

Несмотря на то что уровень DBI (independent) не зависит от СУБД, а уровень DBD (dependent) зависит, это не означает, что в этом кроется смысл аббревиатур DBI и DBD. Эти аббревиатуры означают всего лишь "интерфейс баз данных" и "драйвер баз данных".

Интерфейс PHP API

Как и Perl, язык PHP является языком создания сценариев, но он создавался скорее не как язык общего назначения, а как язык, основной задачей которого является создание Web-приложений. Интерфейс PHP API используется в первую очередь в качестве средства включения сценариев на Web-страницы, что упрощает задачу разработчиков Web-узлов по созданию динамических Web-страниц. Сам обмен происходит следующим образом: клиентский броузер посылает запрос о PHP-странице на Web-сервер, PHP выполняет любой найденный на странице сценарий и замещает его полученным результатом, который отсылается обратно броузеру. Это позволяет странице, которая реально отображается на экране броузера, изменяться в зависимости от обстоятельств, при которых она была вызвана. Например, включение такого короткого PHP-сценария в тело Web-страницы приведет к отображению IP-адреса узла, запросившего страницу:

```
<?php echo $REMOTE_ADDR; ?>
```

А вот менее тривиальная и более интересная программа, которая будет давать поминутную информацию о посетителях на основании информации, содержащейся в базе данных. Следующий пример демонстрирует простейший сценарий, который может найти свое применение на Web-узле "Исторической Лиги". Этот сценарий генерирует запрос, определяющий текущее количество действительных членов, и сообщает его посетителю. (Если происходит ошибка, сценарий ничего не сообщает.)

```
<html>
<head>
<title>Историческая Лига США</title>
</head>
<body bgcolor="white">
<p>Добро пожаловать на Web-узел "Исторической Лиги".</p>
<?php
# Домашняя страница Web-узла "Исторической Лиги"
$link = @mysql_pconnect ("cobra.snake.net", "sampadm", "secret")
or exit();
mysql_select_db("sampdb")
or exit ();
$result = mysql_query("SELECT COUNT(*) FROM member")
```

```
or exit();
if ($row = mysql_fetch_row ($result))
echo "<p>В "Исторической Лиге" зарегистрировано" . $row[0] .
"членов</p>";
mysql_free_result ($result);
?>
</body>
</html>
```

Сценарии PHP, как правило, выглядят как обычные HTML-страницы, но заключенные внутри тэгов <?php и ?>. Страница может содержать сразу несколько сценариев. Это обеспечивает предельно гибкий подход к разработке сценариев. Например, можно создать обычную HTML-страницу, в которую позднее будет вставлен сценарий.

Языку PHP не нужно "делать усилий" для унификации интерфейса с различными базами данных, подобно интерфейсу DBI. Наоборот, интерфейс к каждой базе данных представляет собой интерфейс с соответствующей библиотекой C, осуществляющий API-интерфейс низкого уровня с любой базой данных. Например, имена функций языка PHP, предназначенных для доступа к базам данных MySQL из сценариев PHP, очень похожи на имена и функции клиентской библиотеки языка C. (Если вы предпочитаете DBI-подобный подход, можно воспользоваться расширением PHP, под названием PEAR, с дополняемым репозитарием. PEAR — это дополнение к PHP, которое имеет в своем составе модуль, обеспечивающий более абстрактный интерфейс к базам данных с использованием двухуровневой архитектуры, аналогичной той, что используется DBI. Подробности можно узнать на Web-узле `pear.php.net`.)

Выбор API-интерфейса

В этом разделе вы найдете общие рекомендации, имеющие своей целью помочь в выборе API-интерфейса на этапе разработки приложений. Здесь также приводится сравнительная характеристика трех API-интерфейсов — C, DBI и PHP. Кроме того, даются рекомендации, в каких ситуациях можно остановить свой выбор на том или ином варианте.

В своей работе автор использует все языки, хотя есть такие, которым он отдает предпочтение. Читатели тоже могут иметь свой круг предпочтений, как это имело место у рецензентов данной книги. Один из них настаивал на том, что автор должен акцентировать внимание на важности языка C при программировании задач, работающих с СУБД MySQL, в то время как другой считал, что автор должен всячески предостерегать читателя от использования языка C в программировании и всячески умалять его значение! Я придерживаюсь такой точки зрения, что читатель сам должен взвесить все "за" и "против" и сделать свои собственные заключения.

В процессе выбора типа интерфейса API учитываются многие факторы.

- **Среда, в которой будет выполняться задача.** Контекст, в котором будет выполняться приложение.
- **Производительность.** Какова эффективность приложений, написанных с применением того или иного интерфейса?

- **Простота разработки.** Насколько удобно работать с интерфейсом и его языком при создании приложений?
- **Переносимость.** Будут ли использоваться приложения в работе с другими базами данных?

Все эти факторы будут детально изложены ниже. Помните, что некоторые факторы имеют взаимное влияние. Например, при разработке приложения, которое должно обладать хорошей производительностью, не менее важным фактором является и быстрота разработки приложения. Поэтому предпочтение отдается языку, позволяющему быстро разрабатывать приложения, эффективность работы которых не будет достаточно высокой. Но оба эти фактора являются взаимоисключающими.

Среда выполнения

Разработчик приложения всегда знает, в какой среде будет работать приложение. Например, это может быть программа генерации отчетов, которая будет вызываться из оболочки, или программа, суммирующая задолженность, которая запускается утилитой `cron` в конце каждого месяца. Обычно команды, запускающиеся из оболочки или из утилиты `cron`, не имеют каких-то особенных требований к среде, в которой они выполняются. В качестве другого примера можно привести приложение, которое будет запускаться Web-сервером. Такая программа должна "уметь вытягивать" информацию специфического типа из своей рабочей среды. Каким броузером пользуется клиент? Какие параметры заносятся в форму списка рассылки при запросе на подписку? Вводит ли клиент правильный пароль для того, чтобы получить доступ к информации о персонале?

Все языки API обладают различными характеристиками.

- Язык C является языком программирования общего назначения. Отсюда следует, что его можно использовать для решения самых разнообразных задач. Есть тенденция использования C для написания автономных программ, а не для программирования Web-узлов. Вероятно, одной из причин является то, что язык программирования C не настолько удобен для обработки текстов и управления памяти, как языки Perl и PHP, которые имеют тенденцию активно использоваться для программирования Web-приложений.
- Язык Perl, как и C, отлично подходит для написания автономных программ. Однако в отношении языка Perl справедливо и то, что он подходит и для разработки Web-узлов, например с помощью модуля CGI.pm. Это делает язык Perl очень удобным инструментом для создания приложений, связывающих СУБД MySQL с Internet. Такая программа может подключаться к Internet через модуль CGI.pm, взаимодействуя при этом через интерфейс DBI с СУБД MySQL.
- Язык PHP разрабатывался как инструмент разработки Web-приложений, поэтому является средой, в которой его использование наиболее предпочтительно. В то же время доступ к базам данных является его самым большим преимуществом. Таким образом, его совершенно естественно использовать

в приложениях, осуществляющих доступ к базам данных через Internet. Конечно, можно использовать язык PHP и в роли автономного интерпретатора (например, для выполнения сценариев из оболочки), но он не часто используется в таком качестве.

Принимая во внимание эти доводы, можно сделать вывод, что языки C и Perl являются наиболее вероятными кандидатами для создания автономной программы. Для написания Web-приложений больше всего подходят языки PHP и Perl. Если необходимо написать оба типа приложений, но вы не знаете ни одного из этих языков и не хотите их подробно изучать, наилучшим выбором будет язык Perl.

Производительность

При прочих равных условиях мы будем стремиться к тому, чтобы приложение работало как можно быстрее. Однако реальное влияние фактора производительности имеет тенденцию зависеть от частоты использования программы. Для программ, которые запускаются процессом `cron` ночью один раз в месяц, фактор производительности не будет очень критичным. Но когда программа запускается на перегруженном Web-узле несколько раз в секунду, каждая миллисекунда выигрыша важна. В последнем случае производительность может сыграть существенную роль в отдаче, получаемой от вашего Web-узла, и его популярности. Независимо от своей направленности медленный узел раздражает посетителей. И если Web-узел является источником вашего дохода, уменьшение производительности сразу же ударит по вашему карману. Узел не будет в состоянии одновременно обслужить много запросов, и посетители просто перестанут посещать его.

Оценка производительности является сложной задачей. Лучшим индикатором того, как приложение производит операцию записи при использовании определенного интерфейса, будет создание тестового примера, работающего с этим API. Но лучшим сравнительным тестом будет создание тестов для различных API и сравнение их работы. Наверняка это не даст полной картины. Но чаще всего необходимо просто начать писать программу. После того как она заработает, уже можно будет подумать, каким образом ее усовершенствовать, чтобы она заработала быстрее, потребляла меньше памяти или была оптимизирована по какому-то другому показателю. По крайней мере, существуют два фактора, на которые можно положиться как на факторы, имеющие наибольшее влияние на производительность:

- откомпилированные программы работают быстрее интерпретируемых сценариев;
- скорость выполнения у интерпретируемых языков, используемых в Web-контексте, выше, когда интерпретатор вызывается как модуль, являющийся частью самого Web-сервера, а не отдельным процессом.

Сравнение компилируемых и интерпретируемых языков

С большой долей вероятности можно сказать, что скомпилированные программы эффективнее: они потребляют меньше памяти и выполняются быстрее, чем их аналоги, созданные на языках написания сценариев. Это происходит из-за повышенных накладных расходов, которые требуются для работы интерпретато-

ра. Язык C является языком компилируемого типа, а языки Perl и PHP являются языками интерпретируемого типа, поэтому программы, написанные на языке C, обычно работают быстрее сценариев, написанных на языках Perl и PHP.

Имеют место и факторы, которые сглаживают эти различия. Написание программ на языке C позволяет получить более быстрые программы, но существует вероятность, что написанная программа будет неэффективной. Написание программы с помощью компилируемого языка программирования само по себе не является полной гарантией его более высокой производительности. По-прежнему нужно хорошо понимать то, что вы делаете. Кроме того, различие между скомпилированными и интерпретируемыми программами сглаживается, если приложение проводит большую часть времени, выполняя код из библиотеки клиента СУБД MySQL, которая подключена к механизму интерпретатора.

Сравнение характеристик автономной и модульной версии интерпретаторов языка

В Web-приложениях используются два вида интерпретаторов языков написания сценариев. Это справедливо и для Web-сервера Apache, который будет использоваться нами при написании Web-приложений.

- Web-сервер имеет соответствующие настройки для вызова интерпретатора как отдельного процесса. Когда Web-серверу Apache необходимо запустить какой-нибудь сценарий PHP или Perl, он запускает соответствующую программу и передает ей этот сценарий на выполнение. В этом случае Web-сервер Apache взаимодействует с интерпретаторами, как с CGI-программами, т.е. он осуществляет с ними обмен данными по протоколу Common Gateway Interface (CGI).
- Интерпретатор может быть и модулем, непосредственно подключенным к двоичному коду Apache, работающим как часть процесса Apache. В терминах сервера Apache интерпретаторы Perl и PHP являются модулями `mod_perl` и `mod_php`.

Сторонники Perl и PHP могут спорить относительно преимуществ в скорости их любимого интерпретатора, но с тем, что форма работы интерпретатора является более важным фактором, существенно влияющим на производительность, согласятся все. Очевидно, что любой интерпретатор работает значительно быстрее в модульном режиме, чем в виде автономного CGI-приложения. В автономных приложениях интерпретатор необходимо запускать всякий раз, когда на выполнение приходит сценарий. Таким образом, вносится определенная лепта в перегрузку системы и повышается штраф на запуск интерпретатора. В модульной системе интерпретатор работает как часть уже работающего процесса сервера Apache, и возможности интерпретатора для Web-страниц будут доступны постоянно. Это существенно увеличивает производительность системы, снижает ее перегрузку и непосредственно отражается на возросшей возможности обрабатывать входящие запросы и на скорости отклика сервера.

Штраф на запуск автономного интерпретатора примерно на порядок снижает производительность по сравнению с модульной формой работы интерпретатора. Цена запуска интерпретатора сыграет огромную роль, если предпочтение будет отдаваться обслуживанию "быстрых" транзакций в совокупности с незначитель-

ными вычислениями, а не медленных с большим процентом обработки. Результатом того, что большая часть времени тратится на запуск интерпретатора, а не на выполнение сценария, будет бесполезная растрата ресурсов. Это аналогично ситуации, при которой работник проводит большую часть дня, готовясь к работе, приходит на работу в 4 часа дня и уходит домой в 5 часов.

Возможно, у вас возникнет вопрос, о какой экономии может идти речь, если по-прежнему требуется запускать Web-сервер Apache. А ответ лежит на поверхности — в момент запуска Apache автоматически создает пул порождаемых процессов, который будет в дальнейшем использован для обработки входящих запросов. Всякий вновь прибывший запрос, требующий обработки сценария, уже ожидает свой Apache-процесс. Конечно, каждый новый экземпляр служб сервера Apache умножает количество запросов, таким образом, цена запуска процесса может приниматься во внимание только для группы запросов, а не для одного запроса.

Какой из языков будет лучше работать в модульной форме (`mod_perl` и `mod_php`)? Это и по сей день является предметом споров, хотя они несколько и потеряли свою остроту после появления PHP 4. Язык Perl компилирует сценарий во внутренний скомпилированный формат, а язык PHP — нет. Таким образом, если сравнивать однажды скомпилированный сценарий, то Perl всегда будет работать быстрее, особенно если этот сценарий содержит циклы с большим количеством итераций. В PHP 4 используется значительно более быстрый интерпретатор Zend, в котором применяется модель "компилируй и выполняй", аналогичная Perl. Поэтому по возможности лучше всего использовать PHP 4. (Это справедливо не только из-за повышенной производительности PHP 4, но еще и потому, что в нем появились возможности, которых нет в PHP 3.)

При установке PHP собственными силами я настоятельно рекомендую отдавать предпочтение PHP 4. Даже в случае использования PHP 3, когда провайдер не представляет других вариантов, следует обратиться к провайдеру, чтобы он предоставил также возможность работать и с PHP 4. Одним из существенных различий между Perl и PHP является большее потребление памяти первым. Для работы модуля `mod_perl` потребуется больше памяти, чем для работы модуля `mod_php`. Процессы, создаваемые сервером Apache для модуля `mod_perl`, имеют больший объем, чем процессы, необходимые модулю `mod_php`. Это кроется в идеологии создания языка PHP как такового, который должен взаимодействовать с порождающим процессом и может быть активизирован и остановлен этим процессом несколько раз. Язык Perl был разработан, чтобы запускаться из командной строки как автономная программа, а не как язык, заключенный в процесс Web-сервера. Вероятно, в этом и кроется причина повышенного потребления памяти, ведь в модульном режиме Perl не работает в своей естественной среде. Другими факторами, влияющими на увеличенное потребление памяти, являются кэширование сценария и наличие дополнительных модулей Perl, которые он использует в работе. В обоих случаях программного кода на протяжении всего жизненного цикла процесса Web-сервера Apache в памяти сохраняется очень много кода. (Для минимизации этой проблемы есть специальные методы, которые позволяют назначать определенные процессы сервера Apache. Таким образом, дополнительный перерасход памяти будет присущ только процессам, которые выполняют сценарии Perl. Стратегии использования `mod_perl` Web-сервером Apache обсуждаются на Web-узле `http://perl.apache.org/docs/`.)

Автономная версия интерпретатора имеет одно преимущество над модульным вариантом: ее можно настроить для работы под различными идентификаторами пользователей. Модульные версии работают всегда под тем же пользователем, что и Web-сервер, который в целях безопасности всегда имеет минимальные привилегии. Это составляет определенные трудности для сценариев, которым требуются специальные привилегии (например, если требуется предоставить право чтения файлов, защищенных от записи). По необходимости можно совмещать модульный и автономный подход. По умолчанию пользуйтесь модульной версией, а автономные используйте в случае, когда сценарию требуется предоставить определенные права.

Подводя итог всему сказанному, можно сказать, что независимо от выбранного языка, нужно стремиться к тому, чтобы использовать модульный режим работы интерпретатора, а не автономный. "На откуп" автономной форме работы интерпретаторов можно оставить те задачи, которые невозможно обработать в модульном режиме, как это было в случае, когда запускаемому сценарию необходимо присвоить определенные привилегии. В таких случаях сценарий можно обработать с помощью механизма suEXEC, имеющегося у Web-сервера Apache. Он запускает интерпретатор под указанным идентификатором пользователя. (Новые возможности открывает использование Apache 2.x. Эта версия позволяет группировать сценарии и запускать их работу под управлением определенного пользователя или группы пользователей.)

Время разработки программ

Все описанные выше факторы влияют на производительность приложений. Но сама по себе эффективность выполнения программ не может быть единственной целью. Время разработчика тоже является существенным фактором, поэтому простота программирования также должна приниматься во внимание.

Написание сценария на языке Perl или PHP может занять вдвое меньше времени, чем написание аналогичного приложения на языке C. Поэтому предпочтение может быть отдано Perl DBI API, а не C API, даже принимая во внимание тот факт, что приложение не будет работать достаточно быстро. Для программ, которые будут запускаться не так часто, предпочтение обычно отдается тому инструментарию, который позволяет разрабатывать приложение в короткие сроки. Час времени разработчика сейчас стоит значительно дороже часа машинного времени!

В общем случае, языки написания сценариев позволяют написать действующую программу быстрее. Это особенно ценно при создании макетов будущих приложений. Здесь важны два фактора. Во-первых, языки написания сценариев имеют тенденцию к конструкциям более высокого уровня. Это позволяет разработчику мыслить на более высоком уровне абстракции (когда человек думает о том, что нужно сделать, а не о том, каким образом этого добиться). Например, такая конструкция, как ассоциативные массивы (хэши) языка Perl, позволяет сэкономить время на обработку данных за счет учета соотношений "ключ/значение" (в нашем примере соотношение "идентификатор учащегося/имя учащегося"). Язык C не имеет такой конструкции. Для того чтобы смоделировать такую конструкцию на C, необходимо написать и отладить программу, которая обработает множество деталей низкого уровня, касающихся таких проблем, как управление

памятью, манипуляция строками. Определенно, это потребует времени. Во-вторых, цикл разработки программ с помощью языков написания сценариев можно разделить на несколько этапов. Язык C предполагает обычный цикл "редактирование–компиляция–тестирование".

После каждого изменения программа перекомпилируется и только после этого тестируется. Разработка программного обеспечения с помощью языков Perl и PHP предполагает упрощенный цикл "редактирование–тестирование", потому что запуск сценария возможен сразу же после редактирования и безо всякого компилирования. Однако компилятор накладывает больше ограничений на программы в виде строгой проверки типов. Дисциплинирующее влияние компилятора позволяет избежать тривиальных ошибок, которых бывает трудно избежать в языках Perl и PHP. Компилятор C обязательно предупредит разработчика о наличии ошибки в имени переменной. Язык PHP такого предупреждения не сделает, а язык Perl сделает, только если разработчик об этом попросит. По мере увеличения объема разрабатываемой программы будет возрастать и вес этих ограничений.

В целом — это традиционная проблема выбора между компилирующими и интерпретирующими языками, заключающаяся в выборе между стоимостью разработки и ее производительностью. Даст ли разработка программы с помощью компилирующего языка программирования быстроту выполнения при больших затратах на разработку программы? Или программа будет разработана как интерпретирующий сценарий в кратчайшее время, но за счет эффективности выполнения?

Есть возможность совместить эти два подхода. Сначала программу можно написать в форме сценария как "черновой вариант". С его помощью можно проверить правильность работы алгоритма. Если программа доказала свою жизнеспособность и частота ее выполнения высока настолько, что на первый план выходит фактор производительности, программу можно переписать с помощью компилирующего языка программирования. Это позволит воспользоваться преимуществами обоих методов: быстрым получением макета и отличной производительностью конечного программного продукта.

В строгом смысле, интерфейсы Perl DBI и PHP API не предлагают таких новых возможностей, которых бы не было в клиентской библиотеке языка C. Это происходит потому, что оба интерфейса осуществляют доступ к СУБД MySQL через подключение библиотеки MySQL C к интерпретаторам. Однако среды, в которых реализуются возможности СУБД MySQL для языков C, Perl и PHP, существенно различаются. Рассмотрим некоторые ситуации, которые возникают при взаимодействии с сервером MySQL, и спросим себя, каким образом язык интерфейса может помочь нам в этом. Вот некоторые примеры.

- **Управление памятью.** В языке C задачи динамического распределения памяти осуществляются функциями `malloc()` и `free()`. Языки Perl и PHP делают это с помощью своих внутренних механизмов. Например, в случае увеличения размера массива или строк динамической длины разработчику не придется заботиться об управлении памятью.
- **Манипулирование строками.** Язык Perl имеет самые развитые возможности в этой области, язык PHP с небольшим отрывом идет вторым в этом списке. Для сравнения: язык C в этом отношении сильно отстает.

Конечно, для упрощения своих задач на языке C можно написать библиотеки, которые бы решали такие задачи, как управление памятью и обработка текста. Но их надо будет отлаживать и позаботиться о том, чтобы они работали эффективно. В то же время аналогичные алгоритмы языков Perl и PHP хорошо отлажены и эффективны, так как они были опробованы на практике множество раз. Таким образом, разработчик может сэкономить свое время и воспользоваться плодами чужого труда, прошедшими проверку временем. (Если интерпретатор содержит ошибку, это вполне можно пережить и дождаться, когда она будет устранена. При написании программы на C разработчику потребуется более тщательный контроль над поведением программы.)

Рассматриваемые языки программирования различаются уровнем своей безопасности. Интерфейс C API обеспечивает интерфейс с сервером самого низкого уровня и самую слабую защиту. В этом смысле интерфейс C обеспечивает самый низкий уровень защиты. При неверном использовании функций API в лучшем случае можно получить ошибку "out-of-sync" ("рассогласование"), в худшем случае произойдет крах программы. Механизмы языков Perl и PHP имеют более высокий уровень защиты. Самое страшное, что может произойти при ошибочных действиях, — это сбой сценария, но краха интерпретатора не произойдет. Еще одним источником ошибок, чреватых крахом, в языке C можно назвать использование динамического распределения памяти и указателей, связанных с ними. Языки Perl и PHP сами осуществляют управление памятью, и вероятность их краха в результате ошибок, вызванных управлением памятью, существенно ниже.

Время разработки программы на каком-либо языке зависит от объема внешней поддержки, доступной для этого языка. Внешняя поддержка языка C доступна в форме библиотек, содержащих функции интерфейса C API СУБД MySQL в виде подпрограмм, которые проще использовать. Такие библиотеки имеются как для C, так и для C++. Без сомнения, Perl имеет большее число дополнений в виде Perl-модулей (аналогично модульной концепции Web-сервера Apache). Существует специальная инфраструктура, предназначенная для упрощения поиска этих модулей (CPAN или Comprehensive Perl Archive Network — полная сеть архивов языка Perl). С помощью модулей Perl можно получить доступ к любому типу функций, не написав при этом ни одной строчки кода. Хотите написать сценарий, который генерирует отчет по данным из базы данных, а потом отослать по почте как вложение? Достаточно посетить Web-узел `cpan.perl.org`, где можно найти один из модулей MIME, и в вашем в распоряжении появится возможность быстро генерировать вложения в электронную почту. Язык PHP не имеет аналогичного уровня внешней поддержки, хотя для PHP 4 эта ситуация стала меняться к лучшему с появлением технологии PEAR.

Переносимость

Проблема переносимости заключается в том, насколько просто программа, написанная для работы с СУБД MySQL, может быть изменена для использования с другой СУБД. Это может и не интересовать разработчика приложения, но мнение, что эта программа никогда не будет использоваться ни с какой другой базой данных, кроме СУБД MySQL, может не отражать объективной реальности. Предположим, что вы поменяли работу и хотите использовать свои старые наработки, но новый начальник использует другую СУБД.

Если одним из главных приоритетов является переносимость, то необходимо четко понимать различия между API.

- Интерфейс DBI обеспечивает максимальную переносимость потому, что при его разработке была четко поставлена именно эта задача.
- Интерфейс языка PHP менее переносим потому, что он не обеспечивает такого же унифицированного интерфейса для различных механизмов баз данных, как это делает интерфейс DBI. Вызов функции языка PHP для каждой поддерживаемой базы данных имеет тенденцию повторять аналогичную в интерфейсе нижнего уровня C API. Имеет место некоторое сглаживание различий, но, как минимум, вам потребуется изменить названия функций, осуществляющих связь с базами данных. Возможно, понадобится немного изменить логику вашего приложения, так как не всегда интерфейсы различных баз данных работают аналогичным образом.
- Интерфейс C API обладает самой низкой переносимостью между базами данных. Изначально он был разработан непосредственно для работы с СУБД MySQL.

Переносимость в форме независимости от типа баз данных имеет особенно большое значение для получения доступа к нескольким базам данных из одного приложения. Это могут быть простые задачи переноса данных из одной СУБД в другую или более сложные проекты, рассчитанные на создание отчета на основании данных, хранящихся в различных базах данных.

Одновременное использование интерфейсов DBI и PHP позволяет осуществлять доступ к различным механизмам баз данных так, что пользователь может получить одновременный доступ сразу к серверам нескольких баз данных или к нескольким узлам. Однако эти интерфейсы не все одинаково подходят для решения задач выборки и обработки данных из распределенных СУБД. В этом случае интерфейс DBI предпочтительнее, потому что с ним используется один набор вызовов, независимо от того, какая база данных при этом используется. Предположим, что стоит задача передачи данных между СУБД MySQL, СУБД mSQL и СУБД Postgres. В интерфейсе DBI все различие заключается в использовании вызова `DBI->connect()` для подключения к любому серверу. Интерфейс PHP требует написания довольно сложного сценария, включающего по три запроса на чтение и запись. В этой ситуации с большой долей вероятности можно сказать, что вы воспользуетесь модулем PEAR, чтобы сгладить различия между механизмами баз данных.

6

Программный интерфейс MySQL C API

В этой главе...

Общая процедура создания клиентских программ
Client1 — подключение к серверу
Client2 — добавление диагностики ошибок
Client3 — передача параметров соединения во время выполнения
Обработка запросов
Client4 — программа интерактивного ввода запросов
Создание клиентов с поддержкой протокола SSL
Библиотека встроенного сервера
Разное

СУБД MySQL имеет клиентскую библиотеку, созданную на языке программирования C. Ее можно использовать для создания клиентских программ, работающих с базами данных СУБД MySQL. Библиотека обеспечивает интерфейс, обладающий следующими возможностями:

- функции управления подключением для начала и завершения сеанса связи с сервером;
- функции конструирования запросов, отсылки их на сервер и обработки полученных результатов;
- функции диагностики состояния и ошибок для определения точной причины возникновения ошибки.

В этой главе продемонстрировано использование клиентской библиотеки в процессе написания программ. При этом будет преследоваться цель достижения полного соответствия с уже существующими программами из дистрибутива СУБД MySQL, модульности и возможности повторного использования создаваемого кода. Предполагается, что читатель уже имеет общее представление о программировании на языке C, не являясь при этом экспертом.

В главе рассматривается процесс разработки нескольких клиентских программ. Сначала разрабатывается скелет программы, который не делает ничего, кроме подключения к серверу и отключения от него. (Причина этого заключается в том, что несмотря на то, что клиентские программы написаны для выполнения разных задач, они имеют одну общую функцию: функцию подключения к серверу.) Разобьем процесс создания программы на несколько этапов:

- создание программного кода, обеспечивающего подключение к базе данных и отключение от нее (`client1`);
- добавление проверки на ошибки (`client2`);
- добавление возможности получения параметров подключения (узел, пользователь, пароль) во время работы (`client3`).

Полученная программа `client3` является достаточно общей, и ею можно воспользоваться для создания клиентской программы любого назначения. После этого поразмыслим над тем, каким образом можно обрабатывать различные типы запросов. Сперва научимся обрабатывать жестко заданные операторы SQL, а потом напишем программу, которая позволит обрабатывать произвольные запросы. После этого вставим программный код, обрабатывающий запросы, в программу `client3` для программы (`client4`), позволяющей подобно клиенту `mysql` выполнять интерактивные запросы.

Затем вы узнаете, какую пользу можно извлечь из двух новых возможностей СУБД MySQL 4:

- написание программ, которые могут обмениваться данными с сервером через защищенные соединения с использованием протокола SSL (Secure Sockets Layer);
- написание приложений, использующих библиотеку `libmysqld`, предназначенную для программирования встроенного сервера.

Кроме того, рассмотрим (и решим) наиболее общие проблемы, например, каким образом можно получить информацию о структуре таблиц и как можно хранить в базе данных изображения.

В этой главе информация о функциях и типах данных клиентской библиотеки дана только по мере необходимости. Полный перечень функций и типов данных можно найти в приложении Д, "Программы MySQL". В дальнейшем это приложение можно использовать в качестве справочного материала по клиентской библиотеке.

Все приведенные здесь примеры можно найти на Web-узле. Для этого обратитесь к приложению А, "Получение и инсталляция программного обеспечения".

Где можно найти примеры

Самый распространенный вопрос: где можно найти примеры клиентских программ, написанных на языке С? Конечно же, в этой книге. Но иногда люди забывают, что сам дистрибутив СУБД MySQL содержит несколько клиентских программ (например, `mysql`, `mysqladmin` и `mysqldump`), большинство из которых написано на языке программирования С. Исходные тексты дистрибутива доступны, поэтому СУБД MySQL уже сама по себе содержит примеры кода клиентских программ. Поэтому время от времени просматривайте дистрибутивы.

Общая процедура создания клиентских программ

В данном разделе описаны этапы компиляции и связывания программ, использующих клиентскую библиотеку СУБД MySQL. Набор команд, необходимых для написания клиентских программ в различных системах, несколько варьируется. Поэтому, возможно, вам понадобится немного видоизменить команды, предложенные здесь вашему вниманию. Однако это описание имеет общий характер, и его можно применить в процессе создания любой клиентской программы.

Основные требования к системе

Очевидно, что при программировании клиентских программ на языке С потребуется компилятор С. В примерах, приведенных здесь, используется компилятор `gcc`. Кроме того, потребуются такие дополнения к исходным файлам:

- файлы заголовков MySQL;
- клиентская библиотека MySQL.

Эти два компонента собственно и составляют поддержку программирования клиентов СУБД MySQL. Вероятно, они уже установлены в вашей операционной системе. В противном случае их надо получить. Здесь есть два варианта: если СУБД MySQL устанавливалась с исходного или двоичного дистрибутива, поддержка программирования клиентов должна быть установлена в процессе этой инсталляции. Если СУБД MySQL устанавливалась из RPM-файлов, эта поддержка не требуется, так как в таком случае установка производится с помощью соответствующего RPM-инструментария. Получить справку о процессе установки файлов заголовков и библиотеки можно в приложении А, "Получение и инсталляция программного обеспечения".

Компиляция и редактирование связей клиентской программы

Для успешной компиляции и редактирования связей клиентской программы необходимо задать путь к файлам заголовков СУБД MySQL и клиентской библиотеке. Обычно они находятся не там, где хранятся такие файлы по умолчанию. Предположим, что файл заголовков и клиентскую библиотеку можно найти, используя путь `/usr/local/include/mysql` и `/usr/local/lib/mysql`.

Для того чтобы указать компилятору, как можно найти файл заголовка, ему нужно указать аргумент `-I/usr/local/include/mysql` во время компилирова-

ния исходного файла в объектный файл. Например, для того чтобы из исходного файла myclient.c получить объектный файл myclient.o, необходимо воспользоваться следующей командной строкой:

```
% gcc -c -I/usr/local/include/mysql myclient.c
```

Для того чтобы указать редактору связей, как можно найти клиентскую библиотеку, ему нужно указать аргументы -L/usr/local/lib/mysql и -mysqlclient во время редактирования связей объектного файла при создании исполняемого двоичного кода следующей командной строкой:

```
% gcc -o myclient myclient.o -L/usr/local/lib/mysql -lmysqlclient
```

Если клиентская программа состоит из нескольких объектных файлов, их все необходимо задать в командной строке.

Шаг редактирования может завершиться сообщением об ошибке, в котором говорится, что надо найти где-то недостающую функцию. В таких случаях в командную строку следует добавить дополнительный аргумент -l, в котором будет указана недостающая функция. Если это сообщение о недостающей функции compress() или uncompress(), попробуйте добавить -lz или -lgz, чтобы задать редактору связей поиск библиотеки с функциями сжатия zlib:

```
% gcc -o myclient myclient.o -L/usr/local/lib/mysql -lmysqlclient -lz
```

Если редактирование связей завершилось ошибкой из-за того, что редактор связей не смог найти функцию floor(), подключите математическую библиотеку, добавив в конец строки -lm. Может потребоваться подключить таким же образом и другие библиотеки. Например, в ОС Solaris может понадобиться подключение библиотек -lsocket и -lnsl.

В версии 3.23.21 появилась возможность использовать утилиту mysql_config, предназначенную для определения соответствующих флагов при компиляции и редактировании связей программ MySQL. Например, утилита может отображать необходимые аргументы:

```
% mysql_config --cflags
-I'/usr/local/mysql/include/mysql'
% mysql_config --libs
-L'/usr/local/mysql/lib/mysql' -lmysqlclient -lz -lcrypt -lnsl -lm
```

Для использования утилиты mysql_config непосредственно в командах компиляции или редактирования связей ее можно активизировать, предварительно заключив в обратные одинарные кавычки:

```
% gcc -c `mysql_config --cflags` myclient.c
% gcc -o myclient myclient.o `mysql_config --libs`
```

Оболочка выполнит mysql_config и подставит ее вывод в команду, что позволит автоматически задавать соответствующие флаги для компилятора gcc.

Можно предположить, что уровень позволяет вам работать, избегая ввода командных строк с клавиатуры, пользуясь командой make. Предположим, что ваш клиент myclient состоит из двух исходных файлов main.c и aux.c и файла заголовка myclient.h. Простейший файл Makefile может выглядеть следующим образом:

```
CC = gcc
INCLUDES = -I/usr/local/include/mysql
LIBS = -L/usr/local/lib/mysql -lmysqlclient
```

```
all: myclient
main.o: main.c myclient.h
   $(CC) -c $(INCLUDES) main.c aux.o: aux.c myclient.h
   $(CC) -c $(INCLUDES) aux.c
myclient: main.o aux.o
   $(CC) -o myclient main.o aux.o $(LIBS)
clean:
   rm -f myclient main.o aux.o
```

С помощью `makefile` можно перестраивать программу после внесения изменений в какой-либо из исходных файлов, вводя `make`:

```
% make
gcc -c -I/usr/local/mysql/include/mysql myclient.c
gcc -o myclient myclient.o -L/usr/local/mysql/lib/mysql -lmysqlclient
```

Это проще и влечет за собой меньше ошибок, чем при вводе длинной командной строки `gcc`. А `Makefile` также упрощает процедуру изменения процесса построения. Например, если система требует подключения дополнительных библиотек, наподобие математической или сжатия, просто отредактируйте строку `LIBS` в файле `Makefile`, добавив в нее `-lm` и `-lz`:

```
LIBS = -L/usr/local/lib/mysql -lmysqlclient -lm -lz
```

Если понадобится подключить другие библиотеки, добавьте их в строку `LIBS`. При последующем запуске утилита `make` автоматически использует модифицированное значение `LIBS`.

Другим способом изменения переменных утилиты `make` является непосредственное задание их в командной строке. Например, если ваш компилятор называется `cc`, а не `gcc` (как это будет в случае с Mac OS X), необходимо поступить следующим образом:

```
% make CC=cc
```

При наличии утилиты `mysql_config` можно воспользоваться ее возможностями. Это позволит избежать литерального указания путей к файлу заголовков и библиотечному каталогу в файле `Makefile`. Вместо этого достаточно указать строки `INCLUDES` и `LIBS`:

```
INCLUDES = ${shell mysql_config --cflags}
LIBS = ${shell mysql_config --libs}
```

Утилита `make` выполнит все команды `mysql_config` и использует их вывод для установки значений всех соответствующих переменных. Показанная здесь конструкция `${shell}` поддерживается утилитой `make` в соответствии со стандартом GNU; но вам может потребоваться другой синтаксис, принятый той версией утилиты `make`, которая не соответствует стандарту GNU.

При использовании интегрированной среды разработки (integrated development environment – IDE) файл `Makefile` может не потребоваться вообще. Но это зависит от конкретной среды разработки.

Client1 — подключение к серверу

Наша первая клиентская программа MySQL – наиболее простая. Она подключается к серверу, отключается от него, после чего завершает работу. Само по себе

это не имеет смысла, но чтобы начать работу с базой данных, надо к ней подключиться. Эта операция будет использоваться чаще всего во всех создаваемых вами клиентских программах. Кроме того, эта задача позволяет нам начать знакомство с процессом программирования клиентов на примере самой простой задачи. В дальнейшем мы будем наращивать функциональность программ по методу от простого к сложному.

Исходный текст нашей первой клиентской программы содержится в одном-единственном файле `client1.c`:

```
/* client1.c - подключиться/отключиться от MySQL-сервера */
#include <my_global.h>
#include <mysql.h>
static char *opt_host_name = NULL; /* узел сервера (умолчание = localhost)
*/
static char *opt_user_name = NULL; /* имя пользователя (умолчание = ваше
имя при регистрации в системе)*/
static char *opt_password = NULL; /* пароль (умолчание = отсутствует) */
static unsigned int opt_port_num = 0; /* номер порта (используйте встроен-
ное значение) */
static char *opt_socket_name = NULL; /* имя разъема (используйте встроен-
ное значение) */
static char *opt_db_name = NULL; /* база данных (умолчание = отсутствует)
*/
static unsigned int opt_flags = 0; /* флаги соединения (нет) */
static MYSQL *conn; /* указатель на дескриптор соединения */
int
main (int argc, char *argv[])
{
   /* инициализация дескриптора соединения */
   conn = mysql_init (NULL);
   /* подключение к серверу */
   mysql_real_connect (conn, opt_host_name, opt_user_name, opt_password,
opt_db_name, opt_port_num, opt_socket_name, opt_flags);
   mysql_close (conn);
   /* отключение от сервера */
   exit(0);
}
```

Исходный файл начинается со строк, подключающих заголовки `my_global.h` и `mysql.h`. Клиент MySQL может включать и больше файлов заголовков, но это является допустимым минимумом.

- Файл `my_global.h` берет на себя заботу о подключении нескольких других файлов общепринятых заголовков, таких как `stdio.h`. В этот перечень также попадает файл `windows.h`, предназначенный для обеспечения совместимости с ОС Windows, при компиляции под управлением ОС Windows. (Вероятно, вы и не планируете писать программу, работающую под управлением ОС Windows, сами, но если вы собираетесь распространять свою программу, наличие ссылки на этот файл поможет тем, кто будет компилировать ее потом в ОС Windows.)
- Файл `mysql.h` определяет первостепенные константы и структуры данных, связанные с СУБД MySQL.

Большое значение имеет и порядок следования: файл `my_global.h` должен указываться перед любыми другими файлами заголовков.

Затем в программе объявляется набор переменных, соответствующих параметрам, которые будут задаваться при подключении к серверу. Для этого клиента этим параметрам задаются значения по умолчанию. Позже мы разработаем более гибкий механизм, позволяющий задавать значения, отменяющие умолчание, с помощью конфигурационного файла или в командной строке. (Вот почему все имена начинаются с префикса `opt_`; что подчеркивает то, что в скором будущем эти переменные станут настраиваемыми с помощью командной строки.) В этой программе также объявляется указатель на структуру `MYSQL`, которая будет служить дескриптором соединения.

Функция `main()` устанавливает и разрывает соединение с сервером. Установка соединения с сервером является процессом, который можно разделить на два этапа.

1. Вызов функции `mysql_init()` для получения дескриптора соединения. При передаче значения "`NULL`" функции `mysql_init()` последняя размещает переменную, инициализирует ее и возвращает указатель на нее. Это структура, содержащая информацию о соединении. Переменные данного типа называются дескрипторами соединения.
2. Вызов функции `mysql_real_connect()` для установки соединения с сервером. Функция `mysql_real_connect()` принимает очень много параметров.
   - *Указатель на дескриптор соединения.* Он не может быть равен "`NULL`", а должен представлять собой значение, возвращенное функцией `mysql_init()`.
   - *Имя узла сервера.* Если в качестве имени узла указано имя компьютера или IP-адрес узла, клиент подключится к указанному узлу, используя подключение TCP/IP. Если в качестве имени узла указано "`NULL`" или узел "`localhost`", клиент подключится к серверу, запущенному на локальном узле, через стандартный разъем ОС UNIX.

     Это справедливо и для операционной системы Windows с одним исключением, что вместо разъема ОС UNIX используется подключение по протоколу TCP/IP. (В ОС Windows NT и производных от нее при указании в качестве имени узла значения "`NULL`" или "`.`" до использования протокола TCP/IP делается попытка использовать поименованный канал.)
   - *Имя пользователя и пароль для использования учетной записи СУБД MySQL.* Если имя имеет значение "`NULL`", клиентская библиотека посылает на сервер имя, с которым вы зарегистрировались. Если пароль имеет значение "`NULL`", он не посылается.
   - *Имя базы данных, которая выбирается по умолчанию после установки соединения.* Если это значение равно "`NULL`", база данных не выбирается.
   - *Номер порта и файл разъема.* Номер порта используется для TCP/IP-соединений. Имя разъема используется для соединений с помощью UNIX-разъема в ОС UNIX или соединений с помощью именованных каналов (в Windows). Они равны 0 и `NULL`, чтобы указать клиентской библиотеке воспользоваться значениями по умолчанию.
   - *Значения флагов.* Это значение равно 0. Это значит, что не будут использоваться никакие специальные параметры соединения.

Детальнее обо всем этом можно узнать в описании функции `mysql_real_connect()` в приложении Е, "Программный интерфейс приложений С". Например, там описано, как параметр `hostname` взаимосвязан с параметрами `port number` и `socket name`, и приведены возможные значения параметра `flags`. В приложении также описана функция `mysql_options()`, которая может использоваться для указания параметров соединения до вызова функции `mysql_real_connect()`.

Для завершения соединения передайте указатель на дескриптор соединения функции `mysql_close()`. Дескриптор соединения, автоматически размещенный функцией `mysql_init()`, теперь автоматически отменяется функцией `mysql_close()`. Это приводит к завершению соединения.

Для того чтобы работать с программой `client1`, откомпилируйте и отредактируйте связи, а затем запустите ее, просто выполнив команду:

```
% ./client1
```

Ведущая последовательность символов "`./`" может потребоваться в ОС UNIX, если в пути поиска не задан текущий каталог ("`.`"). Если каталог есть в пути поиска или вы используете ОС Windows, "`./`" в команде можно не указывать:

```
% client1
```

Эта программа подключается к серверу, разрывает соединение и завершает работу. Не слишком много, но это только начало. И тут очевидны два явных недостатка.

- Клиентская программа не диагностирует ошибки. Поэтому невозможно сказать, работает ли она вообще.
- Параметры соединения (имя узла, пользователя и т.д.) прописаны непосредственно в коде. Конечно, лучше предоставить пользователю возможность задавать эти параметры в конфигурационном файле или в командной строке.

Это не самые сложные задачи, и мы легко решим их на следующих этапах.

Client2 — добавление диагностики ошибок

Второй клиент аналогичен первому, но он уже сможет обрабатывать ошибочные ситуации, возникающие при установлении соединения. Достаточно часто можно прочитать в книгах по программированию о том, что диагностика ошибок "отдается на откуп" читателю для упражнений в программировании. Давайте посмотрим правде в глаза: программирование диагностики ошибок — достаточно нудное дело.

Но автор придерживается другого мнения. Клиентские программы должны адекватно реагировать на возникновение ошибок. Все вызываемые функции возвращают значения, свидетельствующие о том или ином исходе операции.

Рассмотрим с этой точки зрения нашу программу `client1`. Как можно узнать, было ли подключение к серверу? Журнал регистрации подключений позволяет судить об этом по событиям "подключение" и "выход", произошедшим примерно в то время, когда была запущена программа:

```
020816 21:52:14       20 Connect   sampadm@localhost on
                         20 Quit
```

В противном случае можно увидеть сообщение об отказе соединения:

```
020816 22:01:47      21 Connect   Access denied for user:
'sampadm@localhost'
                              (Using password: NO)
```

Такое сообщение свидетельствует о том, что соединение не состоялось. К сожалению, сама программа `client1` не информирует, какой из этих двух исходов операции имел место. В любом случае сложилась ненормальная ситуация: чтобы узнать о том, как отработала программа, приходится обращаться к системному журналу! Давайте исправим эту ситуацию.

Функции клиентской библиотеки MySQL, возвращающие значения, обычно отображают успешное или неуспешное завершение операции одним из двух способов.

- Функции, получающие в качестве параметра указатель на значение, возвращают ненулевое значение в случае успешного завершения операции и "NULL" в случае ошибки. (В этом контексте "NULL" означает "пустой указатель", а не пустое значение столбца СУБД MySQL.)

  Все клиентские функции, которыми мы пользовались до сих пор, будь то `mysql_init()` или `mysql_real_connect()`, в случае успешного завершения операции возвращали указатель на дескриптор соединения и значение "NULL" в случае неудачного завершения операции.

- Функции, получающие в качестве параметра целые числа, возвращают значение 0 в случае успешного завершения операции и ненулевое значение в случае ошибки. При этом необходимо четко понимать, что в случае ошибки не следует делать проверку на конкретные ненулевые значения, например -1. Такая функция может не возвратить никакого значения в случае неуспешного завершения операции. Ниже приведен код, который неправильно обрабатывает возвращаемое значение.

  ```
  if (mysql_XXX() == -1)/* это пример неправильной проверки */
  fprintf (stderr, "случилось что-то страшное \n");
  ```

  Такая проверка может сработать, а может и не сработать. Интерфейс MySQL API не определяет никакого конкретного ненулевого значения, возвращаемого в качестве ошибки, за исключением того, что оно должно быть ненулевым. Проверку на ошибочное завершение операции следует переписать следующим образом:

  ```
  if (mysql_XXX() != 0)      /* это правильная проверка */
  fprintf (stderr, "случилось что-то страшное \n");
  ```

  или вот так:

  ```
  if (mysql_XXX())          /* это правильная проверка */
  fprintf (stderr, "случилось что-то страшное \n");
  ```

Это две равноценные проверки. И если просмотреть исходные тексты самой MySQL, можно увидеть, что чаще всего используется проверка второго типа, так как она компактнее.

Но далеко не каждый интерфейс возвращает значение. К числу таких можно причислить другую функцию, которая была использована при написании клиентской программы, — `mysql_close()`. (Как она может не сработать? И если это произошло, что из того? В любом случае соединение обрывается.)

Для выяснения причин сбоя можно воспользоваться вызовом двух функций. Это функция `mysql_error()`, возвращающая строку, содержащую сообщение об ошибке, и функция `mysql_errno()`, возвращающая цифровой код ошибки. Их надо вызывать сразу же после обнаружения ошибки, поскольку, если это сделать после отработки какой-либо другой функции, она вернет уже какой-то свой статус, и вся информация, полученная функцией `mysql_error()` или `mysql_errno()`, будет относиться к функции, вызванной позже.

В общем случае пользователю программ значительно удобнее получать строку, содержащую информацию об ошибке, чем числовой код ошибки. Для целостности картины воспользуемся обеими функциями.

Принимая во внимание все вышесказанное, напишем вторую клиентскую программу `client2`. Эта программа аналогична `client1`, но добавлена соответствующая обработка ошибок. Исходный текст `client2.c` имеет следующий вид:

```
/*
* client1.c - подключиться/отключиться от MySQL-сервера * с диагностированием ошибок
*/
#include <my_global.h>
#include <mysql.h>
static char *opt_host_name = NULL; /* узел сервера (умолчание = localhost)
*/
static char *opt_user_name = NULL; /* имя пользователя (умолчание = ваше имя при регистрации в системе)*/
static char *opt_password = NULL; /* пароль (умолчание = отсутствует) */
static unsigned int opt_port_num = 0; /* номер порта (используйте встроенное значение) */
static char *opt_socket_name = NULL; /* имя разъема (используйте встроенное значение) */
static char *opt_db_name = NULL; /* база данных (умолчание = отсутствует)
*/
static unsigned int opt_flags = 0; /* флаги соединения (нет) */
static MYSQL *conn; /* указатель на дескриптор соединения */
int
main (int argc, char *argv[])
{
   /* инициализация дескриптора соединения */
   if (conn = mysql_init (NULL);
   if (conn == NULL)
   {
     fprintf (stderr, "mysql_init() ошибка (вероятно, не хватает памяти)\n");
     exit (1);
   }
   /* подключение к серверу */
   if (mysql_real_connect (conn, opt_host_name, opt_user_name,
opt_password, opt_db_name, opt_port_num, opt_socket_name, opt_flags) ==
NULL)
   {
     fprintf (stderr, "ошибка mysql_real_connect():\nError %u (%s)\n",
     mysql_errno (conn), mysql_error (conn));
     mysql_close (conn);
     exit (1);
   }
   /* отключение от сервера */
```

```
    mysql_close (conn);
    exit(0);
}
```

Логика проверки на ошибки заключается в том, что в случае ошибочного завершения функции `mysql_init()` и `mysql_real_connect()` возвращают значение "NULL". Обратите внимание, что несмотря на то, что программа анализирует значение, возвращаемое функцией `mysql_init()`, в случае его сбоя функция диагностики ошибок не вызывается. Это происходит потому, что в случае сбоя функции `mysql_init()` дескриптор соединения не может содержать информации, достойной внимания. Однако при сбое функции `mysql_real_connect()` дескриптор соединения не отражает реального соединения, а только содержит информацию об ошибке, которую можно передать функции диагностики ошибок. Дескриптор может быть также передан функции `mysql_close()` для того, чтобы освободить память, которая была автоматически выделена для него функцией `mysql_init()`. (Не передавайте дескриптор другим клиентским программам! Это недопустимо потому, что они предполагают успешное соединение, что, в свою очередь, может стать причиной сбоя программы.)

Откомпилируйте и отредактируйте связи `client2`, а затем запустите программу:

```
% ./client2
```

Если `client2` не выводит никаких сообщений (как показано выше), можно считать, что соединение было успешным. При возникновении сбоя мы увидим следующее:

```
% ./client2
mysql_real_connect() ошибка (вероятно, не хватает памяти):
Error 1045 (Access denied for user:
            'paul@localhost'    (Using password: NO))
```

Это свидетельствует о том, что соединение не было установлено, и указывает причину, почему так случилось. Это также означает, что наша первая программа, `client1`, никогда вообще успешно не подключалась к серверу. (Ведь `client1` использует те же параметры подключения.) Об этом ничего не было известно, так как программа `client1` "не заботилась" о проверке ошибок. Программа `client2` уже делает такую поверку, поэтому она может диагностировать ошибки. Вот почему всегда надо проверять значения, возвращаемые функциями программного интерфейса.

Информированность о возникающих проблемах значительно лучше неизвестности, и в этом кроется причина того, зачем требуется проверка возвращаемых значений функций API. Если этого не делать, могут возникнуть неnужные затруднения при программировании. Эта тема часто является предметом обсуждения в списках рассылки, посвященных данной проблематике. Ответы на многие вопросы можно найти при проверке ошибок. Вот типичные вопросы: "Почему моя программа всегда сбоит при выдаче SQL-запроса?" "Почему мои запросы ничего не возвращают?" Во многих случаях программа, о которой идет речь, не проверяет, было ли до генерирования запроса вообще установлено соединение и был ли запрос завершен успешно. Не допускайте такой ошибочной практики, когда априори считается, что каждое обращение к клиентской библиотеке происходит успешно.

Все остальные примеры из этой главы обязательно выполняют диагностирование ошибок, и вы как разработчик аналогичных программ тоже обязаны делать это. Это может показаться сложным, но в конечном счете это сокращает время разработки программного обеспечения. Аналогичный подход приводится в главе 7, "Программный интерфейс Perl API", и в главе 8, "Интерфейс API для языка написания сценариев PHP".

Теперь предположим, что отработка программы `client2` завершилась сообщением "`Access denied`" ("`Отказ в доступе`"). Каким образом решить эту проблему? Можно, например, внести изменения в строки `#define`, указав там в качестве имен сервера, пользователя и пароля такие, которые позволят получить доступ к серверу. Это поможет хотя бы тем, что появится возможность подключения к серверу. Но в этом случае эти значения жестко прописаны в программном коде. Я не советую применять такой подход, особенно при задании пароля. Ведь может показаться, что в этом случае пароль надежно спрятан потому, что он скомпилирован в двоичном виде, но это совсем не так. (Не говоря уже о том, что любой, кто имеет право чтения ваших исходных файлов, без особого труда сможет узнать ваш пароль.)

В следующем разделе будут разработаны более гибкие методы, демонстрирующие возможности подключения к серверу. Но сначала я хочу разработать упрощенный метод программирования сообщений об ошибках, так как это придется делать достаточно часто. Мы будем продолжать использовать в сообщениях об ошибках как цифровой код СУБД MySQL, так и описательную строку, но я предпочитаю не вызывать постоянно функции сообщения об ошибках `mysql_errno()` и `mysql_error()`:

```
if (...функция MySQL сбоит...)
{
  fprintf (stderr, "...некое сообщение об ошибке...:\nError %u
(%s)\n",
  mysql_errno (conn), mysql_error (conn));
}
```

Вместо этого значительно проще сообщать об ошибках с помощью функции:

```
if (...одна функция MySQL сбоит...)
{
  print_error (conn, "...некое сообщение об ошибке...");
}
```

Так что соберем диагностику ошибок в специальную процедуру `print_error()`. Функция `print_error()` распечатывает сообщение об ошибке и автоматически вызывает сообщение об ошибке СУБД MySQL. Функцией `print_error()` значительно удобнее пользоваться, чем длинной функцией `fprintf()`. Кроме того, это сделает программу более читабельной. Ею можно воспользоваться для того, чтобы выполнить какие-то осмысленные действия, даже если переменная `conn` будет равняться "`NULL`". Точно так же можно воспользоваться процедурой `print_error()`, если функция `mysql_init()` возвращает ошибку. Таким образом, мы избежим путаницы вызовов (в одном случае – функции `fprintf()`, а в другом – функции `print_error()`). Ниже приведен вариант функции `print_error()`, удовлетворяющий этому описанию.

```
void
print_error (MYSQL *conn, char *message)
{
  fprintf (stderr, "%s\n", message);
  if (conn != NULL)
  {
    fprintf (stderr, "Ошибка %u (%s)\n",
    mysql_errno (conn), mysql_error (conn));
  }
}
```

Кто-то может возразить, что при этом программный код становится нечитаемым только потому, что надо обеспечить более полную функциональность программы. Ведь совсем не обязательно писать весь программный код обработки ошибочных ситуаций каждый раз, когда это необходимо. Достаточно просто скопировать его. Все это правильно, но у меня есть ряд возражений.

- Даже если использовать копирование, значительно проще копировать короткие блоки исходного текста.
- Независимо от того, предпочитаете вы или нет вызывать обе функции каждый раз, когда необходимо диагностировать ошибки, постоянная запись диагностирующего кода приводит к внесению изменений и порождает неуверенность при диагностировании ошибок. Заключение диагностирующего кода в отдельную функцию уменьшает этот фактор и повышает однозначность программного кода.
- Если необходимо изменить формат сообщений об ошибках, проще это сделать в одном месте. Или если принято решение выводить сообщение об ошибках в регистрационный журнал, а не на устройство `stderr` (или одновременно и в журнал, и на устройство), это проще сделать только в процедуре `print_error()`. Этот подход позволяет уменьшить количество ошибок и увеличивает уверенность в положительных результатах работы.
- Если при тестировании программы используется отладчик, то можно порекомендовать один очень удобный прием: поместить точку останова внутри функции диагностики ошибок.

По этим причинам до конца этой главы для диагностирования проблем, возникающих при работе с СУБД MySQL, будет использоваться функция `print_error()`.

Client3 — передача параметров соединения во время выполнения

Теперь мы готовы создать нечто "поумнее", чем простое использование жестко прописанных параметров соединения, например предоставить возможность пользователю задавать эти значения при запуске программы. Предыдущие клиентские программы имели один существенный недостаток — параметры соединения были прописаны непосредственно в его исходном тексте. Для того чтобы изменить любое из этих значений, необходимо было редактировать исходный текст программы, компилировать его и потом ре-

дактировать связи. Это не очень удобно, особенно если вы задумали свою программу как распространяемую. Общий способ задания параметров соединения заключается в задании параметров в командной строке. Например, программы из дистрибутива СУБД MySQL принимают параметры в любой из этих двух форм, как показано в следующей таблице.

Параметр	Длинная форма	Короткая форма
Имя узла	`--host=`*имя_узла*	`-h` *имя_узла*
Имя пользователя	`--user=`*имя_пользователя*	`-u` *имя_пользователя*
Пароль	`--password` *или*	`-p` *или*
	`--password=`*ваш_пароль*	`-p`*ваш_пароль*
Номер порта	`--port=`*номер_порта*	`-P` *номер_порта*
Имя разъема	`--socket=`*имя_разъема*	`-S` *имя_разъема*

Для обеспечения совместимости со стандартами СУБД MySQL клиент, создаваемый нами, будет воспринимать данные в аналогичном формате. Это достигается достаточно просто, так как клиентская библиотека включает функцию, которая осуществляет синтаксический анализ вводимых параметров. Кроме того, наша программа-клиент получит возможность получать информацию из конфигурационных файлов. Это позволяет разместить параметры соединения в файле `~/.my.cnf` (в файле `.my.cnf`, находящемся в вашем корневом каталоге) и освободит от необходимости указывать их в командной строке. Клиентская библиотека позволяет проверять конфигурационные файлы СУБД MySQL и выбирать любые значения из них. Добавив только несколько строк в текст своей программы, можно сделать ее более интеллектуальной, и для этого совсем не надо изобретать велосипед. Синтаксис конфигурационного файла описан в приложении Д, "Программы MySQL".

До написания программы `client3`, напишем несколько программ, которые проиллюстрируют, как обрабатываются параметры СУБД MySQL. Здесь показано, что параметры обрабатываются достаточно просто, не добавляя при этом особых проблем при соединении к серверу СУБД MySQL и обработке запросов.

Доступ к содержимому конфигурационного файла

Для чтения конфигурационного файла можно воспользоваться функцией `load_defaults()`, которая просматривает конфигурационный файл, производит синтаксический анализ его содержимого, обновляя вектор аргументов программы (массив `argv[]`). Таким же образом анализируются параметры, заданные в командной строке. Поэтому при синтаксическом анализе параметров командной строки мы получаем параметры как часть обычного цикла анализа параметров. Параметры добавляются в начало массива `argv[]`. Так что если параметры задаются одновременно и в конфигурационном файле, и в командной строке, они попадают в массив `argv[]` позже, чем любой параметр, который добавляется функцией `load_defaults()`.

Вашему вниманию предоставляется небольшая программа `show_argv`, показывающая работу с функцией `load_defaults()` и иллюстрирующая, к каким изменениям это приводит в векторе аргументов:

```
/* show_argv.c - демонстрация воздействия функции load_defaults() на
вектор аргументов */
#include <my_global.h>
#include <mysql.h>
static const char *client_groups[]= { "client", NULL };
int
main (int argc, char *argv[])
{
int i;
  my_init ();
  printf ("Первоначальный вектор аргументов: \n");
  for (i = 0; i < argc; i++)
    printf ("арг %d: %s\n", i, argv[i]);
    my_init ();
    load_defaults ("my", groups, &argc, &argv);
    printf ("Измененный вектор аргументов:\n");
    for (i = 0; i < argc; i++)
        printf ("арг %d: %s\n", i, argv[i]);
    exit (0);
}
```

Конфигурационный файл содержит следующие данные.

- `client_groups[]` — это символьный массив, определяющий, в каких группах конфигурационных файлов вы заинтересованы. Для клиентских программ как минимум определяется "`client`" (для группы `[client]`), а последний элемент обязательно должен иметь значение "`NULL`".
- Подпрограмма `my_init()` является инициализирующей подпрограммой, выполняющей операции установки, необходимые подпрограмме `load_defaults()`.
- Подпрограмма `load_defaults()` принимает содержимое конфигурационного файла, который принимает четыре аргумента: префикс конфигурационных файлов (всегда должен быть "`my`"), массив с перечнем групп и адреса счетчика аргументов программы и вектора. Не передавайте значения счетчика и вектора непосредственно. Передавайте только их адреса, потому что `load_defaults()` должна менять их значения. Необходимо обратить внимание на следующее: несмотря на то, что `argv` является указателем, передается только `&argv`, или адрес указателя.

Подпрограмма `show_argv` печатает свои аргументы дважды: сначала, когда они задаются в командной строке, а потом после изменения их подпрограммой `load_defaults()`. Для того чтобы посмотреть, как работает подпрограмма `load_defaults()`, убедитесь в том, что в вашем корневом каталоге существует файл `.my.cnf` с определенными установками для группы `[client]`. Предположим, что файл `.my.cnf` имеет следующий вид:

```
[client]
user=sampadm
password=secret
host=some_host
```

В этом случае выполнение подпрограммы `show_argv` даст такой результат:

```
% ./show_argv a b
Первоначальный вектор аргументов
```

```
арг 0: ./show_argv
арг 1: a
арг 2: b
Измененный вектор аргументов:
арг 0: ./show_argv
арг 1: - -user=sampadm
арг 2: - -password=secret
арг 3: - -host=some_host
арг 4: a
арг 5: b
```

Возможно, что в выводе появляются некоторые параметры show_argv, которых нет в командной строке или в конфигурационном файле ~/.my.cnf. Это могут быть параметры, заданные в общесистемном конфигурационном файле. Подпрограмма load_defaults() в действительности сначала ищет файл /etc/my.cnf и my.cnf в каталоге данных СУБД MySQL и только потом читает файл .my.cnf в вашем пользовательском каталоге. (В ОС Windows подпрограмма load_defaults() производит поиск в следующей последовательности: файл my.ini в системном каталоге ОС Windows C:\my.cnf и файл my.cnf в каталоге данных СУБД MySQL.)

Функция load_defaults() считывает файл my.ini из системного каталога Windows C:\my.cnf и файл my.cnf из каталога данных СУБД MySQL. Клиентские программы, работающие с подпрограммой load_defaults(), почти всегда в списке групп имеют группу "client" (таким образом, они всегда получают все общие установки клиентов из конфигурационного файла), но программу обработки конфигурационного файла можно настроить на получение параметров из других групп. Предположим, что программа show_argv должна иметь возможность считывать параметры из двух групп — [client] и [show_argv]. Для этого в исходном тексте show_argv.c найдите строку

```
const char *client_groups[] = { "client", NULL };
```

и видоизмените ее:

```
const char *client_groups[] = { "show_argv", "client", NULL };
```

После этого добавьте группу [show_argv] в свой файл ~/.my.cnf:

```
[client]
user=sampadm
password=secret
host=some_host
[show_argv]
host=other_host
```

После внесения этих изменений повторный вызов show_argv даст несколько другой результат:

```
% ./show_argv a b
Первоначальный вектор аргументов
арг 0: /.show_argv
арг 1: a
арг 2: b
Измененный вектор аргументов:
арг 0: /.show_argv
арг 1: - -user=sampadm
арг 2: - -password=secret
арг 3: - -host=some_host
```

```
apr 4: - -host=other_host
apr 5: a
apr 6: b
```

Порядок следования значений параметров в массиве аргументов определяется порядком их следования в конфигурационном файле, а не порядком следования групп в массиве `client_groups[]`. Это означает, что вслед за группой `[client]` в вашем конфигурационном файле вы, вероятно, хотите определить группы, которые характерны для данной программы. Таким образом, если определить параметры в двух группах, преимуществом будут пользоваться параметры, присущие данной программе. Это можно видеть по предыдущему примеру. Параметр узла был задан как в группе `[client]`, так и в группе `[show_argv]`, но поскольку группа `[show_argv]` идет в конфигурационном файле последней, ее установки узла будут представлены в векторе последними. Программа `load_defaults()` не выбирает значения из ваших установок среды.

Для того чтобы воспользоваться переменными установки среды, такими как `MYSQL_TCP_PORT` или `MYSQL_UNIX_PORT`, обратитесь к функции `getenv()`. Такую возможность мы добавлять не будем, но пример проверки значений ряда стандартных переменных, связанных со средой MySQL, привести можно:

```
extern char *getenv();
char *p;
int port_num;
char *socket_name;

if ((p = getenv ("MYSQL_TCP_PORT")) != NULL)
    port_num = atoi (p);
if ((p = getenv ("MYSQL_UNIX_PORT")) != NULL)
    socket_name = p;
```

В стандартных программах MySQL значения стандартных переменных среды имеют более низкий приоритет, чем значения, заданные в конфигурационных файлах или в командной строке. Если есть определенные сомнения в значениях, принимаемых переменными среды, проверьте все установки среды до (а не после) вызова функции `load_defaults()` или обработки параметров командной строки.

Функция `load_defaults()` и безопасность

На многопользовательских системах такие утилиты, как `ps`, могут отображать список аргументов любого произвольного процесса. Поэтому у вас, вероятно, возникнет вопрос, связанный с возможностью просмотра списка аргументов произвольных процессов с помощью команды `ps`, включая и процессы, запущенные другими пользователями. Тут никакой проблемы не возникает, так как `ps` отображает первоначальное содержание `argv[]`. Любой аргумент пароля, созданный функцией `load_defaults()`, содержит ссылку на область, которую он выделил для себя. Эта область не является частью обычного вектора аргументов, поэтому `ps` ее просто "не видит".

С другой стороны, пароль, заданный в командной строке, будет виден с помощью утилиты `ps`. Это является одной из причин, почему следует избегать задания пароля таким образом. Одной из мер предосторожности является удаление пароля из списка аргументов сразу же после запуска программы. О том, как это сделать, рассказывается дальше в этом разделе.

Синтаксический анализ аргументов командной строки

Пользуясь функцией `load_defaults()`, можно загрузить все параметры соединения в вектор аргументов, но его необходимо каким-то образом обработать.

Для этого предназначена другая функция — `handle_options()`. Она встроена в клиентскую библиотеку СУБД MySQL, таким образом, к ней можно обращаться при подключении к этой библиотеке.

Методы обработки параметров, описанные здесь, были впервые анонсированы в СУБД MySQL 4.0.2. До этого для обработки параметров использовалась функция `getopt_long()`. При написании программ, работающих с СУБД MySQL более ранней версии, чем 4.0.2, для справки можно воспользоваться этой главой из первого издания, где описана обработка параметров команд с помощью `getopt_long()`. Главы первого издания можно найти в PDF-формате по адресу `http://www.kitebird.com/mysql-book/`.

Теперь программный, код, опирающийся на функцию `getopt_long()`, заменен интерфейсом, опирающимся на функцию `handle_options()`. Ниже описаны некоторые из усовершенствований, предлагаемые новыми программами обработки параметров.

- **Более точное определение типа и диапазона допустимых значений, принимаемых параметрами.** Например, можно отразить не только то, что параметр принимает целочисленные значения, но и то, что этот параметр должен иметь положительное значение и быть кратным 1024.
- **Интегрирование подсказок, что упрощает их печать по вызову только стандартной библиотечной функции.** Нет необходимости в создании своего собственного кода для создания подсказки.
- **Встроенная поддержка стандартных параметров** `--no-defaults`, `--print-defaults`, `--defaults-file` **и** `--defaults-extra-file`. Эти параметры описаны в разделе "Конфигурационные файлы" приложения Д, "Программы MySQL".
- **Поддержка стандартного набора префиксов параметров наподобие** `--disable-` **и** `--enable-`**, что упрощает применение параметров логического включения/отключения.** Эти возможности не используются в этой главе, но они описаны в приложении Д, "Программы MySQL", в разделе, посвященном обработке параметров.

Примечание

Новые процедуры обработки параметров появились в версии 4.0.2, но лучше всего воспользоваться версией 4.0.5 или более новыми версиями. Некоторые проблемы были обнаружены и устранены в период первоначальной "доводки" этих процедур, которая продолжалась с версии 4.0.2 до 4.0.5.

Использование возможностей обработки параметров СУБД MySQL в этом разделе демонстрируется на примере программы `show_opt`, которая для чтения конфигурационных файлов и установки вектора аргументов вызывает функцию `load_defaults()`, а последующую обработку результатов выполняет с помощью функции `handle_options()`.

Программа `show_otp` позволяет поэкспериментировать с различными параметрами соединения (через конфигурационные файлы или с помощью командной строки) и увидеть в результате, какие значения могут использоваться для подключения к серверу СУБД MySQL. Программа `show_opt` поможет нам по-

нять, что будет происходить с нашей следующей программой, client3, которая будет работать с настоящим кодом, обрабатывающим параметры при реальном подключении к серверу.

Программа show_otp демонстрирует, что происходит на каждом этапе обработки аргументов, выполняя следующие действия.

1. Установка по умолчанию значений имени узла, имени пользователя, пароля и других параметров соединения.
2. Вывод первоначального параметра соединения и значения вектора аргументов.
3. Вызов функции load_defaults() для изменения вектора аргументов в соответствии со значениями, заданными в конфигурационном файле, и распечатки результирующего вектора.
4. Вызов функции getopt_long() для обработки вектора аргументов с последующей распечаткой полученных значений параметров и величин, оставшихся в векторе аргументов.

После этого обсуждается работа show_opt, но сначала необходимо посмотреть на исходный текст самой программы show_opt.c:

```
/*
/* show_opt.c - демонстрация обработки параметров с помощью
load_defaults()
*/
#include <my_global.h>
#include <mysql.h>
#include <my_getopt.h>

static char *opt_host_name = NULL; /* узел сервера (умолчание = localhost)
*/
static char *opt_user_name = NULL; /* имя пользователя (умолчание = ваше
имя при регистрации в системе) */
static char *opt_password = NULL; /* пароль (умолчание = отсутствует)*/
static unsigned int opt_port_num = 0; /* номер порта (используйте встроен-
ное значение) */
static char *opt_socket_name = NULL; /* имя разъема (используйте встроен-
ное значение) */
static const char *client_groups[] = { "client", NULL };
static struct my_option my_opts[] = /* структуры для хранения параметриче-
ской информации */
{
  {"help", '?', "Вывести эту подсказку и выйти",
  NULL, NULL, NULL,
  GET_NO_ARG, NO_ARG, 0, 0, 0, 0, 0, 0},
  {"host", 'h', "Узел, подключения",
  (gptr *) &opt_host_name, NULL, NULL,
  GET_STR_ALLOC, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
  {"password", 'p', "Пароль",
  (gptr *) &opt_password, NULL, NULL,
  GET_STR_ALLOC, OPT_ARG, 0, 0, 0, 0, 0, 0},
  {"port", 'P', "Номер порта",
  (gptr *) &opt_port_num, NULL, NULL,
  GET_UINT, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
  {"socket", 'S', "Путь к разъему",
```

```
   (gptr *) &opt_socket_name, NULL, NULL,
   GET_STR_ALLOC, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
   {"user", 'u', "Имя пользователя",
   (gptr *) &opt_user_name, NULL, NULL,
   GET_STR_ALLOC, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
   { NULL, 0, NULL, NULL, NULL, NULL, GET_NO_ARG,    NO_ARG, 0, 0, 0, 0,
0, 0 }
};
my_bool
get_one_option (int optid, const struct my_option *opt, char *argument)
{
   switch (optid)
   {
    case '?':
    my_print_help (my_opts); /* вывести подсказку */
    exit (0);
   }
   return (0);
}
int
main (int argc, char *argv[])
{
int i;
int opt_err;
   printf ("Первоначальные параметры соединения: \n");
   printf ("имя узла: %s\n", opt_host_name ? opt_host_name : "(null)");
   printf ("имя пользователя: %s\n", opt_user_name ? opt_user_name :
"(null)");
   printf ("пароль: %s\n", opt_password ? opt_password : "(null)");
   printf ("номер порта: %u\n", opt_port_num);
   printf ("имя разъема: %s\n", opt_socket_name ? opt_socket_name :
"(null)");
   printf ("Первоначальный вектор аргументов: \n");
   for (i = 0; i < argc; i++)
    printf ("арг %d: %s\n", i, argv[i]);
   my_init ();
   load_defaults ("my", client_groups, &argc, &argv);
   printf ("Измененный вектор аргументов после load_defaults():\n");
   for (i = 0; i < argc; i++)
    printf ("арг %d: %s\n", i, argv[i]);
    if ((opt_err = handle_options (&argc, &argv, my_opts,
get_one_option)))
    exit (opt_err);
   printf ("Параметры соединения после handle_options():\n");
printf ("имя узла: %s\n", opt_host_name ? opt_host_name : "(null)");
   printf ("имя пользователя: %s\n", opt_user_name ? opt_user_name :
"(null)");
   printf ("пароль: %s\n", opt_password ? opt_password : "(null)");
   printf ("номер порта: %u\n", opt_port_num);
   printf ("имя разъема: %s\n", opt_socket_name ? opt_socket_name :
"(null)");
   printf ("Вектор аргументов после handle_options():\n");
   for (i = 0; i < argc; i++)
    printf ("арг %d: %s\n", i, argv[i]);
   exit (0);
}
```

Метод обработки параметров, продемонстрированный в программе show_opt.c, включает следующие аспекты, являющиеся общими для всех программ, использующих клиентскую библиотеку при обработке параметров, получаемых из командной строки.

1. Кроме файлов заголовков my_global.h и mysql.h, также включен файл my_getopt.h. Файл my_getopt.h задает интерфейс для функций обработки параметров.
2. Определение массива структур my_option. В show_opt.c этот массив называется my_opts. Он имеет по одной структуре на параметр, который может "воспринимать" программа. Каждая структура хранит такую информацию, как короткое и длинное имя параметра, его значение, принимаемое по умолчанию, является ли значение числом или строкой и т.д. Детальное описание структуры my_option приведено ниже.
3. После вызова функции load_defaults() для чтения конфигурационного файла и установки вектора аргументов параметры обрабатываются вызовом функции handle_options(). Первыми двумя аргументами функции handle_options() являются адреса счетчика аргументов и вектора. (Аналогично тому, как это сделано в load_options(), здесь передаются адреса этих переменных, а не их значения.) Третий аргумент указывает на массив структур my_option. Четвертый аргумент является указателем на вспомогательную функцию. Программа handle_options() и структуры my_options разработаны с тем, чтобы основные действия по обработке параметров выполнялись автоматически вашей клиентской библиотекой. Однако, для того, чтобы выполнять специальные действия, которые библиотека не обрабатывает, ваша программа также должна осуществлять вызов вспомогательной функции handle_options(). В модуле show_opt.c эта функция называется get_one_option(). Функционирование вспомогательной функции описано ниже.

Структура my_option определяет информацию для каждого параметра, который "понимает" программа. Она выглядит следующим образом:

```
struct my_option
{
  const char *name; /* длинное имя параметра */
  int id; /* короткое имя или код параметра */
  const char *comment; /* описание параметра для подсказки */
  gptr *value; /* указатель на переменную, в которой будет храниться
значение */
  gptr *u_max_value; /* максимальное значение параметра, определенное
пользователем */
  const char **str_values; /* массив допустимых значений параметра (не
используется) */
  enum get_opt_var_type var_type; /* тип значений параметра */
  enum get_opt_arg_type arg_type; /* требуется ли значение параметра
*/
  longlong def_value; /* значение параметра по умолчанию */
  longlong min_value; /* минимальное допустимое значение параметра */
  longlong max_value; /* максимальное допустимое значение параметра */
  longlong sub_size; /* сдвиг для значения */
```

```
  long block_size; /* множитель значения параметра */
  int app_type; /* зарезервировано для приложения */
};
```

Элементы структуры `my_option` таковы:

- `name`

  Длинное имя параметра. Это форма параметра `--name` без ведущих тире. Например, длинное имя параметра `--user` в структуре `my_option` необходимо задавать как "`user`".

- `id`

  Короткое (однобуквенное) имя параметра или его кодовое обозначение, связанное с параметром, если у него нет однобуквенного обозначения. Например, короткое имя параметра `-u` в структуре `my_option` необходимо задавать как "`u`" для параметров, имеющих только длинные имена и не имеющих соответствующих им коротких имен. Можно задать значения кодов параметров, которые вы будете использовать для коротких внутренних имен. Эти значения должны быть уникальными и отличаться от других односимвольных имен. (Для того чтобы удовлетворить последнее ограничсние, необходимо воспользоваться кодами, превышающими 255, наибольшим возможным кодовым значением для буквы. Пример этого механизма приведен ниже, в разделе "Создание клиентов с поддержкой протокола SSL".)

- `comment`

  Объяснительная строка, в которой описывается назначение параметра. Это текст, который отображается в сообщении подсказки.

- `value`

  Это значение `gptr` (указателя), указывающего на переменные, где будет храниться аргумент параметра. После обработки параметров можно посмотреть, как было установлено значение параметра. Если параметр не имеет аргументов, его значение будет "`NULL`". С другой стороны, тип переменной, на которую указывает указатель, должен совпадать со значением элемента `var_type`.

- `u_max_value`

  Это другое значение `gptr`, используемое только сервером. Для клиентских программ значение `u_max_value` должно быть равно "`NULL`".

- `str_values`

  Этот элемент сейчас не используется. В будущих версиях СУБД MySQL он может использоваться для определения списка допустимых значений, причем любое значение, принимаемое параметром, должно совпадать с одним из этих значений.

- `var_type`

  Этот элемент указывает на то, какой тип значения должен следовать за именем параметра в командной строке, и может принимать одно из следующих значений.

Тип переменной	Описание
GET_NO_ARG	Нет значения
GET_BOOL	Логическое значение
GET_INT	Целое значение
GET_UINT	Беззнаковое целое значение
GET_LONG	Длинное целое значение
GET_ULONG	Беззнаковое длинное целое значение
GET_LL	Длинное-длинное целое значение
GET_ULL	Беззнаковое длинное-длинное целое значение
GET_STR	Строковое значение
GET_STR_ALLOC	Строковое значение

Различие между GET_STR и GET_STR_ALLOC заключается в том, что для GET_STR переменная параметра обеспечивает непосредственную ссылку на значение в векторе аргумента, в то время как для GET_STR_ALLOC создается копия, и переменная параметра дает ссылку на копию.

- arg_type

  Значение arg_type указывает на то, сопровождается ли имя параметра значением, и может принимать одно из следующих значений.

Тип переменной	Описание
NO_ARG	Параметр не принимает значений
OPT_ARG	Параметр может принимать одно из следующих значений
REQUIRED_ARG	Параметру требуется следующий аргумент

Если значение arg_type составляет NO_ARG, то var_type должно быть установлено в GET_NO_ARG.

- def_value

  Для цифровых параметров значение def_value присваивается по умолчанию, если значение параметра не было задано явным образом.

- min_value

  Для цифровых параметров значением min_value определяется минимальное значение параметра. Если задано значение меньше этого значения, оно округляется до значения min_value. Для того чтобы указать, что ограничения по минимуму не существует, задается значение 0.

- max_value

  Для цифровых параметров значением max_value определяется максимальное значение параметра. Если задано значение больше этого значения, оно округляется до значения max_value. Для того чтобы указать, что ограничения по максимуму не существует, задается значение 0.

- sub_size

  Для параметров, имеющих числовые значения, значение sub_size является смещением, которое используется для изменения диапазона, заданного в

векторе аргументов во внутренний диапазон. Например, если значения, заданные в командной строке, лежат в диапазоне от 1 до 256, но в программе требуется использовать внутренний диапазон от 0 до 255, значение sub_size необходимо установить равным 1.

- block_size

  Для параметров, имеющих числовые значения, ненулевое значение block_size означает размер блока. Если понадобится, значения параметров будут округлены до ближайшего меньшего кратного размера. Например, если значения должны быть четными, установите значение block_size равным 2; функция handle_options() округлит все нечетные значения до ближайшего меньшего четного числа.

- app_type

  Этот элемент зарезервирован для использования в приложениях.

Массив my_opts хранит структуры my_option для всех допустимых параметров с завершающей структурой, обозначающей завершение массива:

```
{ NULL, 0, NULL, NULL, NULL, NULL, GET_NO_ARG, NO_ARG, 0, 0, 0, 0, 0, 0 }
```

При обработке вектора аргументов функцией handle_options() первый аргумент (имя программы) пропускается, а затем обрабатываются аргументы параметров, т.е. аргументы, которые начинаются с тире. Это продолжается до тех пор, пока не будет достигнут конец вектора или специальный символ "конец параметров" ('--'). По мере продвижения по вектору аргументов функция handle_options() вызывает функцию подсказки, по одной на параметр, что позволяет производить специальную обработку. Функция handle_options() передает три аргумента функции подсказки: короткое значение параметра, указатель на структуру my_option параметра и указатель на аргумент, который следует за параметром в векторе аргумента (равный NULL при отсутствии значения параметра). Функция handle_options() возвращает счетчик аргумента, а сам вектор будет соответствующим образом переустановлен для отображения списка аргументов, содержащих только непараметрические аргументы.

Ниже приведен пример вызова программы show_opt и полученных результатов (предполагается, что содержимое конфигурационного файла ~/.my.cnf по сравнению с примером show_argv не изменилось).

```
% show_opt -h yet_another_host --user=bill x
Первоначальные параметры соединения:
имя узла: (null)
имя пользователя: (null)
пароль: (null)
номер порта: 0
имя разъема: (null)
Первоначальный вектор аргументов:
арг 0: show_opt
арг 1: -h
арг 2: yet_another_host
арг 3: --user=bill
арг 4: x
Измененный вектор аргументов после load_defaults():
арг 0: ./show_opt
```

```
арг 1: --user=sampadm
арг 2: --password=secret
арг 3: --host=some_host
арг 4: -h
арг 5: yet_another_host
арг 6: --user=bill
арг 7: x
Параметры соединения после handle_options():
имя узла: yet_another_host
имя пользователя: bill
пароль: secret
номер порта: 0
имя разъема: (null)
Вектор аргументов после handle_options():
арг 0: x
```

Этот результат показывает, что имя узла выбирается из командной строки (перекрывая значение, заданное в конфигурационном файле), а имя пользователя и его пароль берутся из конфигурационного файла. Процедура `handle_options()` анализирует полученные параметры, независимо от способа их задания в короткой (`-h yet_another_host`) или длинной (`- -user=bill`) форме.

Функция подсказки `get_one_option()` используется с `handle_options()`. В программе `show_opt` ее функции сведены к минимуму и обрабатывают только параметры `--help` или `-?` (для которых функция `handle_options()` передает значение `optid`, равное `'?'`):

```
my_bool
get_one_option (int optid, const struct my_option *opt, char
*argument)
{
  switch (optid)
  {
    case '?':
    my_print_help (my_opts); /* распечатать сообщение подсказки */
    exit (0);
  }
  return (0);
}
```

Функция `my_print_help()` является программой клиентской библиотеки, которая автоматически передает подсказку на основании имени параметра и строк комментария в массиве `my_opts`. Для того чтобы проверить ее работу, необходимо запустить следующую команду, чтобы в конце вывода получить сообщение подсказки:

```
% ./show_opt --help
```

По необходимости в процедуру `get_one_option()` можно добавить и другие ветви. Например, такая функция может пригодиться при обработке параметра, в котором задается пароль. При определении такого параметра значение пароля может указываться или может быть опущено. Эта возможность отражена элементом `OPT_ARG` информационной структуры параметров (т.е. при использовании длинной формы вы можете задавать параметр как `--password` или `--password=`*`your_pass`* либо как `-p` или `-p`*`your_pass`* при использовании короткой формы задания параметров.) Клиенты MySQL обычно позволяют опускать

значение паролей в командной строке, чтобы запрашивать их потом. Это позволит не указывать пароль в командной строке и сохранить его в тайне. В последующих программах для проверки ввода пароля мы воспользуемся функцией `get_one_option()`. Если пароль был введен, его значение будет сохранено, в противном случае будет установлен флаг, который будет показывать, что перед попыткой пользователя подключиться к серверу должна выводиться подсказка.

Для начинающего программиста могут быть очень полезными упражнения по изменению структуры параметров в программе `show_opt.c`. Это позволит узнать, какое влияние оказывают эти изменения на поведение программы. Например, значения максимума, минимума и размера блока для параметра `--port` были установлены равными соответственно 100, 1000 и 25, а после перекомпиляции программы обнаружилось, что номер порта не может быть представлен значением, лежащим вне диапазона, начинающегося со 100 и завершающегося 1000, и что значения округляются до ближайшего меньшего числа, кратного 25.

Программы обработки параметров также автоматически обрабатывают параметры `--no-defaults`, `--print-defaults`, `--defaults-file` и `--defaults-extra-file`. Попробуйте вызвать `show_opt` с каждым из этих параметров и посмотрите, что при этом произойдет.

Подключение обработки параметров в клиентскую программу СУБД MySQL

Теперь с помощью простейших данных посмотрим, как работают подпрограммы обработки полученных параметров, а потом будем использовать их в качестве основы программы подключения к серверу, основываясь на любых данных, будь то данные, полученные из командной строки или из конфигурационного файла. Полученный исходный текст `client3.c` приобретет следующий вид:

```
/*
* client3.c - подключение к серверу MySQL с использованием параметров со-
единения,
* заданных в конфигурационном файле или в командной строке
*/

#include <string.h> /* для strdup() */
#include <my_global.h>
#include <mysql.h>
#include <my_getopt.h>

static char *opt_host_name = NULL; /* узел сервера (умолчание = localhost)
*/
static char *opt_user_name = NULL; /* имя пользователя (умолчание = ваше
имя при регистрации в системе) */
static char *opt_password = NULL; /* пароль (умолчание = отсутствует)*/
static unsigned int opt_port_num = 0; /* номер порта (используйте встроен-
ное значение) */
static char *opt_socket_name = NULL; /* имя разъема (используйте встроен-
ное значение) */
static char *opt_db_name = NULL; /* имя базы данных (умолчание=ничего) */
static unsigned int opt_flags = 0; /* флаги подключения (нет) */

static int ask_password = 0; /* где запрашивать пароль? */
```

```
static MYSQL *conn; /* указатель на дескриптор соединения */

static const char *client_groups[] = { "client", NULL };

static struct my_option my_opts[] = /* структуры для хранения параметриче-
ской информации */

{
   {"help", '?', "Вывести эту подсказку и выйти",
   NULL, NULL, NULL,
   GET_NO_ARG, NO_ARG, 0, 0, 0, 0, 0, 0},
   {"host", 'h', "Узел подключения",
   (gptr *) &opt_host_name, NULL, NULL,
   GET_STR_ALLOC, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
   {"password", 'p', "Пароль",
   (gptr *) &opt_password, NULL, NULL,
   GET_STR_ALLOC, OPT_ARG, 0, 0, 0, 0, 0, 0},
   {"port", 'P', "Номер порта",
   (gptr *) &opt_port_num, NULL, NULL,
   GET_UINT, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
   {"socket", 'S', "Путь к разъему",
   (gptr *) &opt_socket_name, NULL, NULL,
   GET_STR_ALLOC, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
   {"user", 'u', "Имя пользователя",
   (gptr *) &opt_user_name, NULL, NULL,
   GET_STR_ALLOC, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
   { NULL, 0, NULL, NULL, NULL, NULL, GET_NO_ARG, NO_ARG, 0, 0, 0, 0, 0, 0
}
};

void
print_error (MYSQL *conn, char *message)
{
   fprintf (stderr, "%s\n", message);
   if (conn != NULL)
   {
    fprintf (stderr, "Ошибка %u (%s)\n",
    mysql_errno (conn), mysql_error (conn));
   }
}

my_bool
get_one_option (int optid, const struct my_option *opt, char
*argument)
{
   switch (optid)
   {
    case '?':
        my_print_help (my_opts); /* распечатать сообщение подсказки */
        exit (0);
    case 'p': /* пароль */
        if (!argument) /* нет значения, оно будет введено позднее */
            ask_password = 1;
        else /* копировать пароль, стереть оригинал */
        {
            opt_password = strdup (argument);
            if (opt_password == NULL)
```

```
            {
                print_error (NULL, "нет возможности разместить буфер
пароля");
                exit (1);
            }
        while (*argument)
            *argument++ = 'x';
    }
    break;
  }
  return (0);
}
  }
  return (0);
}

int
main (int argc, char *argv[])
{
  my_init
  load_defaults ("my", groups, $argc, &argv);
  if ((opt_err = handle_options (&argc, &argv, my_opts,
get_one_option)))
    exit (opt_err);
  /* при необходимости запрос пароля */
  if (ask_password)
    opt_password = get_tty_password (NULL);

  /* получить имя базы данных, если оно содержится в командной строке
*/
  if (argc > 0)
  {
    opt_db_name = argv[0];
    --argc; ++argv;
  }

  /* инициализация дескриптора соединения */
  conn = mysql_init (NULL);
  if (conn == NULL)
  {
    print_error (NULL, "сбой mysql_init() (вероятно, не хватает памя-
ти) ");
    exit (1);
  }

  /* подключение к серверу */
  if (mysql_real_connect (conn, opt_host_name, opt_user_name,
opt_password,
    opt_db_name, opt_port_num, opt_socket_name, opt_flags) == NULL)
  {
    print_error (conn, "Сбой mysql_real_connect()");
    mysql_close (conn);
    exit (1);
  }

  /* ...здесь размещаются запросы и обрабатываются полученные резуль-
таты ... */
```

```
  /* отключение от сервера */
  mysql_close (conn);
  exit (0);
}
```

По сравнению с программами client1, client2 и show_opt, разработанными нами ранее, client3 имеет следующие особенности.

- Позволяет задавать имя базы данных непосредственно в командной строке. Это соответствует интерфейсу стандартных программ, которые поставляются с дистрибутивом СУБД MySQL.
- Удаляет любой пароль из вектора аргументов сразу же после его копирования. Таким образом, максимально уменьшается временной интервал, на протяжении которого пароль, заданный в командной строке, будет "виден" для команды ps или других аналогичных программ, имеющих доступ к информации о состоянии системы. (Интервал *уменьшается*, но не исчезает полностью. И все же метод, при котором пароль задается в командной строке, остается очень рискованным с точки зрения безопасности.)
- Параметр пароля может задаваться без значения. При этом функция get_one_option() устанавливает флаг, свидетельствующий о том, что программа должна будет запросить пароль у пользователя позднее. В таком случае main() после обработки всех параметров запрашивает пароль с помощью функции get_tty_password(). Эта подпрограмма из клиентской библиотеки запрашивает пароль, не повторяя его на экране. (В клиентской библиотеке таких "штучек" хватает. Рекомендуется также просматривать исходные тексты программы СУБД MySQL. Это позволит узнать больше о возможностях этих подпрограмм.) Правомочен вопрос: почему просто не вызвать функцию getpass()? Но далеко не все системы имеют эту функцию (в ОС Windows, например, такая возможность отсутствует). Функция get_tty_password() работает в любой системе.

Программа client3 работает в соответствии с заданными параметрами. Для усложнения ее задачи предположим, что конфигурационный файл отсутствует. Запустив client3 без каких-либо аргументов, вы подключаетесь к узлу localhost без пароля, под именем, с которым вы зарегистрировались в ОС UNIX. Запустите client3 с помощью командной строки, приведенной ниже, программа запросит пароль (сразу же после ключа -p пароль отсутствует), соединит вас с узлом some_host и передаст имя пользователя some_user на сервер вместе с введенным вами паролем:

```
% client3 -h some_host -p -u some_user some_db
```

Программа client3 передает имя базы данных some_db функции mysql_real_connect() и делает ее текущей базой данных. Если конфигурационный файл существует, обрабатывается его содержимое, которое затем используется для внесения соответствующих изменений в параметры соединения.

Выполненная нами операция является универсальной для любой клиентской программы MySQL: подключение к серверу с помощью соответствующих параметров. Конечно, необходимо знать, как подключаться. Но теперь вы уже это знаете, и детали этого процесса реализованы в тексте клиентской программы client3.c, так что об

этом можно уже не беспокоиться. Это значит, что можно полностью сфокусировать внимание на самом главном — возможности получить доступ к данным, хранящимся в базе данных. Все действия программы должны размещаться между вызовами `mysql_real_connect()` и `mysql_close()`. Но то, что у нас уже есть, можно использовать в качестве основы для создания самых разных программ. Для того чтобы написать новую программу, необходимо выполнить следующее.

1. Скопировать текст программы `client3.c`.
2. Внести изменения в цикл обработки параметров, при необходимости задействовав другие или дополнительные параметры.
3. Добавить код, реализующий ваш алгоритм обработки данных между вызовами `mysql_real_connect()` и `mysql_close()`.

И все готово.

Обработка запросов

Теперь нам известно, как начать и закончить "общение" с сервером. Посмотрим, как ведет себя само соединение. Этот раздел демонстрирует процедуру "общения" с сервером в процессе обработки запросов. Каждый запрос к базе данных предполагает выполнение следующих шагов.

1. **Создать запрос.** Как это делается, зависит от содержимого запроса, в частности от того, содержит ли он двоичные данные.
2. **Сделать запрос, послав его на сервер.** Сервер выполняет запрос и возвращает результат.
3. **Обработать результат запроса.** Это зависит от типа сделанного запроса. Например, оператор `SELECT` возвращает строки с данными, а оператор `INSERT` не возвращает.

Очень важную роль при создании запросов выполняет функция, с помощью которой запросы отсылаются на сервер. Чаще всего используется подпрограмма `mysql_real_query()`. С ее помощью запрос высылается в виде строки со счетчиком длины. Так как строка запроса передается вместе со счетчиком, ее содержимое может быть чем угодно, даже двоичными данными или пустыми байтами. В этом случае запрос не завершается символом "NULL".

Другая функция передачи запросов, `mysql_query()`, более ограничена в своих возможностях, но при этом и более удобна в использовании. Запросы, которые передаются функции `mysql_query()`, завершаются символом "NULL", что означает, что такие запросы не могут содержать символ "NULL". (Наличие в строке байтов со значением "NULL" просто приведет к ошибочной интерпретации такой строки как более короткой, чем она есть на самом деле.) В общем, если запрос создается для того, чтобы передавать произвольные данные, и существует вероятность, что они будут содержать пустые байты, использования функции `mysql_query()` следует избегать. Существенным преимуществом использования строк с ограничением символом "NULL" является возможность использования строковых функций, вероятно, из хорошо уже вам знакомой стандартной библиотеки C, таких как `strcpy()` и `sprintf()`.

Обратим внимание на еще один фактор, с которым обычно сталкиваешься при создании запросов: прибегаете ли вы к операциям, содержащим управляющие символы? Да, если создается запрос с использованием двоичных данных или других символов, таких как кавычки или обратная косая черта. Эта проблема будет обсуждаться в дальнейшем.

Простейший набросок кода, обрабатывающего запрос, выглядит следующим образом:

```
if (mysql_query (conn, query) != 0)
{
  /* ошибка, сообщение об ошибке */
}
else
{
  /* успешное выполнение; определить результаты выполнения запроса */
}
```

Как функция `mysql_query()`, так и функция `mysql_real_query()` возвращают 0 в случае успешного завершения запроса и ненулевое значение в случае ошибки. Успешное завершение запроса означает, что сервер не нашел в нем ошибки и смог его выполнить. Это никак не свидетельствует о результате выполнения запроса. Например, это не значит, что запрос `SELECT` выбрал хоть одну строку или что оператор `DELETE` удалил хотя бы одну строку. Проверка результата, полученного от выполнения запроса, требует дополнительной обработки.

Запрос может дать ошибку по самым различным причинам. Вот самые вероятные из них:

- в запросе содержится синтаксическая ошибка;
- в запросе содержится семантическая ошибка (например, в запросе делается обращение к столбцу, не существующему в таблице);
- у вас нет прав для доступа к данным, указанным в запросе.

Запросы можно разбить на две категории: запросы, возвращающие результаты, и запросы, не возвращающие результатов. Такие запросы, как `INSERT`, `DELETE` и `UPDATE`, попадают в категорию запросов, "не возвращающих результата". Они ничего не возвращают даже для запросов, которые модифицируют базу данных. Единственную информацию, которую можно от них получить, — это количество обработанных строк.

Такие запросы, как `SELECT` и `SHOW`, попадают в категорию запросов, "возвращающих результат", как-никак их предназначением является возвращение какого-то результата. Набор строк, возвращаемый таким запросом, называется *результирующим набором*. В MySQL C API это тип данных `MYSQL_RES`, представляющий собой структуру, содержащую значения данных в строках и метаданные (названия столбцов и длины значений данных). Пустой результирующий набор (т.е. такой, который содержит нуль строк) нельзя идентифицировать как "отсутствие результата".

Обработка запросов, не возвращающих результатов

Для обработки запросов, не возвращающих результирующих наборов, запрос генерируется с помощью функции `mysql_query()` или `mysql_real_query()`. При

успешном завершении запроса можно будет узнать, сколько строк было добавлено, удалено или модифицировано. Для этого достаточно вызвать функцию mysql_affected_rows():

```
if (mysql_query (conn, "INSERT INTO my_tbl SET name =
                  'My Name'") != 0)
{
  print_error ("Сбой оператора INSERT");
}
else
{
printf ("Оператор INSERT успешно отработал, обработано: %lu строк\n",
        (unsigned long) mysql_affected_rows (conn));
}
```

Обратите внимание, каким образом результат mysql_affected_rows() "преобразуется" в переменную типа unsigned long для вывода на печать. Эта функция возвращает значение типа my_ulonglong. Так сделано потому, что печатать значение этого типа напрямую в некоторых системах нельзя. (Например, я убедился, что она работает под UNIX FreeBSD, но дает сбой при работе в ОС Solaris.) Преобразование типа значения в тип unsigned long и указание типа формата "%lu" решает эту проблему. Подобное рассуждение применимо к любым функциям, возвращающим значения типа my_ulonglong, таким как mysql_num_rows() и mysql_insert_id(). Помните это, если хотите, чтобы ваши клиентские программы были переносимы.

Функция mysql_rows_affected() возвращает количество строк, обработанных запросом, но значение самого термина "обработано строк" зависит от типа запроса. Для запросов INSERT, REPLACE или DELETE это количество добавленных, замещенных или удаленных строк соответственно. Для запроса UPDATE это означает количество реально модифицированных строк. СУБД MySQL не модифицирует строку, которая содержит те же данные, что и новые. Это значит, что хотя строка и может быть выбрана для модификации (с помощью предложения WHERE оператора UPDATE) она не будет реально изменена.

Вообще, для оператора UPDATE термин "обработано строк" является неоднозначным. Это может означать "совпадающих строк", т.е. количество строк, выбранных для модификации, даже если операция модификации реально не меняет их значений. Существует возможность настроить систему таким образом. Для этого во время подключения к серверу нужно передать процедуре mysql_real_connect() значение флага CLIENT_FOUND_ROWS. Значение CLIENT_FOUND_ROWS также можно передать как аргумент и процедуре mysql_real_connect().

Обработка запросов, возвращающих результаты

Запросы, возвращающие данные, делают это в виде результирующего набора. Они могут быть получены после передачи запроса на сервер с помощью функции mysql_query() или mysql_real_query(). Следует помнить, что запрос SELECT в СУБД MySQL не является единственным запросом, возвращающим строки. К запросам такого типа можно отнести запросы SHOW, DESCRIBE и EXPLAIN. Для всех этих запросов необходимо обработать строки, полученные после запроса.

Обработка результирующего набора предполагает следующие шаги.

1. **С помощью функции `mysql_store_result()` или `mysql_use_result()` получить результирующий набор.** Эти функции возвращают указатель на MYSQL_RES в случае успешного завершения или значение "NULL" в случае ошибки. Позже мы остановимся на различиях между `mysql_store_result()` и `mysql_use_result()` и условиях, при которых можно будет предпочесть одну функцию другой. Здесь в примерах мы будем использовать функцию `mysql_store_result()`, немедленно возвращающую полученные строки с сервера и сохраняющую их в областях данных клиента.
2. **Для каждой строки результирующего набора вызовем функцию `mysql_fetch_row()`.** Эта функция возвращает значение MYSQL_ROW, которое представляет собой указатель на массив строк, представляющих значения каждого столбца строки. Что вы будете делать со строкой, зависит от задач приложения. Можно просто распечатать значения в столбцах и на их основе произвести какие-то статистические вычисления или выполнить какие-то другие действия.
3. **После завершения всех операций с результирующим набором для очистки памяти, которую он использует, необходимо вызвать функцию `mysql_free_result()`.** Если этого не сделать, ваше приложение может вызвать переполнение оперативной памяти. (Очень важно делать это для приложений, работающих длительное время; в противном случае скоро станет заметно, что система все чаще будет занята процессами, которые будут поглощать все больше ресурсов системы.)

В следующем примере рассмотрим обработку результирующего набора, полученного после выполнения запроса:

```
MYSQL_RES *res_set;

if (mysql_query (conn, "SHOW TABLES FROM mysql") != 0)
  print_error (conn, "Сбой mysql_query()");
else
{
  res_set = mysql_store_result (conn); /* генерировать результирующий
набор */
  if (res_set == NULL)
    print_error (conn, "Сбой mysql_store_result()");
  else
  {
    /* обработать результирующий набор, а затем удалить его */
    process_result_set (conn, res_set);
    mysql_free_result (res_set);
  }
}
```

Здесь мы совсем обошли вниманием вызов функции `process_result_set()`, с помощью которой обрабатывается каждая строка из результирующего набора. Сделаем это. В общем случае функции обработки результирующего набора основаны на цикле, имеющем следующий вид:

```
MYSQL_ROW row;
```

```
while ((row = mysql_fetch_row (res_set)) != NULL)
{
  /* произвести определенные операции над содержимым строки */
}
```

Функция `mysql_fetch_row()` возвращает в `MYSQL_ROW` значение, которое является указателем на массив значений. Таким образом, доступ к каждому значению можно осуществить с помощью выражения `row[i]`, где `i` лежит в диапазоне от нуля до количества столбцов в строке минус один. Вот несколько особенностей типа данных `MYSQL_ROW`, которые необходимо иметь в виду при работе с ним.

- Тип `MYSQL_ROW` является указательным типом. Таким образом, переменные этого типа необходимо объявлять как `MYSQL_ROW row`, а не как `MYSQL_ROW *row`.
- Значения данных всех типов возвращаются в массив `MYSQL_ROW` как строки. Для обработки числовых значений как чисел их необходимо преобразовать из строкового формата в числовой самостоятельно.
- Строки массива `MYSQL_ROW` завершаются пустым значением. Однако если столбец содержит двоичные данные, он может содержать и пустые байты. Такую строку нельзя рассматривать как строку, ограниченную пустым значением. Получите длину столбца, чтобы определить, какую длину может иметь значение столбца. (Каким образом это можно сделать, вы узнаете в разделе "Метаданные результирующего набора".)
- Значения "NULL" в массиве `MYSQL_ROW` представляются указателями "NULL". За исключением случая, когда столбец объявлен как "NOT NULL", всегда необходимо проверять, являются ли значения в столбцах указателями со значением "NULL".

Приложения, создаваемые вами, могут делать с данными в каждой строке все, что угодно. Для иллюстрации просто распечатаем строки с табуляциями между столбцами в качестве разделителей. Для этого нам потребуется дополнительная функция из клиентской библиотеки `mysql_num_fields()`. Эта функция возвращает количество столбцов в строке.

Вот программный код для `process_result_set()`:

```
void
process_result_set (MYSQL *conn, MYSQL_RES *res_set)
{
MYSQL_ROW      row;
unsigned int   i;

  while ((row = mysql_fetch_row (res_set)) != NULL)
  {
    for (i = 0; i < mysql_num_fields (res_set); i++)
    {
        if (i > 0)
            fputc ('\t', stdout);
        printf ("%s", row[i] != NULL ? row[i] : "NULL");
        }
        fputc ('\n', stdout);
    }
    if (mysql_errno (conn) != 0)
```

```
        print_error (conn, "mysql_fetch_row() failed");
    else
        printf ("Возвращено строк %lu \n",
                    (unsigned long) mysql_num_rows (res_set));
}
```

Программа `process_result_set()` распечатывает каждую строку в формате разделенных табуляцией столбцов (отображая значения "NULL" словом NULL) с последующим количеством выбранных строк. Это количество получено с помощью функции `mysql_num_rows()`. Как функции `mysql_affected_rows()` и `mysql_num_rows()`, эта функция возвращает значение типа `my_ulonglong`. Таким образом, преобразуем в тип `unsigned long` и с помощью формата "%lu" распечатаем его. Но заметим, что, в отличие от функции `mysql_affected_rows()`, которая принимает аргумент дескриптора соединения, функция `mysql_num_rows()` принимает в качестве аргумента указатель на результирующий набор.

За циклом выборки строк следует тестирование на ошибки. Это всего лишь мера предосторожности. При создании результирующего набора с помощью `mysql_store_result()` значение "NULL", возвращаемое из `mysql_fetch_row()`, будет означать "больше строк нет". Однако при создании результирующего набора с помощью `mysql_use_result()` значение "NULL", возвращаемое из `mysql_fetch_row()`, может означать как "больше строк нет", так и то, что произошла ошибка. Это тестирование просто дает возможность программе `process_result_set()` определить, произошла ли ошибка, независимо от метода создания результирующего набора.

Эта версия `process_result_set()` взяла на вооружение минималистский подход к выводу значений столбцов, которому присущи определенные недостатки. Например, предположим, что выполняется запрос

```
SELECT last_name, first_name, city, state FROM president
ORDER BY last_name, first_name
```

Он даст следующий результат:

```
Adams       John            Braintree       MA
Adams       John Quincy     Braintree       MA
Arthur      Chester A.      Fairfield       VT
Buchanan    James           Mercersburg  PA
Bush        George W.           Milton      MA
Bush        George W.           New Haven   CT
Carter      James E.Jr      Plains       GA
...
```

Этот вывод можно "облагородить", добавив наименования столбцов и выровняв столбцы по вертикали. Для этого необходимо узнать название столбцов и самое длинное значение в каждом столбце. Эта информация имеется в наличии, но не как часть данных, а как часть метаданных результирующего набора (т.е. данных о данных). После небольшого усовершенствования кода обработки запроса в разделе "Метаданные результирующего запроса" мы сможем усовершенствовать формат вывода данных.

Печать двоичных данных

Значения столбцов, содержащих пустые данные, могут не сработать правильно со спецификацией формата "%s" оператора `printf()`. Оператор `printf()` ожидает строку, завер-

шающуюся пустым значением, и распечатывает строку только до пустого значения. Для двоичных данных лучше всего воспользоваться длиной столбца. Таким образом, можно будет распечатать все значение. Для этого можно воспользоваться операторами `fwrite()`.

Обработка общих запросов

Все предыдущие примеры обработки запросов были написаны исходя из того, возвращает или не возвращает оператор результирующие данные. Это было возможно благодаря тому, что текст самого запроса был включен в код. Мы использовали оператор `INSERT`, который не возвращал результирующий набор, и оператор `SHOW TABLES`, который его возвращал.

Однако зачастую нельзя заранее знать, к какому типу можно отнести запрос. Например, при выполнении запроса, который вводится с клавиатуры или из файла, это может быть произвольный оператор, т.е. заранее нельзя предсказать, возвращает ли этот файл строки. Что делать в таком случае? Выполнить синтаксический анализ запроса для того, чтобы определить, какой тип он представляет? Это не совсем просто. Недостаточно также проверить первое слово в строке, тем более, если это будет комментарий:

```
/* комментарий */ SELECT ...
```

К счастью, совсем не обязательно знать тип запроса заранее для того, чтобы обработать его соответствующим образом. Интерфейс MySQL C API позволяет создавать программы, безошибочно обрабатывающие запросы любого типа, независимо от того, возвращает или не возвращает он результирующий набор.

Перед тем как его создать, определим, как он работает.

1. Делаем запрос. Если он дает сбой, завершаем работу.
2. Запрос срабатывает без ошибок. Для выборки строки с сервера и создания результирующего набора вызываем функцию `mysql_store_result()`.
3. Функция `mysql_store_result()` выполняется успешно, и запрос возвращает результирующий набор. Строки необходимо обрабатывать с помощью функции `mysql_fetch_row()` до тех пор, пока она не возвратит значение "NULL".
4. Сбой функции `mysql_store_result()` может быть вызван тем, что запрос не возвращает результирующего набора или в процессе выборки произошла ошибка. Различить эти два результата можно, если вызвать функцию `mysql_field_count()` и проверить результат, возвращаемый ею.
   - Функция `mysql_field_count()` возвращает 0; это означает, что запрос не возвращает результирующего набора. (Это был запрос `INSERT`, `DELETE` или `UPDATE`.)
   - Функция `mysql_field_count()` возвращает значение, отличное от нуля. Запрос должен возвратить результирующий набор, но произошла ошибка. Это может случиться по самым различным причинам. Например, результирующий набор может быть слишком велик и для его размещения не хватило памяти или во время выборки результатов с сервера произошел какой-то сбой в сети.

Ниже приведена распечатка программы, обрабатывающей запрос любого типа.

```
void
process_query (MYSQL *conn, char *query)
{
MYSQL_RES *res_set;
unsigned int field_count;

   if (mysql_query (conn, query) 1= 0) /* сбой запроса */
   {
     print_error (conn, "Невозможно выполнить запрос");
     return;
   }


   /* запрос завершил работу успешно; определить,
        вернул ли он данные */

   res_set = mysql_store_result (conn);
   if (res_set)   /* результирующий набор возвращен */
   {
     /* обработать полученные строки, очистить результирующий набор */
         process_result_set (conn, res_set);
     mysql_free_result (res_set);
     }
   else /* результирующий набор не был возвращен */
     {
         /*
         * означает ли отсутствие результирующего набора, что запрос
его не
         * возвратил или что он должен был возвратить, но произошла
ошибка?
         */
         if (mysql_field_count (conn) == 0)
         {
             /*
             * запрос не сгенерировал результирующий набор (это не был
оператор SELECT, SHOW,
             * DESCRIBE и т.д.), таким образом, отображается только ко-
личество обработанных строк
             */
             printf ("обработано %lu строк \n",
             (unsigned long) mysql_affected_rows (conn));
         }
         else /* произошла ошибка */
         {
             print_error (conn, "Нельзя найти результирующий набор ");
         }
     }
   }
}
```

Ситуацию немного усложняет тот факт, что функции `mysql_field_count()` до версии 3.22.24 не существовало. В более ранних версиях вместо нее можно было воспользоваться функцией `mysql_num_fields()`. Для того чтобы программа могла работать с более ранними версиями MySQL, в программный код, работающий с `mysql_field_count()`, после `mysql.h` и перед `mysql_field_count()` включите следующий фрагмент:

```
#if !defined(MYSQL_VERSION_ID) || MYSQL_VERSION_ID<32224
#define mysql_field_count mysql_num_fields
#endif
```

Он позволит при работе с СУБД MySQL до версии 3.22.24 вместо вызова `mysql_field_count()` делать вызов `mysql_num_fields()`.

Альтернативные методы обработки запросов

Эта версия процедуры `process_query()` имеет три особенности.

- С помощью `mysql_query()` она вызывает запрос.
- Для выборки результирующего набора используется процедура `mysql_store_query()`.
- Если результирующий набор не получен, вызывается процедура `mysql_field_count()`. Она должна распознать ситуацию, была ли это ошибка, или результирующего набора вообще быть не должно.

Для всех этих особенностей обработки запроса можно предложить альтернативные варианты.

- Вместо запроса, использующего ограниченные символом "NULL" строки и функции `mysql_query()`, воспользоваться строкой со счетчиком и с функцией `mysql_real_query()`.
- Для получения результирующего набора необходимо заменить функцию `mysql_store_result()` функцией `mysql_use_result()`.
- В процессе определения, произошла ли ошибка, или просто этот запрос ничего не должен возвращать, отказаться от использования функции `mysql_field_count()` в пользу функции `mysql_error()`.

Любой из этих вариантов или все сразу можно использовать в процедуре `process_query()`. Ниже приведена процедура `process_real_query()`, аналогичная процедуре `process_query()`, но в ней использованы все три варианта:

```
void
process_real_query (MYSQL *conn, char *query, unsigned int len)
{
MYSQL_RES *res_set;
unsigned int field_count;
   if (mysql_real_query (conn, query, len) != 0) /* the query failed */
   {
     print_error (conn, "Невозможно выполнить запрос");
   return;
   }
   /* запрос завершил работу успешно; определить, вернул ли он данные */
```

```
    res_set = mysql_use_result (conn);
    if (res_set) /* результирующий набор возвращен */
    {
      /* обработать строки и очистить результирующий набор */
      process_result_set (conn, res_set);
      mysql_free_result (res_set);
    }
    else /* результирующий набор не возвращен */
    {
      /*
      * означает ли отсутствие результирующего набора,
      * что произошла ошибка или не вернулся результирующий набор?
      */
      if (mysql_errno (conn) == 0)
      {
          /*
          * результирующий набор не возвращен (это не был запрос SELECT,
SHOW, DESCRIBE и т.д.),
          * верните количество строк, обработанных запросом
          */
          printf ("обработано %lu строк \n", (unsigned long)
mysql_affected_rows (conn));
      }
      else /* произошла ошибка */
      {
          print_error (conn, "Невозможно обработать результирующий
                       набор ");
      }
    }
}
```

Сравнение возможностей функций mysql_store_result() и mysql_use_result()

Функции mysql_store_result() и mysql_use_result() подобны в том, что обе принимают аргумент дескриптора соединения и возвращают результирующий набор. Но между ними есть существенные различия. Первое и главное различие заключается в том, каким образом результирующий набор выбирается с сервера. Функция mysql_store_result() выбирает все строки сразу же после ее вызова. Функция mysql_use_result() инициирует выборку, но в действительности не получает строк. Эти различия в выборке строк лежат в основе всех остальных различий между этими двумя функциями. В этом разделе их характеристики сравниваются для того, чтобы разработчик сделал осознанный выбор программного обеспечения.

Функция mysql_store_result() в процессе выборки результирующего набора с сервера производит собственно выборку, распределяет для него память и сохраняет его на компьютере клиента. Последующий вызов mysql_fetch_row() никогда не возвращает ошибки, так как она просто выбирает строку из структуры данных, которые уже содержат результирующий набор. Возвращаемое функцией mysql_fetch_row() значение "NULL" означает только, что достигнут конец результирующего набора.

Однако mysql_use_result() никогда сама не выбирает строк. Она просто инициирует построчную выборку, которая будет завершена, если для каждой строки вы-

звать процедуру mysql_fetch_row(). В этом случае возвращение значения "NULL" из процедуры mysql_fetch_row() может означать как достижение конца результирующего набора, так и возникновение ошибки в процессе работы сервера. Разобраться в различиях можно, только вызвав функции mysql_errno() или mysql_error().

Процедура mysql_store_result() потребляет больше памяти и вычислительных ресурсов, чем mysql_use_result(). Это происходит вследствие того, что вся обработка результирующего набора производится клиентской программой. При необходимости одновременной выборки большого количества данных может создаться ситуация, когда лучше воспользоваться процедурой mysql_use_result().

Для выполнения процедуры mysql_use_result() требуется меньший объем памяти: ей достаточно столько памяти, сколько необходимо для размещения одной строки. Эта функция будет работать быстрее, потому что не потребуется проводить реконфигурацию в зависимости от получаемых объемов. Но функция mysql_use_result() сильно перегружает сервер, который должен хранить все строки результирующего набора до тех пор, пока клиент не соберется их выбрать. Следовательно, функция mysql_use_result() — не самый лучший выбор в качестве инструмента разработки программ определенного класса.

- Интерактивные клиентские программы, выполняющие построчный просмотр по запросу пользователя. (Сервер не должен ожидать посылки следующей строки только потому, что пользователь делает короткий перерыв в работе.)
- Клиентские программы, выполняющие между выборками строк большой объем вычислений.

В обоих случаях клиент не имеет возможности быстро выбрать все строки в результирующий набор. Это снижает возможности сервера и может отрицательно повлиять на работу других клиентов, так как таблицы, из которых производится чтение, закрыты для чтения на все время, пока происходит выполнение запроса. Действия любого клиента, пытающегося внести изменения в эти таблицы или вставить новые строки, будут блокироваться.

Возмещением расходов на дополнительную память, требующуюся функции mysql_store_result(), является возможность получения доступа одновременно ко всему результирующему набору. Функции mysql_data_seek(), mysql_row_seek() и mysql_row_tell() позволяют осуществить доступ к строкам в любом нужном порядке. При работе с функцией mysql_use_result() доступ к строкам возможен только в том порядке, в котором они были выбраны функцией mysql_fetch_row(). При необходимости обрабатывать строки в любом другом порядке, отличающемся от последовательности, в которой они возвращены с сервера, нужно воспользоваться возможностями функции mysql_store_result(). Например, если разрабатываемое приложение должно разрешить пользователю просматривать выбранные строки вперед и назад, более подходящего решения, чем mysql_store_result(), найти трудно.

Работая с функцией mysql_store_result(), можно получить такую информацию о столбцах, которую невозможно получить с помощью mysql_use_result(). Количество строк, полученных в результирующем наборе, можно получить с помощью функции mysql_num_rows(). Максимальная длина значений в каждой колонке сохраняется в элементе max_width структуры MYSQL_FIELD, хранящей информа-

цию о столбцах обрабатываемой таблицы. Функции mysql_use_result() и mysql_num_rows() не возвращают точного значения до тех пор, пока не будут выбраны все строки, а max_width будет недоступна, так как она может быть просчитана только после того, как будут видны все данные во всех столбцах.

Так как функция mysql_use_result() выполняет меньше обработки, чем mysql_store_result(), она требует соблюдения особого условия: программа-клиент *должна* вызывать функцию mysql_fetch_row() для каждой строки в результирующем наборе. В противном случае любая оставшаяся в наборе запись попадает в результирующий набор следующего запроса, что вызывает появление ошибки "out of sync" ("рассинхронизация"). (Этого можно избежать, вызвав mysql_free_result() перед запуском второго запроса. Функция mysql_free_result() выберет и отменит все строки, ожидающие своей очереди.) Единственным затруднением в данном случае является то, что при использовании функции mysql_use_result() можно работать только с одним результирующим набором.

Такая ситуация не может возникнуть при использовании функции mysql_store_result() в принципе, так как в тот момент, когда эта функция завершает работу, все строки результирующего набора уже выбраны. Действительно, при работе с функцией mysql_store_result() необходимость вызывать функцию mysql_fetch_row() полностью отпадает. Такая возможность может понадобиться при обработке запросов, результаты которых нас интересуют только с той точки зрения, что получен пустой или непустой результирующий набор данных, а не собственно результирующие данные. Например, для того, чтобы проверить факт существования таблицы mytbl, нужно выполнить запрос:

```
SHOW TABLES LIKE "mytbl"
```

Пусть после отработки функции mysql_store_result() mysql_num_rows() возвратит ненулевое значение. Следовательно, таблица существует. Функцию mysql_fetch_row() вызывать не надо.

Результирующие наборы, полученные с помощью mysql_store_result(), в определенной точке должны освобождаться функцией mysql_free_result(), но это совсем не обязательно делать перед выдачей следующего запроса. Это значит, что можно генерировать множество результирующих наборов и обрабатывать их одновременно, что резко контрастирует с методологией обработки "один результирующий набор в один момент времени", который применяется при работе с функцией mysql_use_result().

Можно предоставить пользователям самим выбирать способ обработки данных. Кстати, программы mysql и mysqldump такой выбор предоставляют. По умолчанию они используют mysql_store_result(). Но имеется возможность переключения на режим использования mysql_use_result(). Для этого необходимо задать параметр - -quick.

Метаданные результирующего набора

Результирующий набор содержит не только значения, хранящиеся в столбцах, но и информацию о самих данных. Эта информация называется *метаданными результирующего набора* и включает следующие элементы.

- Количество строк и столбцов в результирующем наборе, которые можно получить вызовом функций `mysql_num_rows()` и `mysql_num_fields()`.
- Длина каждого столбца в строке, которую можно получить вызовом функции `mysql_fetch_lengths()`.
- Такая информация о каждой строке, как имя столбца и его тип, максимальная длина значения, хранящегося в каждом столбце, и имя таблицы, которой этот столбец принадлежит. Эти данные хранятся в структурах типа `MYSQL_FIELD`, которые можно получить, вызвав `mysql_fetch_field()`. Структура `MYSQL_FIELD` детально описана в приложении E, "Программный интерфейс приложений C". Там же можно найти список всех функций, обеспечивающих доступ к информации, содержащейся в структуре.

Доступность метаданных частично зависит от метода обработки результирующего набора. Как было показано в предыдущем разделе, если необходимо узнать количество строк или максимальные длины значений столбцов для создания результирующего набора, следует вызывать функцию `mysql_store_result()`, а не `mysql_use_result()`.

Метаданные результирующего набора нужны, чтобы выбрать метод обработки данных результирующего набора.

- Имя и длина столбца нужны для планирования формата вывода данных.
- Количество строк результирующего набора необходимо для программирования цикла обработки строк.
- Счетчик строк или столбцов можно использовать, если необходимо определить объемы памяти под структуры данных, которые будут получены в результирующем наборе.
- Возможность определения типа данных столбца, что позволяет определить тип данных, хранящихся в столбце.

Выше, в разделе "Обработка запросов, возвращающих результаты", была представлена версия процедуры `process_result_set()`, которая распечатывает строки результирующего набора в формате разделения столбцов символами табуляции. В определенных случаях это хорошо (тогда, когда данные импортируются в электронные таблицы), но это не совсем то, что нужно для визуального изучения или получения распечаток. Помните, результат ранней версии `process_result_set()` имел вид

```
Adams    JohnBraintree    MA
Adams    John Quincy  Braintree    MA
Arthur   Chester A.   Fairfield    VT
Buchanan James    Mercersburg  PA
BushGeorge W.     Milton   MA
Bush George W. New Haven CT
Carter   James E.Jr   Plains   GA
...
```

Внесем изменения в процедуру `process_result_set()`. Теперь столбцы будут иметь заголовки, печатная форма будет разграфлена. Такая форма представления более репрезентативна.

```
+-----------+------------+-------------+-------+
| last_name | first name |    city     | state |
+-----------+------------+-------------+-------+
| Adams     | John       | Braintree   | MA    |
| Adams     | John Quincy| Braintree   | MA    |
| Arthur    | Chester A. | Fairfield   | VT    |
| Buchanan  | James      | Mercersburg | PA    |
| Bush      | George W.  | Milton      | MA    |
| Bush      | George W.  | New Haven   | CT    |
| Carter    | James E.Jr | Plains      | GA    |
| ...       | ...        | ...         | ...   |
+-----------+------------+-------------+-------+
```

Основные операции алгоритма отображения данных следующие.

1. Определить ширину отображения каждого столбца.
2. Распечатать строку с именами столбцов (ограничено вертикальными линиями, предваряется и завершается строками с тире).
3. Распечатать значения каждой строки результирующего набора (каждый столбец которого расположен в соответствующей графе). Кроме того, цифры выравниваются по правому краю, и для пустых значений распечатывается "NULL".
4. В конце распечатывается счетчик выбранных строк.

Это хороший пример того, каким образом можно использовать метаданные результирующего набора. Для того чтобы создать такое отображение данных, нужно знать не только значения данных, которые содержат строки, но и более детальную информацию.

Конечно, можно подумать, что это очень напоминает то, как клиентская программа `mysql` отображает данные. Да, это так, но вы можете сравнить исходный текст `mysql` с кодом, который мы получим в конце концов в модифицированном коде `process_result_set()`. Они не похожи друг на друга. И будет очень полезно на реальном примере сравнить два подхода к решению одной и той же проблемы.

Сначала необходимо определить длину отображаемых столбцов. Ниже приведена распечатка, показывающая, как это делается. Обратите внимание на тот факт, что все вычисления целиком основываются на метаданных результирующего набора данных, и нет никаких ссылок на значения данных в строках.

```
MYSQL_FIELD   *field;
unsigned long col_len;
unsigned int  i;

/* определение длины отображаемых столбцов */
/* создано функцией mysql_store_result(), а не mysql_use_result() */
mysql_field_seek (res_set, 0);
for (i = 0; i < mysql_num_fields (res_set); i++)
{
   field = mysql_fetch_field (res_set);
   col_len = strlen (field->name);
   if (col_len < field->max_length)
     col_len = field->max_length;
   if (col_len = 4;    /* 4 = длине слова "NULL"  */
     field->max_length = col_len;         /* обновить информацию
                                             о столбце}
}
```

Здесь в последовательных итерациях вычисляется длина столбцов из структур MYSQL_FIELD для результирующего набора. Позиционирование на первую структуру осуществляется вызовом функции mysql_fetch_seek(). Последующие вызовы mysql_fetch_field() возвращают указатели на следующие столбцы. Ширина столбца выбирается как максимум трех величин, каждая из которых зависит от метаданных информационной структуры столбца.

- Длины field->name титула столбца.
- Длины field->max_length самого длинного значения в столбце.
- Длины строки "NULL", если столбец содержит пустое значение. Значение field->flags указывает, может ли столбец содержать пустые данные.

После того как длина отображения вычислена, она присваивается элементу max_length структуры, которую мы получаем из клиентской библиотеки. Допустим, такой прием или элементы структуры MYSQL_FIELD должны быть открыты только для чтения. В общем случае, "только для чтения", но некоторые из клиентских программ дистрибутива СУБД MySQL ведут себя со значением max_length аналогичным образом. Так что можно сказать, что все нормально. (Если вы предпочитаете не использовать max_length, объявите массив типа unsigned long и храните в нем полученные значения длин.)

Вычисление длины столбца имеет один недостаток. Вспомним, что max_length не имеет никакого смысла, если результирующий набор создается с помощью функции mysql_use_result(). По этой же причине (необходимость вычисления max_length) для реализации этого алгоритма лучше подходит функция mysql_store_result().

Когда нам известна ширина столбца, мы готовы к печати. Печать титульной части таблицы несложна. Воспользуемся для этого информационной структурой столбца и с учетом ранее вычисленной длины распечатаем имя элемента структуры:

```
printf (" %-*s |", (int) field->max_length, field->name);
```

Для распечатки данных мы просто просмотрим строки в результирующем наборе, распечатывая при каждой итерации значение текущего столбца. Это достаточно сложный процесс, поскольку значение может быть равно "NULL" или может представлять число (в этом случае печать выравнивается по правому краю). Значения столбцов распечатываются следующим образом. Здесь row[i] содержит значения данных, а field — указатель на информацию о столбце.

```
if (row[i] == NULL) /* печать слова "NULL" */
  printf (" %-*s |", (int) field->max_length, "NULL");
else if (IS_NUM(field->type)) /* печать значения, выровненного по пра-
вому краю */
  printf (" %*s |", (int) field->max_length, row[i]);
else /* печать значения, выровненного по левому краю */
  printf (" %-*s |", field->max_length, row[i]);
```

Макрос IS_NUM() принимает значение "истина", если тип столбца, который указывается field->type, является числовым (INT, FLOAT или DECIMAL).

Окончательный программный код отображения результирующего набора данных имеет следующий вид. Процедура печати тире выделена в отдельную функцию print_dashes():

```
void
print_dashes (MYSQL_RES *res_set)
{
MYSQL_FIELD *field;
unsigned int i, j;

   mysql_field_seek (res_set, 0);
   fputc (' + ', stdout);
   for (i = 0; i < mysql_num_fields (res_set); i++)
   {
     field = mysql_fetch_field (res_set);
     for (j = 0; j < field->max_length + 2; j++)
         fputc (' -', stdout);
     fputc ('+', stdout);
   }
   fputc ('\n', stdout) ;
}

void
process_result_set (MYSQL *conn, MYSQL_RES *res_set)
{
MYSQL_ROW row;
MYSQL_FIELD *field;
 unsigned long i, col_len;

   /* определить длину отображаемого столбца - для этого требуется,
чтобы результирующий набор */
   /* был создан с помощью функции mysql_store_result(), а не
mysql_use_result() */
   mysql_field_seek (res_set, 0);
   for (i = 0; i < mysql_num_fields (res_set); i++)
   {
     field = mysql_fetch_field (res_set);
     col_len = strlen (field->name);
     if (col_len < field ->max_length)
         col_len = field ->max_length;
     if (col_len < 4 && !IS_NOT_NULL (field->flags))
         col_len =4;    /* 4 = длине слова "NULL" */
     field->max_length = col_len;         /* обновить информацию
                                             о столбце}
   }

   print_dashes (res_set);
   fputc ( '|', stdout);
   mysql_field_seek (res_set, 0);
   for (i = 0; i < mysql_num_fields (res_set); i++)
   {
     field = mysql_fetch_field (res_set);
     printf (" %-*s |", (int) field ->max_length, field->name);
   }
   fputc ( '\n', stdout);
   print_dashes ( res_set );

   while ((row = mysql_fetch_row (res_set)) != NULL)
   {
     mysql_field_seek (res_set, 0);
     fputc ( '|', stdout);
     for (i = 0; i < mysql_num_fields (res_set); i++)
```

```
    {
        field = mysql_fetch_field (res_set);
        if (row[i] == NULL) /* печать слова "NULL" */
            printf (" %-*s |", (int) field ->max_length, "NULL");
        else if (IS_NUM (field->type)) /* печать значения, выровненно-
го по правому краю */
            printf (" %*s |", (int) field->max_length, row[i]);
        else /* печать значения, выровненного по левому краю */
            printf (" %-*s |", (int) field->max_length, row[i]);
    }
    fputc ( ' \n', stdout);
  }
  print_dashes (res_set);
  printf ("обработано %lu строк \n", (unsigned long) mysql_num_rows
(res_set));
}
```

Клиентская библиотека СУБД MySQL поддерживает несколько способов доступа к структурам, содержащим информацию о столбцах. Например, код в предыдущем примере получает доступ к этим структурам несколько раз с помощью цикла такого вида:

```
mysql_field_seek (res_set, 0);
for (i = 0; i < mysql_num_fields (res_set);
{
  field = mysql_fetch_field (res_set);
  ...
}
```

Однако комбинация `mysql_field_seek()`/`mysql_fetch_field()` является единственным методом доступа к структурам `MYSQL_FIELD`. Более детальная информация изложена в разделах, посвященных функциям `mysql_fetch_fields()` и `mysql_fetch_field_direct()` приложения E, "Программный интерфейс приложений C".

Client4 — программа интерактивного ввода запросов

Теперь соберем вместе все сделанные нами наработки и на базе этого попробуем создать простейшую программу-клиент, обладающую интерактивными возможностями. Ее задача заключается в вводе запросов, выполнении их с помощью обработчика запросов общего назначения `process_query()` и отображении полученных результатов с помощью процедуры `process_result_set()`.

Программа `client4` в некотором смысле аналогична программе `mysql`, хотя, конечно, не имеет таких широких возможностей. Есть некоторые ограничения на то, что эти клиенты позволяют в качестве ввода:

- каждая вводимая строка должна содержать один полный запрос;
- запросы не должны завершаться точкой с запятой или "`\g`";
- для выхода нужно пользоваться комбинацией клавиш <Ctrl+D>, командой "`quit`" и ее аналогом — "`\q`".

Оказывается, что клиентские программы очень просто написать (это немногим более 10 строк нового кода). Почти все, что нам нужно, уже присутствует в "скелете" клиентской программы (`client3.c`) и в другом программном коде, который уже был написан нами. Осталось только добавить цикл, который собирает входные строки и выполняет их.

Для создания программы-клиента `client4.c` сначала скопируем "скелет" программы `client3.c` в файл `client4.c`. Затем добавим вызовы `process_query()`, `process_result_set()` и `print_dashes()`. Наконец, найдем следующую строку в `main()`:

```
/* поместите сюда основные операции */
```

И заменим их следующими операциями:

```
while (1)
{
char buf[10000];
  printf (stderr, "query> ");     /* вывод подсказки */
  if (fgets (buf, sizeof (buf), stdin) == NULL)   /* прочесть
                                                     запрос */

  break;
  if (strcmp (buf, "quit\n") == 0 || strcmp (buf, "\\q\n") == 0)
  break;
  process_query (conn,  buf);     /* выполнение запроса */
}
```

Теперь откомпилируйте `client4.c`, чтобы получить объектный код `client4.o`, скомпонуйте `client4.o` с `common.o` и клиентской библиотекой, и дело сделано! В результате в нашем распоряжении будет новая клиентская программа, которая может выполнить любой запрос и отображать полученные результаты. Следующий пример отражает, как работает программа с операторами типа SELECT и с типами запросов, отличных от SELECT, а также с ошибочными операторами:

```
% ./client4
query> USE sampdb
0 rows affected
query> SELECT DATABASE(), USER()
+------------+--------------------+
| DATABASE() | USER()             |
+------------+--------------------+
| sampdb     | sampadm@localhost  |
+------------+--------------------+
1 rows returned
query> SELECT COUNT(*) FROM president
+----------+
| COUNT(*) |
+----------+
|       42 |
+----------+
1 rows returned
query> SELECT last_name, first_name FROM president ORDER BY last_name
LIMIT 3
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Adams     | John        |
```

```
| Adams      | John Quincy |
| Arthur     | Chester A.  |
+------------+-------------+
3 rows returned
query> CREATE TABLE t (i INT)
0 rows affected
query> SELECT j FROM t
Невозможно выполнить запрос
Error 1054 (Unknown column 'j' in 'field list')
query> USE mysql
Невозможно выполнить запрос
Error 1044 (Access denied for user: 'sampadm@localhost' to database
'mysql')
```

Создание клиентов с поддержкой протокола SSL

СУБД MySQL версии 4 снабжена поддержкой протокола SSL, который можно использовать для подключения программ к серверу с использованием защищенных соединений. Для того чтобы показать, как это сделать, в данном разделе показан процесс внесения изменений в программу `client4`. В результате будет получен клиент `sslclient`, который по внешним проявлениям аналогичен клиенту `client4`, но при этом позволяет устанавливать защищенные соединения. Для того чтобы программа `sslclient` работала соответствующим образом, СУБД MySQL должна поддерживать протокол SSL, который запущен с соответствующими параметрами, идентифицирующими сертификаты и ключевые файлы. При этом необходимо иметь сертификат и ключевые файлы на стороне клиента. Подробнее об этом можно прочитать в разделе "Установка безопасных соединений" в главе 12, "Безопасность". Кроме того, требуется версия 4.0.5 или более новая. Программы поддержки протокола SSL и обработки параметров для ранних выпусков 4.0.x ведут себя не совсем так, как описано здесь.

Дистрибутив базы данных `sampdb` содержит исходный файл, `sslclient.c`, из которого можно создать программу `sslclient`. Вся процедура, описывающая процесс преобразования программы `client4.c` в программу `sslclient.c`, может быть сформулирована следующим образом.

1. Скопировать `client4.c` в `sslclient.c`. Все остальные действия имеют отношение только к `sslclient.c`.
2. Для того чтобы компилятор мог определить наличие поддержки протокола SSL, в файле заголовка MySQL `my_config.h` объявляется символ `HAVE_OPENSSL`. Это значит, что при создании программы, работающей с протоколом SSL, можно воспользоваться следующей конструкцией, позволяющей игнорировать данный код, если нет возможности использовать протокол SSL:

   ```
   #ifdef HAVE_OPENSSL
   ...Здесь помещается код, работающий с протоколом SSL...
   #endif
   ```

   Заголовок `my_config.h` включен в другой заголовок `my_global.h`. Программа `sslclient.c` уже включает последний файл, следовательно, явно включать файл `my_config.h` нет смысла.

3. Внести изменения в массив my_opts, содержащий структуры с параметрической информацией, чтобы он включал параметры, имеющие отношение к протоколу SSL (- -ssl-ca, - -ssl-key и т.д.). Самый простой способ сделать это — включить содержимое файла sslopt-longopts.h в массив my_opts с директивой #include. После внесения этого изменения массив my_opts будет выглядеть следующим образом:

```
static struct my_option my_opts[] = /* структуры параметрической
информации */
{
    {"help", '?', "Отобразить эту подсказку и выйти",
    NULL, NULL, NULL,
    GET_NO_ARG, NO_ARG, 0, 0, 0, 0, 0, 0},
    {"host", 'h', "Узел подключения",
    (gptr *) &opt_host_name, NULL, NULL,
    GET_STR_ALLOC, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
    {"password", 'p', "Пароль",
    (gptr *) &opt_password, NULL, NULL,
    GET_STR_ALLOC, OPT_ARG, 0, 0, 0, 0, 0, 0},
    {"port", 'P', "Номер порта",
    (gptr *) &opt_port_num, NULL, NULL,
    GET_UINT, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
    {"socket", 'S', "Путь к разъему",
    (gptr *) &opt_socket_name, NULL, NULL,
    GET_STR_ALLOC, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
    {"user", 'u', "Имя пользователя",
    (gptr *) &opt_user_name, NULL, NULL,
    GET_STR_ALLOC, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
#include <sslopt-longopts.h>
    { NULL, 0, NULL, NULL, NULL, NULL, GET_NO_ARG, NO_ARG, 0, 0,
0, 0, 0, 0
}
};
```

Файл заголовка sslopt-longopts.h является общим файлом заголовка MySQL, и его содержимое выглядит следующим образом (немного видоизменено):

```
#ifdef HAVE_OPENSSL
    {"ssl", OPT_SSL_SSL,
    "Активизировать протокол SSL для соединения. Отключение - -
skip-ssl",
    (gptr*) &opt_use_ssl, NULL, 0,
    GET_BOOL, NO_ARG, 0, 0, 0, 0, 0, 0},
    {"ssl-key", OPT_SSL_KEY, "Ключ X509 в формате PEM (включение -
-ssl)",
    (gptr*) &opt_ssl_key, NULL, 0,
    GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
    {"ssl-cert", OPT_SSL_CERT, "Сертификат X509 в формате PEM
(включение - -ssl)",
    (gptr*) &opt_ssl_cert, NULL, 0,
    GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
    {"ssl-ca", OPT_SSL_CA,
    "файл CA в формате PEM (проверка документов OpenSSL, включение
- -ssl)",
    (gptr*) &opt_ssl_ca, NULL, 0,
    GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
    {"ssl-capath", OPT_SSL_CAPATH,
```

```
    "Каталог CA (проверка документов OpenSSL, включение - -ssl)",
    (gptr*) &opt_ssl_capath, NULL, 0,
    GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
    {"ssl-cipher", OPT_SSL_CIPHER, "Используемое шифрование SSL
(включение - -ssl)",
    (gptr*) &opt_ssl_cipher, NULL, 0,
    GET_STR, REQUIRED_ARG, 0, 0, 0, 0, 0, 0},
#endif /* HAVE_OPENSSL */
```

4. Параметрические структуры, объявленные в `sslopt-longopts.h`, ссылаются на значения OPT_SSL_SSL, OPT_SSL_KEY и т.д. Это применяется для коротких кодов параметров и должно задаваться в вашей программе, чего можно добиться, добавив следующие строки, предшествующие объявлению массива my_opts:

```
#ifdef HAVE_OPENSSL
enum options
{
    OPT_SSL_SSL=256,
    OPT_SSL_KEY,
    OPT_SSL_CERT,
    OPT_SSL_CA,
    OPT_SSL_CAPATH,
    OPT_SSL_CIPHER
};
#endif
```

   При создании своих собственных приложений, когда программа также объявляет какие-то свои значения и другие параметры, убедитесь в том, что значения символов OPT_SSL_XXX отличаются от этих значений.

5. Параметрические структуры, относящиеся к протоколу SSL в файле `sslopt-longopts.h`, ссылаются на набор переменных, предназначенных для хранения значений параметров. Для того чтобы это объявить, необходимо воспользоваться директивой `#include` для включения содержимого файла `sslopt-vars.h` в вашу программу до объявления массива my_opts. Содержимое файла `sslopt-vars.h` выглядит следующим образом:

```
#ifdef HAVE_OPENSSL
static my_bool opt_use_ssl = 0;
static char *opt_ssl_key = 0;
static char *opt_ssl_cert = 0;
static char *opt_ssl_ca = 0;
static char *opt_ssl_capath = 0;
static char *opt_ssl_cipher = 0;
#endif
```

6. Добавьте в программу `get_one_option()` строку, включающую файл `sslopt-case.h`:

```
my_bool
get_one_option (int optid, const struct my_option *opt, char
*argument)
{
    switch (optid)
    {
    case '?':
        my_print_help (my_opts); /* распечатать сообщение подсказки
*/
```

```
            exit (0);
        case 'p': /* пароль */
            if (!argument) /* значение не задано, поэтому оно
                              запрашивается позднее */
                ask_password = 1;
            else /* копировать пароль, удалить оригинал */
            {
                opt_password = strdup (argument);
                if (opt_password == NULL)
                {
                    print_error (NULL, "нет памяти для буфера пароля ");
                    exit (1);
                }
                while (*argument)
                    *argument++ = 'x';
            }
            break;
#include <sslopt-case.h>
        }
        return (0);
}
```

Файл `sslopt-case.h` включает ветви оператора switch(), который определяет, заданы ли параметры SSL, и в этом случае устанавливает переменную `opt_use_ssl`. Он имеет следующий вид:

```
#ifdef HAVE_OPENSSL
    case OPT_SSL_KEY:
    case OPT_SSL_CERT:
    case OPT_SSL_CA:
    case OPT_SSL_CAPATH:
    case OPT_SSL_CIPHER:
    /*
    Активизировать использование SSL, если используется хотя бы
    один ssl-параметр
    Отключить SSL можно потом, воспользовавшись --skip-ssl или
    --ssl=0
    */
    opt_use_ssl= 1;
    break;
#endif
```

В результате после обработки параметров появилась возможность определить, потребовал ли пользователь безопасного соединения. Для этого достаточно проверить значение `opt_use_ssl`.

Если придерживаться изложенной процедуры, обычные программы `load_defaults()` и `handle_options()` возьмут на себя заботу об анализе параметров, от которых зависит SSL-подключение, и установят их значения автоматически. Единственное, что остается сделать, — это передать параметрическую информацию протокола SSL на сервер, если по параметрам видно, что пользователю требуется SSL-соединение. Это можно сделать, вызвав функцию `mysql_ssl_set()` после вызова `mysql_init()` и до вызова `mysql_real_connect()`. Этот алгоритм имеет следующий вид:

```
/* инициализация дескриптора соединения */
conn = mysql_init (NULL);
if (conn == NULL)
```

```
{
   print_error (NULL, "сбой mysql_init()(вероятно, не хватает памяти) ");
   exit (1);
}
#ifdef HAVE_OPENSSL
   /* передать информацию SSL клиентской библиотеке */
if (opt_use_ssl)
   mysql_ssl_set (conn, opt_ssl_key, opt_ssl_cert, opt_ssl_ca,
                        opt_ssl_capath, opt_ssl_cipher);
#endif
/* подключение к серверу */
if (mysql_real_connect (conn, opt_host_name, opt_user_name, opt_password,
   opt_db_name, opt_port_num, opt_socket_name, opt_flags) == NULL)
{
   print_error (conn, "сбой mysql_real_connect()");
   mysql_close (conn);
   exit (1);
}
```

Заметим, что работа функции `mysql_ssl_set()` не проверяется на ошибки. Любая ошибка в информации, которая предоставляется этой функции, может привести к ошибке при вызове `mysql_real_connect()`.

После компиляции `sslclient.c` получим программу `sslclient`, а затем запустим ее. Полагая, что функция `mysql_real_connect()` отработала правильно, переходим к запросам. Запуск программы `sslclient` с соответствующими параметрами SSL, позволяет производить обмен данными с сервером через зашифрованные соединения. Для того чтобы определить, так ли это, сделаем следующий запрос:

```
SHOW STATUS LIKE 'Ssl_cipher'
```

Значение `Ssl_cipher` будет непустым, если используется шифр кодировки. (Для простоты понимания версия программы `sslclient`, имеющаяся в дистрибутиве `sampdb`, делает запрос и выводит полученный результат.)

Библиотека встроенного сервера

В MySQL 4 впервые появилась библиотека встроенного сервера, `libmysqld`, содержащая сервер в форме, которая может быть подключена (встроена) прямо в приложения. Это позволит создавать приложения, самостоятельно работающие с СУБД MySQL, в отличие от приложений, которые подключаются как клиент к отдельной серверной программе. Для того чтобы написать встроенное серверное приложение, необходимо выполнить два требования. Во-первых, должна быть установлена встроенная серверная библиотека:

- при создании из исходного текста активизируйте библиотеку с помощью параметра `--withembedded-server` при запуске `configure`;
- для двоичных дистрибутивов используйте дистрибутив с максимальными возможностями, так как другой дистрибутив не включает `libmysqld`;
- для RPM-инсталляций убедитесь в том, что вы устанавливаете встроенный RPM-сервер.

Во-вторых, вам потребуется включить в приложение небольшой код, обеспечивающий запуск и отключение самого сервера.

После того как вы убедитесь в том, что оба требования удовлетворены, необходимо откомпилировать и подключить библиотеку встроенного сервера (`-lmysqld`), а не обычную клиентскую библиотеку (`-lmysqlclient`). В действительности библиотека сервера разработана с учетом того, что при написании приложения, использующего ее, можно создать встроенную или клиент-серверную версию приложения, просто подключив соответствующую библиотеку. Это будет работать, так как обычная клиентская библиотека содержит функции, обеспечивающие интерфейс и имеющие ту же последовательность вызова, что и вызовы библиотеки встроенного сервера, являясь при этом заглушками (пустыми подпрограммами), которые ничего не делают.

Создание приложений со встроенным сервером

Процесс создания приложений, использующих встроенный сервер, не очень отличается от создания приложений, работающих в режиме клиент/сервер. Действительно, приступив к созданию программы, которая работает как приложение клиент/сервер, ее можно легко переделать в приложение, работающее со встроенным сервером. Например, для того чтобы переделать программу `client4` в программу `embapp`, которая работает со встроенным сервером, скопируем исходный текст из `client4.c` в `embapp.c`, а потом внесем в `embapp.c` следующие изменения.

1. Добавим заголовок `mysql_embed.h` в набор файлов заголовков MySQL:

```
#include <my_global.h>
#include <mysql.h>
#include <mysql_embed.h>
#include <my_getopt.h>
```

2. Встроенное приложение одновременно включает клиентскую и серверную стороны. Таким образом, оно обрабатывает одну группу параметров, предназначенную для клиента, и другую группу параметров, предназначенную для сервера. Например, приложение `embapp` может считывать группы `[client]` и `[embapp]` из параметрических файлов для клиентской части. Для того чтобы это установить, обратимся к определению массива `client_groups`, чтобы придать ему следующий вид:

```
static const char *client_groups[] =
{
    "client", "embapp", NULL
};
```

Параметры в этих группах должны обрабатываться функциями `load_defaults()` и `handle_options()` в обычном порядке. После этого идет объявление параметрических групп для использования их серверной стороной. В соответствии с соглашением этот перечень должен включать группы `[server]` и `[embedded]` и группу `[`*appname*`_SERVER]`, где *appname* является именем вашего приложения. Так, для программы `embapp` эта группа должна иметь имя `[embapp_SERVER]`, таким образом, список имен групп будет следующим:

```
static const char *server_groups[] =
{
    "server", "embedded", "embapp_SERVER", NULL
};
```

3. Функцию `mysql_server_init()` следует вызывать перед подключением к серверу. Это лучше всего сделать до вызова `mysql_init()`.
4. Функцию `mysql_server_end()` следует вызывать после завершения работы с сервером. Это лучше всего сделать после вызова `mysql_close()`.

После внесения всех этих изменений функция `main()` в `embapp.c` будет выглядеть следующим образом:

```
int
main (int argc, char *argv[])
{
int opt_err;
  my_init ();
  load_defaults ("my", client_groups, &argc, &argv);
  if ((opt_err = handle_options (&argc, &argv, my_opts, get_one_option)))
  exit (opt_err);
  /* по необходимости - запрос пароля */
  if (ask_password)
    opt_password = get_tty_password (NULL);
  /* получить имя базы данных, если оно присутствует в командной строке */
  if (argc > 0)
  {
    opt_db_name = argv[0];
    --argc; ++argv;
  }
  /* инициализация встроенного сервера */
  mysql_server_init (0, NULL, (char **) server_groups);
  /* инициализация дескриптора соединения */
  conn = mysql_init (NULL);
  if (conn == NULL)
  {
    print_error (NULL, "сбой mysql_init() (вероятно, не хватает памяти) ");
    exit (1);
  }
  /* подключение к серверу */
  if (mysql_real_connect (conn, opt_host_name, opt_user_name, opt_password,
    opt_db_name, opt_port_num, opt_socket_name, opt_flags) == NULL)
  {
    print_error (conn, "сбой mysql_real_connect()");
    mysql_close (conn);
    exit (1);
  }
  while (1)
  {
    char buf[10000];
    fprintf (stderr, "query> "); /* вывести подсказку */
    if (fgets (buf, sizeof (buf), stdin) == NULL) /* ввод запроса */
    break;
    if (strcmp (buf, "quit\n") == 0 || strcmp (buf, "\\q\n") == 0)
    break;
    process_query (conn, buf); /* выполнение запроса */
  }
  /* отключение от сервера */
  mysql_close (conn);
  /* выключение встроенного сервера */
  mysql_server_end ();
  exit (0);
}
```

Создание программы

Для получения исполняемого кода программы со встроенным сервером embapp необходимо воспользоваться библиотекой -lmysqld, а не -lmysqlclient. Здесь может понадобиться утилита mysql_config. Кроме того, что с ее помощью можно просмотреть флаги, необходимые для подключения к клиентской библиотеке, можно также посмотреть флаги, необходимые для встроенного сервера:

```
% mysql_config --libmysqld-libs
-L'/usr/local/mysql/lib/mysql' -lmysqld -lz -lm
```

Таким образом, для того, чтобы получить встроенную версию программы embapp, воспользуемся следующими командами:

```
% gcc -c `mysql_config --cflags` embapp.c
% gcc -o embapp embapp.o `mysql_config --libmysqld-libs`
```

После этого у вас имеется приложение со встроенным сервером, содержащее все, что требуется для доступа к базам данных MySQL. Однако при работе с embapp необходимо быть уверенным в том, что она не пытается использовать те же самые каталоги с данными, которые может при этом использовать автономный сервер, уже работающий на этой же самой машине. Кроме того, в ОС UNIX это приложение должно работать с привилегиями, позволяющими получить доступ к каталогу данных. Можно запустить программу embapp, зарегистрировавшись в качестве пользователя, который является владельцем каталога данных, или можно воспользоваться программой setuid, которая при запуске позволяет изменить идентификатор пользователя на идентификатор пользователя. Например, для того, чтобы программа embapp работала с привилегиями пользователя mysqladm, введите следующие команды, работая как пользователь root:

```
# chown mysqladm embapp
# chmod 4755 embapp
```

Но в случае, если вы передумали и решили создать невстроенную версию приложения, которая будет работать в контексте клиент/сервер, редактируйте связи с применением обычной клиентской библиотеки. Это можно сделать следующим образом:

```
% gcc -c `mysql_config --cflags` embapp.c
% gcc -o embapp embapp.o `mysql_config --libs`
```

Обычная клиентская библиотека содержит пустые версии функций mysql_server_init() и mysql_server_end(), поэтому делать дополнительно ничего не потребуется, ошибка при редактировании связей не возникнет.

Разное

Этот раздел посвящен некоторым проблемам, которые не рассматривались в данной главе.

- Вычисление результата на основании данных результирующего набора, после предварительной проверки метаданных, с целью определения пригодности этих данных для проведения вычислений.

- Что делать с данными, которые нельзя вставить в запросы.
- Обработка изображений.
- Получение информации о структуре таблиц.
- Ошибки программирования в СУБД MySQL, которые встречаются наиболее часто, и способы их предотвращения.

Выполнение вычислений на основе данных, полученных в результирующем наборе

До этого момента все наши усилия были сфокусированы на метаданных результирующего набора, прежде всего, с точки зрения распечатки данных. Но со всей определенностью можно утверждать, что наступят времена, когда с данными потребуется произвести операции другого рода. Например, на основании этих данных потребуется подсчитать статистическую информацию, воспользовавшись метаданными, чтобы: проверить, соответствуют ли данные нужным вам условиям. Условиям какого типа? Для начала можно проверить, что столбец, который планируется использовать в цифровых вычислениях, действительно содержит числа!

Ниже приведен листинг, представляющий простую функцию `summary_stats()`, которая на основании данных из результирующего набора и индекса столбцов вычисляет итоговую статистику значений, содержащихся в столбце. Функция также сообщает количество пустых значений, которые она определяет проверкой на пустые значения. При этих вычислениях к данным выдвигаются два требования, которые должны быть соблюдены. Функция `summary_stats()` проверяет их с помощью метаданных.

- Указанный столбец должен существовать (т.е. индекс столбца должен лежать в пределах диапазона номеров столбцов результирующего набора).
- Столбец должен содержать числовые данные.

Если эти условия не соблюдаются, `summary_stats()` распечатывает сообщение об ошибке. *В этом случае текст программы будет иметь такой вид:*

```
void
summary_stats (MYSQL_RES *res_set, unsigned int col_num)
{
MYSQL_FIELD    *field;
MYSQL_ROW      row;
unsigned int n, missing;
double val, sum, sum_squares, var;

  /* проверка соответствия условиям */
  if (col_num < 0 || col_num >= mysql_num_fields (res_set))
  {
    print_error (NULL, "Ошибочный номер столбца");
    return;
  }
  mysql_field_seek (res_set, col_num);
  field = mysql_fetch_field (res_set);
  if (!IS_NUM (field->type))
  {
    print_error (NULL, "Столбец не является числовым");
```

```
    return;
  }
  /* вычисление итоговой статистики */
  n = 0;
  missing = 0;
  sum = 0;
  sum_squares = 0;
  mysql_data_seek (res_set, 0);
  while ((row = mysql_fetch_row (res_set)) != NULL)
  {
    if (row[col_num] == NULL)
        missing++;
    else
    {
        n++;
        val = atof (row[col_num]); /* преобразование строки
                                      в число */
        sum += val;
        sum_squares += val * val;
    }
  }
  if (n == 0)
    printf ("Наблюдения отсутствуют \n") ;
  else
  {
    printf ("Количество наблюдений: %lu\n", n);
    printf ("Наблюдения отсутствуют: %lu\n", missing);
    printf ("Сумма: %g\n", sum);
    printf ("Среднее: %g\n", sum / n);
    printf ("Сумма квадратов: %g\n", sum_squares) ;
    var = ((n * sum_squares) - (sum * sum)) / (n * (n - 1))
    printf ("Дисперсия: %g\n", var);
    printf ("Квадратичное отклонение: %g\n", sqrt (var));
  }
}
```

Обратите внимание на вызов функции `mysql_data_seek()`, который предшествует циклу функции `mysql_fetch_row()`. Он здесь приведен для того, чтобы дать возможность несколько раз вызвать функцию `summary_stats()` для обработки одного и того же результирующего набора (в случае, если требуется вычислить статистику для нескольких столбцов). Перед каждым вызовом `summary_stats()` он делает "перемотку" на начало результирующего набора. (Это предполагает создание результирующего набора с помощью `mysql_store_result()`. В случае создания результирующего набора с помощью функции `mysql_use_result()` строки можно обработать по порядку и только один раз.)

Функция `summary_stats()` является относительно простой функцией, но на ее примере можно посмотреть, как написать программу для более сложных вычислений, таких как средняя квадратичная регрессия двух столбцов, или такой стандартной статистики, как `t-test`.

Кодирование неоднозначных данных в запросах

Вставленные в запрос значения данных, содержащие кавычки, значения "NULL" или обратную косую черту, могут вызвать неоднозначность при выполне-

нии запроса. Ниже изложен материал, в котором описываются природа этого явления и способы разрешения возникающих проблем.

Предположим, что нам требуется создать запрос SELECT, обрабатывающий содержимое строки, завершающейся значением "NULL". Обращение к этой строке осуществляется по имени:

```
char query[1024];
sprintf (query, "SELECT * FROM mytbl WHERE name='%s'", name_val);
```

Значение имени может быть чем-то вроде "O'Malley, Brian". Из-за того, что одинарная кавычка находится внутри такой строки, будет получен ошибочный запрос:

```
SELECT * FROM mytbl WHERE name=' O'Malley, Brian'
```

Одинарные кавычки необходимо обрабатывать каким-то специальным способом для того, чтобы сервер не интерпретировал его как конец имени. Одним из решений может быть дублирование одинарной кавычки в строке. Это соглашение ANSI SQL. СУБД MySQL распознает данные в соответствии с этим соглашением. Кроме того, стандарт допускает использование обратной косой черты перед одинарной кавычкой. Таким образом, этот запрос можно переписать двумя разными способами:

```
SELECT * FROM my_tbl WHERE name='O''Malley, Brian'
SELECT * FROM my_tbl WHERE name='O\'Malley, Brian'
```

Еще одна неоднозначная ситуация связана с использованием в запросе произвольных двоичных данных. Это происходит в приложениях, которые хранят в базах данных изображения. Ведь двоичные данные могут содержать любой символ, и поэтому их нельзя обрабатывать запросами в том виде, в каком они есть.

Для решения этой проблемы воспользуйтесь функцией `mysql_escape_string()`, которая кодирует специальные символы с помощью кавычек. Функция `mysql_escape_string()` рассматривает как специальные следующие символы: символ "NULL", одинарные и двойные кавычки, обратную косую черту, символ новой строки, символ возврата каретки и "<Cntl-Z>". (Последний символ характерен для ОС Windows, где им обозначается конец файла.)

В каком случае может использоваться `mysql_escape_string()`? Самым безопасным ответом будет "всегда". Однако, если есть уверенность в данных и все нормально (возможно, потому, что была сделана предварительная проверка), необходимость в кодировке данных отпадает. Например, нет необходимости в вызове `mysql_escape_string()` при работе со строками, которые, как известно, должны представлять телефонные номера, состоящие только из цифр и тире. В противном случае в этом существует определенная необходимость.

Функция `mysql_escape_string()` кодирует проблемные символы в символьные последовательности, состоящие из двух символов, начинающиеся с обратной косой черты. Так, байт "NULL" преобразуется в "\0", где символ "0" является обычным печатаемым символом "нуль" в кодировке ASCII. А символы обратной косой черты, одинарная и двойная кавычка становятся "\\", "\" и "\"".

Функция `mysql_escape_string()` вызывается следующим образом:

```
to_len = mysql_escape_string (to_str, from_str, from_len);
```

Функция `mysql_escape_string()` кодирует `from_str` и записывает результат в `to_str`. Кроме того, она добавляет символ "NULL" в конец строки. Это

очень удобно, так как полученную строку можно будет обрабатывать с помощью функций `strcpy()` и `strlen()`.

Переменная `from_str` содержит указатель на буфер типа `char`, в котором хранятся данные, предназначенные для кодировки. Эта строка может содержать все, что угодно, включая двоичные данные. Переменная `to_str` содержит указатель на существующий буфер типа `char`, куда и будет записываться закодированная строка. Ни в коем случае *не передавайте* не проинициализированный указатель или указатель, содержащий значение "NULL". Функция `mysql_escape_string()` сама не выделяет память. Длина буфера, адрес которого содержит `to_str`, должна быть по крайней мере `(from_len*2)+1` байт. (Вполне возможно, что каждый символ в `from_str` потребует кодировки двумя символами; плюс еще один байт для символа "NULL" в конце строки.)

Переменные `from_len` и `to_len` являются переменными типа `unsigned int`, `from_len` содержит длину данных в `from_str`. Эту длину необходимо знать потому, что строка `from_str` может содержать пустые байты, которые нельзя рассматривать как символ конца строки. Значение `to_len`, возвращаемое функцией `mysql_escape_string()`, является реальной длиной полученной закодированной строки, не считая символа "NULL" в конце строки.

Функция `mysql_escape_string()` возвращает закодированную результирующую строку в `to_str`, которую можно обрабатывать как строку с символом "NULL" в конце, так как любое пустое значение, которое присутствовало в строке `from_str`, кодируется в виде последовательности символов "`\0`".

Для того чтобы переписать программный код, конструирующий оператор `SELECT`, так, чтобы он мог безошибочно преобразовывать даже имена, содержащие кавычки, необходимо сделать что-то вроде следующего:

```
char query[1024], *p;
p = strcpy (query, "SELECT * FROM mytbl WHERE name='");
p += strlen (p);
p += mysql_escape_string (p, name, strlen (name));
p = strcpy (p, "'");
*p++ = '\'';
*p = '\0';
```

Да, этот код можно немного упростить, применив второй буфер:

```
char query[1024], buf[1024];
(void) mysql_real_escape_string (conn, buf, name, strlen (name));
sprintf (query, "SELECT * FROM mytbl WHERE name='%s'", buf);
```

Функция `mysql_real_escape_string()` отсутствовала до версии MySQL 3.23.14. Для ее замены можно воспользоваться функцией `mysql_escape_string()`:

```
to_len = mysql_escape_string (to_str, from_str, from_len);
```

Различие между ними заключается в том, что функция `mysql_real_escape_string()` при перекодировке использует текущую кодировку. Функция `mysql_escape_string()` использует кодировку, принятую по умолчанию. (Так происходит потому, что она не принимает параметр, содержащий информацию о дескрипторе соединения.) Для того чтобы написать программу, которая могла бы работать под управлением любой версии СУБД MySQL, необходимо добавить следующую конструкцию:

```
#if !defined(MYSQL_VERSION_ID) || (MYSQL_VERSION_ID<32314)
#define mysql_real_escape_string(conn,to_str,from_str,len) \
mysql_escape_string(to_str,from_str,len)
#endif
```

После этого можно писать программу с использованием функции `mysql_real_escape_string()`, но если таковая отсутствует, то конструкция `#define` вызовет переключение на работу с функцией `mysql_escape_string()`.

Обработка изображений

Основное предназначение функции `mysql_real_escape_string()` — загрузка изображений в таблицы. Этот раздел полностью посвящен этой теме. (Материал, изложенный здесь, можно также применить и для других типов двоичных данных.)

Предположим, что вы хотите получить изображения, хранящиеся в файле, и сохранить их с каким-то уникальным идентификатором в таблице. Для хранения двоичных данных лучше всего подходит тип BLOB, поэтому для этой цели можно воспользоваться таблицей с такой структурой:

```
CREATE TABLE picture
(
  pict_id INT NOT NULL PRIMARY KEY,
  pict_data BLOB
);
```

При заданных идентификационном номере и указателе на открытый файл, содержащем данные изображения, загрузку двоичных данных в таблицу можно выполнить, воспользовавшись функцией `load_image()`:

```
int
load_image (MYSQL *conn, int id, FILE *f)
{
char        query[1024*100], buf[1024*10], *p;
unsigned long from_len;
int         status;

  sprintf (query, "INSERT INTO picture (pict_id,pict_data) VALUES
(%d,'", id);
  p = query + strlen (query);
  while ((from_len = fread (buf, 1, sizeof (buf), f)) > 0)
  {
    /* не выходите за границы буфера запроса! */
    if (p + (2*from_len) + 3 > query + sizeof (query))
    {
        print_error (NULL, "изображение слишком велико");
        return (1);
    }
    p += mysql_real_escape_string (p, buf, from_len);
  }
  *p++ = '\'';
  *p++ = ')';
  status = mysql_real_query (conn, query, (unsigned long) (p - query));
  return (status);
}
```

Функция `load_image()` не располагает большим буфером (100 Кбайт), достаточным для хранения изображений, поэтому она предназначается, прежде всего,

для небольших изображений. В реальном приложении буфер можно разместить динамическим способом, основываясь на размере файла изображения.

Обработка изображений (как и любых других двоичных данных) не настолько трудна, как, скажем, загрузка их в базу данных, поскольку значения данных можно получить в переменной MYSQL_ROW, а длину этих данных можно получить с помощью функции `mysql_fetch_lengths()`. Предварительно необходимо убедиться в том, что это строки с известной длиной, а не ограниченные символом "NULL".

Получение информации из таблицы

С помощью одного из запросов (которые являются эквивалентными) СУБД MySQL позволяет получить информацию о структуре таблиц:

```
DESCRIBE tbl_name;
SHOW FIELDS FROM tbl_name;
DESCRIBE tbl_name;
EXPLAIN tbl_name;
```

Все эти операторы аналогичны оператору SELECT, так как они возвращают результирующий набор. Для того чтобы получить информацию о структуре таблицы, все, что надо сделать, — это обработать строки результирующего набора данных. Например, оператор `DESCRIBE president` возвратит следующую информацию:

```
mysql> DESCRIBE president;
+------------+-------------+------+-----+------------+-------+
| Field      | Type        | Null | Key | Default    | Extra |
+------------+-------------+------+-----+------------+-------+
| last_name  | varchar(15) |      |     |            |       |
| first_name | varchar(15) |      |     |            |       |
| suffix     | varchar(5)  | YES  |     | NULL       |       |
| city       | varchar(20) |      |     |            |       |
| state      | char(2)     |      |     |            |       |
| birth      | date        |      |     | 0000-00-00 |       |
| death      | date        | YES  |     | NULL       |       |
+------------+-------------+------+-----+------------+-------+
```

При выполнении такого же запроса с клиентского места будет получена аналогичная информация (но уже без рамки).

Для получения информации только об одном столбце добавьте имя столбца:

```
mysql> DESCRIBE president birth;
+------------+-------------+------+-----+------------+-------+
| Field      | Type        | Null | Key | Default    | Extra |
+------------+-------------+------+-----+------------+-------+
| birth      | date        |      |     | 0000-00-00 |       |
+------------+-------------+------+-----+------------+-------+
```

Наиболее часто встречающиеся ошибки при программировании клиентских программ

В этом разделе описаны некоторые наиболее часто встречающиеся ошибки, происходящие в процессе программирования на языке C с использованием интерфейса MySQL C API, и методы, помогающие их избежать. (Эти проблемы часто обсуждаются на Web-узле MySQL, и я тоже не буду их скрывать.)

Ошибка № 1 — работа с неинициализированными указателями дескрипторов соединения

Во всех примерах, которые продемонстрированы в этой главе, функция `mysql_init()` вызывалась с аргументом "NULL". Это позволяло ей создать и инициализировать структуру MYSQL и возвратить указатель на нее. Можно воспользоваться и другим подходом, при котором функции `mysql_init()` передается указатель на уже существующую структуру MySQL. В этом случае `mysql_init()` инициализирует эту структуру и возвращает указатель на нее, не создавая при этом новой структуры. Выбирая второй подход, вы должны отдавать себе отчет в том, что он может вызвать определенные скрытые проблемы. И здесь они будут затронуты.

Указатель, передаваемый функции `mysql_init()`, должен на что-то указывать. Рассмотрим следующий программный код:

```
main ()
{
MYSQL   *conn;
   mysql_init (conn);
   ...
}
```

Проблема заключается в том, что функция `mysql_init()` принимает указатель, но этот указатель не указывает на что-либо имеющее значение. Переменная `conn` является локальной переменной, при этом она не была инициализирована. Следовательно, в момент старта модуля `main()` она может указывать на что угодно. Это значит, что функция `mysql_init()` будет указывать на совершенно произвольное место в памяти. Хорошо, если указатель `conn` будет указывать на область памяти, находящуюся вне адресного пространства вашей программы, вследствие чего система тотчас ее остановит. Таким образом, сразу же станет ясно, что в программе содержится какая-то ошибка. Но если удача от вас отвернулась, указатель `conn` будет указывать на область данных, которая до поры до времени программой не используется, и вы просто ничего не заметите. В таком случае проблема возникнет позднее во время выполнения программы, поэтому диагностировать и устранить такую ошибку будет значительно труднее.

Вот аналогичная часть ошибочного программного кода:

```
MYSQL   * conn;

main ()
{
   mysql_init (conn);
   mysql_real_connect (conn, ...)
   mysql_query(conn, "SHOW DATABASES");
   ...
}
```

В этом случае переменная `conn` является глобальной. Она инициализируется со значением 0 (т.е. "NULL") до момента старта программы. Функция `mysql_init()` принимает аргумент со значением "NULL" и таким образом инициализирует новый дескриптор соединения. К сожалению, при этом переменная `conn` все еще имеет значение "NULL", поскольку ей так и не было присвоено никакого значения. А так как значение `conn` уже передано функции интерфейса MySQL C API, которая требует,

чтобы дескриптор соединения содержал значение, отличное от "NULL", программа прекратит выполнение из-за ошибки. Для того чтобы исправить обе программы, необходимо убедиться, что переменная conn содержит допустимое значение. Например, можно присвоить ей адрес уже существующей структуры MYSQL:

```
MYSQL conn_struct, *conn = &conn_struct;
...
mysql_init (conn);
```

Однако рекомендуемым (и самым простым!) решением будет просто задание явным образом функции `mysql_init()` значения "NULL", тем самым вынудив ее выделить структуру MYSQL и присвоить переменной conn возвращаемое значение:

```
MYSQL *conn;
...
conn = mysql_init (NULL);
```

В любом случае не забывайте проверять значение, возвращаемое функцией `mysql_init()`, чтобы убедиться, что оно не равно "NULL".

Ошибка № 2 — непроверенный результирующий набор

Помните о том, что необходимо проверять состояние вызываемых функций, от которых ожидается получение результирующего набора. Вот программа, которая этого не делает:

```
MYSQL_RES *res_set;
MYSQL_ROW row;

res_set = mysql_store_result (conn);
while ((row = mysql_fetch_row (res_set)) != NULL)
{
   /* обработка строки */
}
```

Очевидно, что в случае ошибочного завершения функции `mysql_store_result()` значение переменной `res_set` будет "NULL", и цикл `while` даже не будет выполнен. Таким образом, можно подытожить: обязательно проверяйте значения, возвращаемые функциями, которые возвращают результирующий набор, чтобы было с чем работать.

Аналогичный принцип можно применить к любой функции. Когда выполнение части программы, следующей за функцией, зависит от успешного завершения работы функции, проверяйте возвращаемое ею значение и предусмотрите соответствующее действие на случай возникновения сбоя в работе.

Ошибка № 3 — ошибка при выявлении столбцов, имеющих значение "NULL"

Не забывайте проверять значения столбцов в массиве MYSQL_ROW, возвращаемые функцией `mysql_fetch_row()`. Они не должны быть равны "NULL". Ниже приведен программный код, который сбоит на некоторых платформах, когда значение `row[i]` равно "NULL".

```
for (i = 0; i < mysql_num_fields (res_set); i++)
{
   if (i > 0)
```

```
    fputc ('\t', stdout);
    printf ("%s", row[i]);
}
fputc ('\n', stdout);
```

Самое неприятное в данной ситуации — то, что операторы `printf()` в некоторых версиях "игнорируют" эту ошибку и печатают "`(null)`" для указателей, имеющих значение "NULL". Это позволяет просто уйти от решения проблемы. Если передать эту программу другу, имеющему более "строгий" оператор `printf()`, программа даст сбой и ваш друг сделает необоснованный вывод, что вы плохой программист. Этот цикл нужно переписать следующим образом:

```
for (i = 0; i < mysql_num_fields (res_set); i++)
{
  if (i > 0)
    fputc ('\t', stdout);
    printf ("%s", row[i] != NULL ? row[i] : "NULL");
}
fputc ('\n', stdout);
```

Единственным случаем, когда не надо проверять значение столбца на значение "NULL", является случай, когда из атрибута этого столбца ясно, что он может принимать пустые значения.

Ошибка № 4 — передача бессмысленных результирующих значений

Функции из клиентской библиотеки, ожидающие буферы, предполагают, что они действительно существуют. Вот код, нарушающий этот принцип:

```
char *from_str = "some string";
char *to_str;
unsigned long len;
len = mysql_real_escape_string (conn, to_str, from_str, strlen
(from_str));
```

В чем здесь дело? Указатель `to_str` должен содержать адрес реально существующего буфера. В этом примере буфер не существует, и указатель содержит адрес какого-то произвольного места в памяти. Не передавайте неинициализированный указатель, как это было в примере с аргументом `to_str` функции `mysql_real_escape_string()`, конечно, если вы не хотите "разгуливать" где-то в произвольной области памяти.

7

Программный интерфейс Perl API

В этой главе...

Свойства сценариев языка Perl
Основы интерфейса базы данных для Perl
Подготовка интерфейса к работе
Работа с интерфейсом DBI в Web-приложениях

Эта глава посвящена описанию принципов работы интерфейса Perl DBI с СУБД MySQL. Здесь не обсуждаются философия и архитектура интерфейса с базами данных. Информацию об этих аспектах интерфейса можно почерпнуть в главе 5, "Введение в программирование в СУБД MySQL" (где сравниваются интерфейсы языков C и PHP).

Для более осознанного восприятия материала, изложенного в этой главе, достаточно иметь общее представление о языке написания сценариев Perl. Если его нет, эту главу можно пропустить и просто скопировать примеры, приведенные в ней. Кстати, в качестве примеров используется хорошо известная по прежним главам база данных `sampdb`. В качестве хорошо зарекомендовавшего себя руководства по языку Perl можно порекомендовать книгу *Освой самостоятельно Perl за 24 часа*, выпущенную "Издательским домом "Вильямс" (2000 г.).

Сейчас используется версия интерфейса 1.32, но большая часть материала этой главы опирается на более ранние версии 1.xx. Возможности, которые здесь описаны, но не представлены в ранних версиях, будут отмечены особо. Для интерфейса СУБД MySQL требуется версия языка Perl не меньше 5.005_03. Кроме клиентской библиотеки языка C СУБД MySQL и файлов заголовков, потребуется установка драйвера DBD::mysql. Если планируется написание сценариев интерфейса на базе Web-сервера

Apache, может потребоваться подключение модуля CGI.pm. Если вам понадобятся эти пакеты, обратитесь к приложению А, "Получение и инсталляция программного обеспечения", где описано, каким образом можно получить исходные тексты этих сценариев. Их можно найти на Web-узле в каталоге `perlapi`.

Методы и переменные интерфейса Perl описаны здесь только по мере необходимости. Более полный перечень методов и переменных интерфейса Perl дан в приложении Ж, "Программный интерфейс DBI языка написания сценариев Perl". Это приложение можно использовать также в качестве источника получения базовых данных о любом интерфейсе, который будет использоваться в дальнейшем. Получить документацию в режиме "on-line" можно с помощью следующих команд:

```
% perldoc DBI
% perldoc DBI::FAQ
% perldoc DBD::mysql
```

На уровне драйвера базы данных (DBD) драйвер для СУБД MySQL встроен в клиентскую библиотеку СУБД MySQL. Для получения более детальной информации об этой библиотеке обратитесь к главе 6, "Программный интерфейс MySQL C API".

Свойства сценариев языка Perl

Сценарии языка Perl представляют собой обычные текстовые файлы, поэтому их можно создать с помощью любого текстового редактора. Все приведенные в этой главе сценарии Perl написаны в соответствии с соглашением, принятым в ОС UNIX, поэтому для запуска сценария используется строка, начинающаяся с "`#!`", объявляющая имя пути программы, которая будет выполнять сценарий. Для этого сценария использовалась следующая строка:

```
#! /usr/bin/perl -w
```

Если путь на вашем компьютере отличается от приведенного здесь, внесите изменения в строку "`#!`", например `/usr/local/bin/perl5` или `/opt/bin/perl`. В противном случае сценарии Perl на вашем компьютере работать не будут.

После "`#!`" следует пробел, потому что некоторые системы интерпретируют последовательность символов "`#! /`" как символ комментария и рассматривают сценарий как сценарий оболочки.

Параметр `-w` сообщает Perl о том, что в случае обнаружения неоднозначной языковой конструкции или выполнения таких операций, как распечатка неинициализированных переменных, должно быть выдано соответствующее сообщение.

Это очень полезно, так как должно показывать, что текст программы требует исправлений. Запустить сценарий Perl `myscript.pl` можно следующим образом:

```
% perl -w myscript.pl
```

В ОС UNIX сценарий Perl должен быть исполнимым файлом. Таким образом, он может вызываться на выполнение простым вводом его имени. Для того чтобы это сделать, измените доступ к файлу:

```
% chmod +x myscript.pl
```

После этого сценарий можно запустить вводом его имени:

```
% ./myscript.pl
```

Этот стиль запуска программ и будет использоваться в сценариях, приведенных в этой главе. Ведущая последовательность символов "./" должна использоваться в том случае, когда сценарий располагается в текущем каталоге "." и ваша оболочка не имеет текущего каталога в путях, по которым она осуществляет поиск. В противном случае эту последовательность символов в командной строке можно не указывать:

```
% myscript.pl
```

При работе под управлением ОС Windows, построенных на технологии NT (NT, 2000 и XP), можно задать связь по имени файла между Perl и файлами с расширением .pl. Например, при установке ActiveState Perl программа установки позволяет вам установить связь между именами файлов с расширением .pl и Perl. В таком случае запускать сценарии Perl можно, указав их имя в командной строке:

```
C:\> myscript.pl
```

Основы интерфейса базы данных для Perl

В данном разделе содержится основная информация об интерфейсе. Ее будет вполне достаточно как для написания собственных сценариев, так и для понимания сценариев, написанных другими. Если вы уже знакомы с интерфейсом, переходите к разделу "Подготовка интерфейса к работе".

Типы данных интерфейса DBI

В некотором смысле использование интерфейса Perl DBI API аналогично работе с клиентской библиотекой языка C, описанной в главе 6, "Программный интерфейс MySQL C API". При работе с ней доступ к данным производится с помощью вызова функций и обращения к данным посредством указателей на структуры или массивы. При работе с интерфейсом Perl DBI API также вызываются функции и используются указатели на структуры, с той лишь разницей, что функции здесь называются *методами (methods)*, указатели — *ссылками (references)*, указательные переменные — *дескрипторами (handles)*, а структуры, на которые указывает дескриптор, — *объектами (objects)*.

Интерфейс базы данных использует несколько типов дескрипторов. Имена дескрипторов присваиваются в соответствии с соглашением, приведенным в табл. 7.1. Способ их применения будет проясняться по ходу изложения материала. Кроме того, существуют соглашения на некоторые переменные, не использующие дескрипторов (табл. 7.2). Не все из них будут упоминаться в этой главе, но знание этих соглашений очень может пригодиться для понимания сценариев, написанных другими программистами.

Таблица 7.1. Соглашение об именах переменных дескрипторов языка Perl DBI

Название	Значение
$dbh	Дескриптор объекта базы данных
$sth	Дескриптор объекта оператора (запроса)
$fh	Дескриптор открытия файла
$h	"Общий" дескриптор, смысл которого зависит от содержания

Таблица 7.2. Соглашение о значении имен переменных, не использующих дескрипторы

Название	Значение
$rc	Код, возвращаемый операциями ("true" или "false")
$rv	Код, возвращаемый операциями, возвращающими целое значение
$rows	Код, возвращаемый операциями, возвращающими значение счетчика строк
@ary	Массив, содержащий строки, возвращаемые запросом

Простой DBI-сценарий

Начнем с простого сценария dump_members.pl, иллюстрирующего несколько стандартных приемов программирования с использованием интерфейса DBI, таких как подключение к серверу СУБД MySQL и отключение от него, создание запросов и выборка данных. Этот сценарий делает выборку членов "Исторической Лиги" в формате колонок с табуляцией. Нас сейчас не интересует формат как таковой, в фокусе нашего внимания лежат возможности интерфейса. Сценарий dump_members.pl выглядит следующим образом:

```
#! /usr/bin/perl -w

# dump_members.pl - список членов "Исторической Лиги"

use strict;
use DBI;

my ($dsn) = "DBI:mysql:sampdb:cobra.snake.net"; # имя
                                                источника данных
my ($user_name) = "sampadm";   # имя пользователя
my ($password) = "secret"; # пароль

# подключение к базе данных
my $dbh = DBI->connect ($dsn, $user_name, $password, { RaiseError => 1,
PrintError => 0 });

# создать запрос
my $sth = $dbh->prepare ("SELECT last_name, first_name, suffix, email,"
. "street, city, state, zip, phone FROM member ORDER BY last_name");
$sth->execute();

# прочесть результаты, возвращаемые запросами, и очистить память
while (my @ary = $sth->fetchrow_array())
{
   print join ("\t", @ary), "\n";
}
$sth->finish();
$dbh->disconnect();
exit(0);
```

Для того чтобы проверить этот сценарий в действии, можно воспользоваться сценарием, который имеется в дистрибутиве sampdb, или создать его с помощью обычного текстового редактора, сделав затем исполнимым. Конечно, может понадобиться изменить по крайней мере один из параметров подключения (имя узла, имя базы данных, имя пользователя, пароль). Это справедливо и для других сценариев, приведенных в этой главе. Ниже, в разделе "Определение параметров соеди-

нения", будет показано, как можно получить параметры из файла, чтобы не вводить их в сценарий.

Теперь рассмотрим сценарий построчно. Первая строка содержит стандартный индикатор, определяющий "местоположение Perl":

```
#! /usr/bin/perl -w
```

Данная строка присутствует в каждом сценарии, представленном в этой главе, и больше ему внимание уделяться не будет.

Уместно будет включить в сценарий короткое описание его назначения, поэтому следующая строка содержит подсказку о том, чем "занимается" данный сценарий:

```
# dump_members.pl - список членов "Исторической Лиги"
```

Строки, начинающиеся с символов "#", считаются комментариями. Дополнение сценариев комментариями с разъяснениями работы сценариев является общепринятой практикой.

Далее следуют такие строки:

```
use strict;
use DBI;
```

Строка `use strict` сообщает интерпретатору Perl, что перед использованием переменных их требуется декларировать. Эту строку в сценарии можно опустить, но такая возможность очень полезна при диагностировании ошибок, так что я настоятельно рекомендую всегда ее включать в исходный текст. Например, если переменная `$my_var` была объявлена при включенном режиме `strict`, но потом было ошибочное обращение к переменной `$mv_var`, во время выполнения сценария будет получено следующее сообщение:

```
Global symbol "$mv_var" requires explicit package name at line n
Для глобального символа "$mv_var" требуется явное объявление имени в
строке n
```

Когда вы увидите это сообщение, возникнет мысль, что вы никогда не использовали переменную с подобным именем! А когда вы посмотрите в своем сценарии на строку с номером n, вы, конечно, увидите, в чем заключается проблема, и легко исправите ошибку. Не работая в режиме `strict`, интерпретатор Perl не сообщит об ошибке, допущенной в переменной `$mv_var`, а просто создаст новую переменную с таким именем и значением `undef` (неопределенный) и будет использовать ее, а вы очень скоро с удивлением обнаружите, что ваш сценарий не работает.

Строка `use DBI` сообщает интерпретатору Perl, что ему необходимы ресурсы модуля DBI. Без этой строки при попытке сделать в сценарии что-либо, связанное с интерфейсом, произойдет ошибка. При этом нет надобности указывать, какой модуль уровня DBD нужен. Интерфейс DBI инициирует нужный драйвер DBD в момент подключения к базе данных.

Работая в режиме `strict`, придется объявить все переменные, которые используются сценарием с применением ключевого слова `my`. Следующая часть сценария задает переменные соединения, а затем использует их для подключения к базе данных:

```
my ($dsn) = "DBI:mysql:sampdb:localhost"; # наименование источника
                                          # данных
my ($user_name) = "sampadm";              # имя пользователя
my ($password) = "secret";                # пароль
```

```
# подключение к базе данных
my $dbh = DBI->connect ($dsn, $user_name, $password, { RaiseError => 1,
PrintError => 0 });
```

Вызов функции `connect()` осуществляется как `DBI->connect()` потому, что этот метод присущ классу DBI. Вам совсем не обязательно знать, что это значит, это просто такой объектно-ориентированный жаргон, который призван сбить вас с толку. (Если вы хотите знать, то это значит, что метод `connect()` "принадлежит" интерфейсу DBI.) Метод `connect()` принимает несколько аргументов.

- **Источник данных.** (Обычно это называется "data source name" (имя источника данных), или DSN.) Форматы источника данных определены требованиями к используемому модулю DBD. Форматы драйвера СУБД MySQL могут включать что-нибудь подобное:

  `"DBI:mysql:`*db_name*`"`
  `"DBI:mysql:`*db_name*`:`*host_name*`"`

  Представление `DBI` не имеет значения, но `mysql` должен быть задан в нижнем регистре. Значение *db_name* представляет собой имя используемой базы данных, *host_name* отражает имя узла, на котором работает сервер. Если имя узла не задано, по умолчанию принимается значение `localhost`. (В действительности существуют и другие допустимые исходные форматы данных, которые будут изложены ниже, в разделе "Определение параметров соединения".)
- **Ваше имя пользователя и пароль для учетной записи СУБД MySQL.**
- **Дополнительный аргумент, отражающий дополнительные атрибуты соединения.** Он управляет поведением DBI при обработке ошибок интерфейсом, а загадочная конструкция, указанная здесь, активизирует атрибут `RaiseError` и отключает атрибут `PrintError`. Это вынуждает интерфейс DBI проверять возникающие ошибки, которые связаны с базами данных, выдавать диагностические сообщения и выходить при обнаружении таковых. (Вот почему полностью отсутствует программный код проверок на ошибки в сценарии `dump_members.pl`; интерфейс DBI берет все это на себя.) В разделе "Обработка ошибок" изложены и другие способы реакции на ошибки.

В случае успешного завершения работы метода `connect()` переменной `$dbh` возвращается дескриптор базы данных. Если работа метода `connect()` завершается ошибкой, он возвращает значение `undef`. Однако с атрибутом `RaiseError` в сценарии метод `connect()` в случае ошибки не возвращает управление, а интерфейс DBI выдаст сообщение об ошибке и завершит выполнение. (Это справедливо и для других методов DBI; здесь будет показано, какую информацию они возвращают для того, чтобы идентифицировать ошибку, но они ничего не возвращают, когда аргумент `RaiseError` активизирован.)

После подключения к базе данных сценарий `dump_members.pl` генерирует запрос `SELECT` для выборки списка членов "Исторической Лиги", а затем выполняет цикл по обработке строк, возвращаемых запросом. Эти строки и составляют результирующий набор.

Для выполнения запроса сначала его нужно создать, а затем выполнить:

```
# создать запрос
my $sth = $dbh->prepare ("SELECT last_name, first_name,
                          suffix, email," . "street, city, state, zip,
                          phone FROM member ORDER BY last_name");
$sth->execute();
```

Метод `prepare()` вызывается с использованием дескриптора базы данных. Он передает оператор SQL драйверу для предварительной обработки перед выполнением. Одни драйверы действительно производят какие-то операции на этой фазе выполнения. Другие просто запоминают эту информацию до момента вызова метода `execute()` для выполнения оператора. Значение, возвращаемое методом `prepare()`, является дескриптором оператора, и присваивается здесь `$sth`. Дескриптор оператора используется при дальнейшей обработке, имеющей отношение к данному оператору.

Прошу обратить внимание, что запрос не завершается символом "точка с запятой". В процессе длительного использования программы `mysql` у вас, без сомнения, выработалась привычка завершать операторы языка SQL символом ";". Однако эту привычку придется забыть, так как интерфейс DBI расценивает такой синтаксис как ошибочный. Это справедливо и относительно добавления к запросам последовательности "\g". Эти ограничители операторов соответствуют соглашениям `mysql`, но не используются при выдаче запросов в DBI-сценариях. Завершение строки запроса неявным образом завершает запрос, и указания явного ограничителя при этом не требуется.

При запуске метода без передачи ему каких-либо параметров можно опустить скобки. Вот два абсолютно аналогичных вызова:

```
$sth->execute();
$sth->execute;
```

Я предпочитаю указывать скобки потому, что это позволяет отличать метод от переменной. Вы вольны поступать по-своему.

После вызова `execute()` обработаем полученный список. В сценарии `dump_members.pl` цикл выборки строк просто распечатывает содержимое каждой строки:

```
# прочесть результаты, возвращаемые запросами, и очистить память
while (my @ary = $sth->fetchrow_array())
{
  print join ("\t", @ary), "\n";
}
$sth->finish();
```

Метод `fetchrow_array()` возвращает массив, содержащий значения столбцов текущей строки и пустую строку, когда таких строк больше нет. Таким образом, цикл последовательно выбирает строки, возвращаемые оператором `SELECT`, и распечатывает каждый, разделяя значения столбцов символами табуляции. Пустые значения из базы данных возвращаются в сценарий Perl как пустые строки, а не значение "NULL". Значения столбцов `undef` также имеют другое влияние при запуске сценария — они завершаются при получении от Perl предупреждающих сообщений следующего вида:

```
Use of uninitialized value in join at dump_members.pl line n.
Использование неинициализированного значения в строке n в
dump_members.pl.
```

Эти сообщения запускаются с помощью параметра `-w`, имеющегося в первой строке сценария. Убрав этот параметр и запустив сценарий повторно, вы увидите, что это сообщение исчезнет. Однако параметр `-w` очень может пригодиться при отладке сценария (например, для распечатки неинициализированных переменных!), таким образом, наилучшим методом ограничения сообщений является определение и обработка значений `undef`. Некоторые подобные приемы описаны ниже, в разделе "Обработка запросов, возвращающих результирующий набор данных".

Обратите внимание, что символы перевода на новую строку в операторе `print` ("`\t`" и "`\n`") заключены в кавычки. Управляющие последовательности в Perl интерпретируются только тогда, когда они заключены в кавычки. При наличии одинарных кавычек выводимые строки будут заполнены литеральными образцами строк "`\t`" и "`\n`".

После завершения цикла выборки вызов метода `finish()` "дает знать" интерфейсу DBI, что дескриптор оператора больше не нужен и все временные ресурсы, отданные под него, могут быть освобождены. В этом сценарии вызов метода `finish()` сделан только в качестве примера. В этом случае его вызывать совсем не обязательно, так как вызов функции выборки строк сделает это автоматически при обнаружении конца результирующего набора. Метод `finish()` предпочтительнее в ситуациях, когда производится выборка только части результирующего набора и не будет достигнут его конец (например, если по какой-либо причине производится выборка только первой строки). С этого момента метод `finish()` будет использоваться только по мере необходимости.

После того как список членов будет распечатан, можно отключиться от сервера и завершить работу сценария:

```
$dbh->disconnect();
exit(0);
```

Сценарий `dump_members.pl` демонстрирует несколько самых характерных концепций для большинства программ, работающих с интерфейсом DBI. Этого вполне достаточно для того, чтобы писать свои программы, работающие с интерфейсом DBI. Например, чтобы распечатать содержимое другой таблицы, все, что необходимо сделать, — это изменить текст оператора `SELECT`, который передается методу `prepare()`. Конечно, если есть насущная необходимость посмотреть, каким образом работает этот метод, можно перейти к разделу "Подготовка интерфейса к работе" "Putting DBI to Work". В нем показано создание списка членов для программы ежегодного банкета "Исторической Лиги" и списка членов лиги. Однако интерфейс DBI обладает и многими другими весьма полезными возможностями, детальное изложение которых вы найдете в последующих разделах. Там же рассказывается, каким образом в Perl можно выполнять более сложные операции, чем простой оператор `SELECT`.

Обработка ошибок

Сценарий `dump_members.pl` активизирует атрибут обработки ошибок `RaiseError` при инициализации метода `connect()`. Таким образом, возникно-

вение ошибки автоматически прерывает выполнение сценария и выдает диагностическое сообщение. Ошибки можно обрабатывать и другим способом. Например, обработку ошибочных ситуаций можно запрограммировать, не полагаясь на возможности интерфейса DBI.

Для того чтобы подробнее ознакомиться с поведением интерфейса DBI при обработке ошибочных ситуаций, обратим внимание на последний аргумент метода `connect()`. Там задается значение двух атрибутов, `RaiseError` и `PrintError`.

- Атрибут `RaiseError` активизирован (задано ненулевое значение). В случае возникновения ошибки интерфейс DBI для вывода сообщения и завершения выполнения сценария вызывает метод `die()`.
- При активизированном атрибуте `PrintError` в случае возникновения ошибки интерфейс DBI вызывает метод `warn()`, но сценарий продолжает выполнение.

По умолчанию параметр `RaiseError` заблокирован, а параметр `PrintError` активизирован. В этом случае, если при выполнении метода `connect()` возникнет ошибка, интерфейс DBI выдаст сообщение, но выполнение сценария не прерывается. Таким образом, при обработке ошибок по умолчанию, которую можно выполнить, не передавая методу аргументов, ошибки можно обрабатывать следующим образом:

```
my $dbh = DBI->connect ($dsn, $user_name, $password)
   or exit (1);
```

При возникновении ошибки метод `connect()` возвращает значение `undef` и инициирует вызов метода `exit()`. В этом случае не нужно специально печатать сообщение об ошибке, поскольку интерфейс DBI уже один раз выдал такое сообщение.

При явном определении атрибутов проверки ошибок вызов метода `connect()` выглядит следующим образом:

```
my $dbh = DBI->connect ($dsn, $user_name, $password,
    { RaiseError => 0, PrintError => 1})
    or exit (1);
```

Это потребует больших затрат времени, но даст больше информации пользователю о поведении программного кода.

Если есть намерение писать собственноручно программный код, обрабатывающий ошибочные ситуации, необходимо заблокировать параметры `RaiseError` и `PrintError`:

```
my $dbh = DBI->connect ($dsn, $user_name, $password,
{ RaiseError => 0, PrintError => 0})
   or die "Невозможно подключиться к серверу:
           $DBI::err ($DBI::errstr)\n";
```

Переменные `$DBI::err` и `$DBI::errstr`, которые используются в только что показанном вызове метода `die()`, нужны для создания диагностических сообщений. Они содержат коды и тексты ошибок СУБД MySQL, совсем как функции интерфейса C API `mysql_errno()` и `mysql_error()`. При отсутствии ошибки `$DBI::err` будет равна `0` или `undef`, а `$DBI::errstr` будет пустой строкой или иметь значение `undef`. (Другими словами, обе переменные будут ложными.)

Если вы хотите возложить обработку всех ошибочных ситуаций на интерфейс DBI, чтобы не проверять их самому, достаточно активизировать параметр `RaiseError` и заблокировать параметр `PrintError`:

```
my $dbh = DBI->connect ($dsn, $user_name, $password,
   { RaiseError => 1, PrintError => 0 });
```

Этот подход значительно проще и применяется фактически во всех представленных в этой главе сценариях. Причина отключения атрибута `PrintError` и включения атрибута `RaiseError` заключается в необходимости предотвратить дублирование диагностических сообщений. (При активизации обоих атрибутов дескрипторы DBI при определенных обстоятельствах могут вызываться вместе.)

При необходимости проводить некие операции очистки памяти при выходе по ошибке актуализация параметра `RaiseError` не совсем подходит, но и в этом случае остается возможность сделать все необходимое. Для этого достаточно переопределить дескриптор `$SIG{__DIE__}`. Кроме того, желание избежать разблокировки параметра `RaiseError` может диктоваться тем, что интерфейс DBI распечатывает в своих диагностических сообщениях только техническую информацию:

```
disconnect(DBI::db=HASH(0x197aae4)) invalidates 1 active statement.
Either destroy statement handles or call finish on them before
disconnecting.
Отключение(DBI::db=HASH(0x197aae4)) делает недействительным 1 активный
оператор.
Удалите дескриптор оператора или завершите его работу перед отключением.
```

Эта информация понятна программисту, но ее вряд ли следует показывать обычному пользователю. В этом случае лучше сделать проверку на ошибки самому и сформировать более понятные сообщения для пользователей данного сценария. Кроме того, в качестве альтернативного метода здесь можно рассмотреть возможность переопределения дескриптора `$SIG{__DIE__}`. Это может упростить программирование обработки ошибочных ситуаций, так как позволит воспользоваться возможностями параметра `RaiseError`, заменив при этом обычные сообщения интерфейса DBI своими собственными сообщениями. Для того чтобы создать собственный дескриптор `__DIE__`, до вызова DBI поступите так:

```
$SIG{_DIE_} = sub { die "Извините, обнаружена ошибка \n"; };
```

Подпрограмму можно объявить обычным образом и установить значение дескриптора с помощью обращения к подпрограмме:

```
sub die_handler
{
   die "Извините, обнаружена ошибка \n";
}
$SIG{_DIE_} = \&die_handler;
```

В качестве альтернативы передаче атрибутов обработки ошибок методу `connect()` литеральным образом можно определить их с помощью хэш-памяти и передавать уже только ссылку на хэш-память. Есть мнение, что такие установки атрибутов делают сценарии более читаемыми и редактируемыми, но функционально оба подхода идентичны. Вот пример использования хэша атрибутов:

```
my %attr =
(
   PrintError => 0,
```

```
  RaiseError => 0
);
my $dbh = DBI->connect ($dsn, $user_name, $password, \%attr)
  or die "Невозможно подключиться к серверу: $DBI::err
($DBI::errstr)\n";
```

Сценарий dump_members2.pl демонстрирует возможность создания сценария, в котором запрограммирована обработка ошибок, и печати своих собственных сообщений. Сценарий dump_members2.pl обрабатывает тот же запрос, что и сценарий dump_members.pl, но явным образом блокирует работу параметров PrintError и RaiseError, а затем блокирует результат каждого обращения к интерфейсу DBI. При возникновении ошибок сценарий перед выходом вызывает подпрограмму bail_out(), которая выводит сообщение и содержимое переменных $DBI::err и $DBI::.

```
#! /usr/bin/perl -w

# dump_members2.pl - список членов "Исторической Лиги"

use strict;
use DBI;

my ($dsn) = "DBI:mysql:sampdb:cobra.snake.net"; # наименование источника
                                                данных
my ($user_name) = "sampadm";   # имя пользователя
my ($password) = "secret"; # пароль
my ($dbh, $sth);           # дескрипторы базы данных и операторов
my (%attr) =               # атрибуты обработки ошибок
(
  PrintError => 0,
  RaiseError => 0
)

# подключение к базе данных
my $dbh = DBI->connect ($dsn, $user_name, $password, \%attr)
  or bail_out ("Невозможно подключиться к базе данных");

# создать запрос
my $sth = $dbh->prepare ("SELECT last_name, first_name, suffix,
                         email,"
  . "street, city, state, zip, phone FROM member ORDER BY
      last_name")
  or bail_out ("Невозможно создать запрос");
$sth->execute ()
  or bail_out ("Невозможно выполнить запрос");

# прочесть результаты запросов и очистить
while (my @ary = $sth->fetchrow_array ())
{
  print join ("\t", @ary), "\n";
}
$DBI::err
  or bail_out ("Во время выборки произошла ошибка");

# очистить память
$dbh->disconnect ()
  or bail_out ("Невозможно отключиться от базы данных");
```

```
exit (0);

# подпрограмма bail_out()- распечатать код и сообщение об ошибке, а затем
завершить выполнение

sub bail_out
{
my $message = shift;

   die "$message\nError $DBI::err ($DBI::errstr)\n";

}
```

Подпрограмма `bail_out()` аналогична функции `print_error()`, которая использовалась нами в главе 6, "Программный интерфейс MySQL C API", за одним исключением: подпрограмма `bail_out()` завершает выполнение сценария, а не возвращает управление вызывающей программе. Подпрограмма `bail_out()` освобождает от необходимости работы с переменными `$DBI::err` и `$DBI::errstr` всякий раз, когда есть необходимость напечатать диагностическое сообщение. Обработка диагностических сообщений в одной подпрограмме позволяет при необходимости менять формат диагностического сообщения во всем сценарии простым изменением его в одном месте подпрограммы.

Сценарий `dump_members2.pl` делает проверку сразу же после цикла выборки строк, чего нет в сценарии `dump_members.pl`. Поскольку сценарий `dump_members2.pl` не завершает работу при возникновении ошибки в `fetchrow_array()` автоматически, он вправе определить, завершил ли свою работу цикл из-за того, что результирующий набор был выбран полностью (нормальное завершение), или из-за возникновения ошибки. Работа цикла завершается в любом случае, но при возникновении ошибки вывод будет сделан в усеченном виде. В случае отсутствия проверки на ошибку пользователь не будет знать того, что что-то не так! Поэтому, если сценарий программируется таким образом, что проверка ошибочных ситуаций производится вручную, обязательно проверяйте результаты выборки.

Обработка запросов, не возвращающих результирующего набора данных

Операторы типа `DELETE`, `INSERT`, `REPLACE` и `UPDATE`, не возвращающие строк, по сравнению с операторами класса `SELECT`, `DESCRIBE`, `EXPLAIN` и `SHOW`, возвращающими строки, обрабатывать относительно просто. Для обработки операторов, не принадлежащих классу `SELECT`, их следует передать методу `do()` с использованием дескриптора. Метод `do()` подготавливает и выполняет запрос за один прием. Например, для того, чтобы добавить запись о новом члене по имени Marcia Brown с датой истечения срока членства 3 июня 2005, необходимо сделать следующее:

```
$rows = $dbh->do ("INSERT member (last_name,first_name,expiration)"
    .  " VALUES('Brown','Marcia','2005-6-3')");
```

Метод `do()` возвращает количество обработанных строк, или `undef`, в случае возникновения ошибочной ситуации, или -1, если количество возвращаемых

трок неизвестно. Ошибка может произойти по нескольким причинам. Например, запрос сам может содержать ошибку, или у вас нет разрешения на оступ к таблице.) В случае возврата значения, отличного от undef, обратите нимание на случай, когда нет обработанных строк. Когда такое происходит, ме-од do() возвращает не значение 0, а строку "0E0" (форма записи 0, принятая для зыка написания сценариев Perl). Значение "0E0" в цифровом контексте равно 0, в условных операциях дает значение "истина", которое может быть отлично от начения undef. Если бы метод do() возвращал 0, было бы значительно сложнее азличать ошибочную ситуацию (undef) и ситуацию "строки не обработаны". ыполнить проверку на ошибку можно с помощью одного из тестов:

```
f (!defined ($rows))

  print "Произошла ошибка \n";

f (!$rows)

  print "Произошла ошибка \n";
```

В цифровом контексте "0E0" расценивается как 0. Следующий программный од правильно печатает число строк, дающих значение переменной $rows, отличное от undef:

```
f (!$rows)

  print "Произошла ошибка \n";

lse

  $rows += 0; # преобразовать в число, если значение равно "0E0"
  print "Обработано $rows строк \n";
```

Вывести значение $rows также можно с помощью формата "%d" оператора rintf() для преобразования в числовой формат:

```
f (!$rows)

  print "Произошла ошибка \n";

lse

  printf "Обработано %d строк \n", $rows;
```

Метод do() аналогичен последовательности методов prepare() и xecute(). Предшествующий оператор INSERT можно запрограммировать, не ызывая метода do():

```
sth = $dbh->prepare ("INSERT member
last_name,first_name,expiration)"
   . " VALUES('Brown','Marcia','2005-6-3')");
rows = $sth->execute ();
```

Обработка запросов, возвращающих результирующий набор данных

Этот раздел посвящен возможностям выполнения циклов выборки строк запросов типа SELECT (или других подобных запросов SELECT, возвращающих строки, такие как DESCRIBE, EXPLAIN и SHOW). Кроме того, мы коснемся темы определения количества результирующих строк, обработки результирующих наборов строк, для которых цикл не нужен. Вы также узнаете, как произвести единовременную выборку.

Как писать циклы выборки строк

Сценарий dump_members.pl производит выборку данных с помощью стандартной последовательности методов интерфейса DBI: метода prepare(), осуществляющего предварительную обработку запроса драйвером; метода execute(), запускающего выполнение запроса, и метода fetchrow_array(), реализующего выборку строк результирующего запроса.

Методы prepare() и execute() являются достаточно традиционными частями процедуры обработки запросов, возвращающих строки. Однако для операции выборки строк метод fetchrow_array() действительно является единственно возможным (табл. 7.3).

Таблица 7.3. Методы выборки строк для интерфейса DBI

Название метода	Возвращаемое значение
fetchrow_array()	Массив со значениями строк
fetchrow_arrayref()	Ссылка на массив со значениями строк
fetch()	Аналогично методу fetchrow_arrayref()
fetchrow_hashref()	Ссылка на хэш со значениями строк, индексированная по именам столбцов

Следующие примеры демонстрируют работу методов выборки строк. В этих примерах в цикле просматривается результирующий набор и построчно распечатываются значения столбцов, разделенные запятыми. В некоторых случаях существуют и более эффективные способы создания программ, но эти примеры написаны таким образом, чтобы лучше продемонстрировать синтаксис, предоставляющий доступ к значениям в каждом отдельном столбце. Метод fetchrow_array() работает следующим образом:

```
while (my @ary = $sth->fetchrow_array ())
{
  my $delim = "";
  for (my $i = 0; $i < @ary; $i++)
  {
    $ary[$i] = "" if !defined ($ary[$i]); # значение NULL?
    print $delim . $ary[$i];
    $delim = ",";
  }
  print "\n";
}
```

Каждый вызов метода fetchrow_array() возвращает массив, содержащий значения строк, или пустой массив, когда строк больше нет. Внутренний цикл

тестирует значение каждого столбца, чтобы просмотреть, определен он или нет, и устанавливает пустую строку, если столбец не определен. Таким образом, пустые значения (которые отражаются интерфейсом DBI как undef) преобразуются в пустые строки.

С первого взгляда это может показаться абсолютно ненужной операцией, так как Perl ничего не напечатает как для значений undef, так и для пустой строки. Цель этого теста заключается в том, чтобы показать, что, работая с параметром -w, сценарий при попытке распечатать значение undef, выведет диагностическое сообщение Use of uninitialized value (Использование неинициализированного значения). Аналогичную конструкцию вы будете встречать в этой главе повсеместно. Если необходимо печатать что-то другое для значений undef, например строку "NULL", достаточно немного видоизменить данный тест:

```
while (my @ary = $sth->fetchrow_array ())
{
  my $delim = "";
  for (my $i = 0; $i < @ary; $i++)
  {
    $ary[$i] = "NULL" if !defined ($ary[$i]); # значение NULL?
    print $delim . $ary[$i];
    $delim = ","
  }
  print "\n";
}
```

Работая с массивом значений, можно немного упростить текст программы, воспользовавшись методом map для одновременного преобразования всех значений undef:

```
while (my @ary = $sth->fetchrow_array ())
{
  @ary = map { defined ($_) ? $_ : "NULL" } @ary;
  print join (",", @ary) . "\n";
}
```

Метод map обрабатывает каждый элемент массива, пользуясь выражением внутри фигурных скобок, и возвращает массив результирующих значений.

Существует альтернативный метод выборки значений столбцов не в массивы, а в скалярные переменные. Такая необходимость может возникнуть в работе с переменными, имена которых будут понятнее, чем, скажем, имена массивов $ary[0], $ary[1] и т.д. Предположим, что надо произвести выборку имени и адреса электронной почты. С помощью метода fetchrow_array() производится выборка целых строк набора данных:

```
my $sth = $dbh->prepare ("SELECT last_name, first_name, suffix, email"
  .   " FROM member ORDER BY last_name");
$sth->execute ();
while (my ($last_name, $first_name, $suffix, $email)
  = $sth->fetchrow_array ())
{
  # произвести определенные действия над переменными
}
```

Конечно, при такой работе со списком переменных у программиста всегда есть уверенность в том, что столбцы выбираются в нужном порядке. Интерфейс DBI не

"знает" порядка следования столбцов в операторе SELECT, таким образом, задачей программиста является соблюдение правильного порядка следования столбцов. Кроме того, можно задать автоматическое присвоение значений столбцов отдельным переменным, пользуясь так называемым методом привязки параметров. Этот метод будет изложен позднее, в разделе "Маркеры и привязка параметров".

При выборке одного значения в переменную будьте внимательны при выполнении операции присвоения. Если написать начало цикла следующим образом, он будет работать правильно:

```
while (my ($val) = $sth->fetchrow->array ()) ...
```

Значение выбирается в контексте списка, таким образом, этот тест даст сбой только тогда, когда больше не останется строк. Но тест даст сбой загадочным образом, если он будет написан так:

```
while (my $val = $sth->fetchrow->array ()) ...
```

Различие здесь заключается в том, что значение выбирается в скалярном контексте, поэтому переменная $val может иметь нулевое значение, значение undef или быть пустой строкой. В этом случае тест определит ложное значение и прекратит выполнение цикла, даже несмотря на то, что не был достигнут конец результирующего набора.

Метод fetchrow_arrayref() аналогичен методу fetchrow_array(), но он возвращает не массив, содержащий значение столбцов текущих строк, а ссылку на массив или значение undef, если строк больше не осталось. Этот метод можно запустить следующим образом:

```
while (my $ary_ref = $sth->fetchrow_arrayref ())
{
  $delim = "";
  for (my $i = 0; $i < @{$ary_ref}; $i++)
  {
    $ary_ref->[$i] = "" if !defined ($ary_ref->[$i]); # значение NULL?
    print $delim . $ary_ref->[$i];
    $delim = ",";
  }
  print "\n";
}
```

Доступ к элементам массива осуществляется ссылкой на массив $ary_ref. Для использования механизма указателей здесь применяется нотация $ary_ref->[$i], а не $ary[$i]. Для обращения к массиву можно воспользоваться конструкцией @{$ary_ref}.

Метод fetchrow_arrayref() не подходит для выборки переменных в список. Так, например, такой цикл работать не будет:

```
while (my ($var1, $var2, $var3, $var4)
          = @{$sth->fetchrow_arrayref ()})

{
  # произвести определенные действия над переменными
}
```

Пока метод fetchrow_arrayref() действительно производит выборку строк, этот цикл функционирует нормально. Но как только строки завершаются,

fetchrow_arrayref() возвращает значение undef, и выражение @{undef} становится ошибочным. (Чтобы было понятно – это аналогично ссылке на пустой указатель в C-программе.)

Еще один метод выборки строк fetchrow_hashref() работает следующим образом:

```
while ($hashref = $sth->fetchrow_hashref ())
{
   $delim = "";
   foreach my $key (keys (%{$hashref})
   {
    $hash_ref->{$key} = "" if !defined ($hash_ref->{$key}); # значение
NULL?
    print $delim . $hashref->{$key};
    $delim = ",";
   }
   print "\n";
}
```

Каждый вызов fetchrow_hashref() возвращает ссылку на хэш строковых значений или, если строк больше не осталось, значение undef. В этом случае чередование значений столбцов не имеет какого-либо определенного порядка и элементы хэша языка Perl не следуют в каком-то определенном порядке. Однако элементы хэша имеют ключевые ссылки на имена столбцов, таким образом, $hashref дает одну переменную, с помощью которой можно получить доступ к каждому столбцу по имени. Это позволяет получать значения (или любое их подмножество) в нужном порядке, и совсем не обязательно знать порядок, в котором были выбраны столбцы в запросе SELECT. Например, получить имя и электронный адрес можно следующим образом:

```
while ($hashref = $sth->fetchrow_hashref ())
{
   my $delim = "";
   foreach my $key ("last_name", "first_name", "suffix", "email")
   {
    $hash_ref->{$key} = "" if !defined ($hash_ref->{$key}); # значение
NULL?
    print $delim . $hashref->{$key};
    $delim = ",";
   }
   print "\n";
}
```

Процедура fetchrow_hashref() очень удобна при передаче функции последовательности значений, не зависящих от порядка, в котором столбцы были перечислены в операторе SELECT. В этом случае для выборки строк можно вызвать fetchrow_hashref() и создать функцию, которая получает доступ к хэшу по именам столбцов.

Необходимо быть очень внимательным при использовании процедуры fetchrow_hashref().

- Метод fetchrow_hashref() не может считаться лучшим решением, если необходимо экономить каждую секунду, так как он не настолько эффективен, как метод fetchrow_array() или fetchrow_arrayref().

- Имена столбцов, которые здесь используются в качестве ключевых значений в хэше, имеют тот же регистр, что и в операторе SELECT. В СУБД MySQL имена столбцов не чувствительны к регистру, поэтому запрос будет работать независимо от того, в каком регистре набраны имена столбцов. Но ключевые имена хэш-памяти языка написания сценариев Perl чувствительны к регистру. Это несоответствие может вызвать определенные затруднения в работе. Для того чтобы избежать вероятных проблем несоответствия регистра, можно непосредственно задать `fetchrow_hashref()` режим обработки имен в определенном регистре, передав методу атрибут `NAME_lc` или `NAME_uc`:

```
$hash_ref = $sth->fetchrow_hashref ('NAME_lc');
# имена в нижнем регистре
$hash_ref = $sth->fetchrow_hashref ('NAME_uc');
# имена в верхнем регистре
```

- Структура хэша такова, что один элемент содержит одно уникальное имя столбца. Выборка же может содержать объединение нескольких таблиц, имена столбцов которых могут совпадать. В этом случае значение таких столбцов получить невозможно. Например, `fetchrow_hashref()` вернет только один элемент при запросе типа

```
SELECT a.name, b.name FROM a, b WHERE a.name = b.name
```

  Для того чтобы разрешить эту проблему, необходимо пользоваться псевдонимами. Это позволит дать каждому столбцу уникальное имя. Например, переписав запрос следующим образом (тогда метод `fetchrow_hashref()` вернет ссылку на хэш с двумя элементами `name` и `name2`):

```
SELECT a.name, b.name AS name2 FROM a, b WHERE a.name = b.name
```

Определение количества строк, возвращаемых запросами

Каким образом можно определить количество строк, возвращаемых запросом SELECT или другим подобным ему запросом? Конечно, одним способом является подсчет возвращаемых столбцов по мере их выборки. И действительно, это единственный универсальный способ узнать количество строк, возвращаемых запросом SELECT. Пользуясь драйвером СУБД MySQL, после вызова `execute()` можно вызвать метод `rows()` с помощью дескриптора оператора, но такой прием неприменим для других баз данных. Более того, даже для СУБД MySQL метод `rows()` при установленном атрибуте `mysql_use_result` до полной выборки всех строк не возвращает правильного результата. (Подробнее об этом — в приложении Ж, "Программный интерфейс DBI языка написания сценариев Perl".) Так что можно просто подсчитать количество строк по мере их выборки.

Выборка результатов, содержащих одну строку

Совсем не обязательно создавать цикл для того, чтобы произвести выборку, заведомо содержащую одну строку. Предположим, что вы собираетесь написать сценарий `count_members.pl`, который подсчитывает количество членов "Исторической Лиги" на данный момент времени. Следующий программный код выполняет такой запрос:

```
# создать запрос
my $sth = $dbh->prepare ("SELECT COUNT(*) FROM member");
$sth->execute ();

# прочесть результаты запроса
my $count = $sth->fetchrow_array ();
$sth->finish ();
$count = "трудно судить" if [defined ($count);
print "$count\n";
```

Оператор SELECT возвращает только одну строку, поэтому нет необходимости в цикле, так как метод `fetchrow_array()` вызывается только один раз. Кроме того, так как производится выборка только одного столбца, нет необходимости присваивать возвращаемое значение массиву. При вызове `fetchrow_array()` в скалярном контексте (когда ожидается возвращение одного, а не нескольких значений) она возвращает первый столбец строки или значение `undef`, если строк больше нет. Интерфейс DBI не определяет, какой элемент строки возвращает функция `fetchrow_array()` в скалярном контексте, но для данного запроса все нормально. Он делает выборку только одного значения, поэтому не возникает никакой неоднозначности относительно того, какое значение им возвращается.

Этот код вызывает функцию `finish()` для освобождения памяти, занимаемой результирующим набором, даже если последний содержит только одну строку. (Функция `fetchrow_array()` неявно освобождает результирующий набор после достижения конца набора, но здесь это произойдет только при ее повторном вызове.)

Еще одним типом запроса, для которого ожидается как максимум одна запись, является запрос, содержащий конструкцию `LIMIT 1`, ограничивающую количество возвращаемых строк одной строкой. Наиболее традиционным применением запросов такого типа является возвращение строки, содержащей максимальное или минимальное значение для определенного столбца. Например, следующий запрос возвращает имя и дату рождения самого молодого президента:

```
my $query = "SELECT last_name, first_name, birth"
   . " FROM president ORDER BY birth DESC LIMIT 1";
my $sth = $dbh->prepare ($query);
$sth->execute ();

# получить результаты запроса
my ($last_name, $first_name, $birth) = $sth->fetchrow_array ();
$sth->finish ();
if ((!defined ($last_name))
{
   print "Запрос не дал результатов \n";
}
else
{
   print "Самый молодой президент: $first_name $last_name ($birth)\n";
}
```

Другими типами запросов, которым не требуется цикл выборки, являются запросы, использующие агрегирующую функцию `MAX()` или `MIN()` для выборки единственного значения. Но во всех этих случаях существует еще один способ получить результат, состоящий из одной строки, — воспользоваться методом `selectrow_array()`, объединяющим возможности методов

prepare(), execute() и выборки строк в одном вызове. Этот метод возвращает массив (а не ссылку на массив) или пустой массив в случае возникновения ошибки. С помощью метода selectrow_array() можно переписать предыдущий пример следующим образом:

```
my $query = "SELECT last_name, first_name, birth"
   . " FROM president ORDER BY birth DESC LIMIT 1";
my ($last_name, $first_name, $birth) = $dbh->selectrow_array ($query);
if (!defined ($last_name))
{
   print "Запрос не дал результатов \n";
}
else
{
   print "Самый молодой президент: $first_name $last_name ($birth)\n";
}
```

Выборка результирующих наборов данных

Интерфейс DBI не позволяет производить выборку в произвольном порядке и обрабатывать возвращаемые строки в порядке, отличном от того, в котором они поступают. Кроме того, после выборки строки предыдущая выбранная строка теряется, если до этого ее результаты не будут сохранены в памяти. Такое положение вещей не всегда удобно по ряду причин. Например, они являются нежелательными, если необходимо сделать несколько проходов по строкам, чтобы выполнить статистические вычисления. (Вероятно, вам потребуется пройти результирующий набор один раз, чтобы получить общие цифровые характеристики, а затем пройти строки еще раз для осуществления более детального анализа.)

Получить доступ к результирующему набору как к одному целому можно разными способами. Это можно сделать с помощью обычного построчного цикла выборки, сохраняя каждую строку после ее выборки. Можно воспользоваться методом, который сразу же возвращает весь результирующий набор. Но в любом случае будет получена матрица, содержащая результирующий набор данных. Элементы матрицы можно обрабатывать в произвольном порядке столько раз, сколько это необходимо. Ниже сравниваются оба эти подхода.

Задействовать цикл выборки с целью получить результирующий набор данных можно с помощью метода fetchrow_array(), сохраняя при этом ссылки на строки. Вот исходный текст программы, который производит цикл выборки и печати, аналогичный представленному в процедуре dump_members.pl, за исключением того, что сначала он сохраняет все выбранные строки, а затем, для того, чтобы продемонстрировать определение количества строк и столбцов в матрице и осуществить доступ к отдельным ее элементам, распечатывает всю матрицу:

```
my @matrix = (); # массив ссылок на массивы

while (my @ary = $sth->fetchrow_array ())      # построчная выборка
{
   push (@matrix, [ @ary ]); # сохранить ссылку на только что
                             # выбранную строку
}

# определить размеры матрицы
```

```
my $rows = scalar (@matrix);
my $cols = ($rows == 0 ? 0 : scalar (@{$matrix[0]}));

for (my $i = 0; $i < $rows; $i++)        # Построчная печать
{
  my $delim = "";
  for (my $j = 0; $j < $cols; $j++)
  {
    $matrix[$i][$j] = "" if !defined ($matrix[$i][$j]); # значение NULL?
    print $delim . $matrix[$i][$j];
    $delim = ",";
  }
  print "\n";
}
```

При определении размеров матрицы сначала определяется количество столбцов в матрице, потому что число столбцов матрицы обусловлено тем, пуста эта матрица или нет. Если значение переменной `$rows` равно 0, матрица будет пустой и `$cols` тоже будет равно 0. В противном случае количество столбцов можно принять равным количеству элементов в первой строке, пользуясь синтаксисом `@{$matrix[0]}`, позволяющим получить доступ к строке как к одному целому.

В предыдущем примере мы сначала производили выборку каждой строки, а затем запоминали ссылку на нее. Можно предположить, что гораздо эффективнее вызвать `fetchrow_arrayref()`, а не выполнять непосредственный поиск ссылок на строки:

```
my @matrix = (); # массив ссылок на массив
while (my $ary_ref = $sth->fetchrow_arrayref ())
{
  push (@matrix, $ary_ref);       # запомнить ссылку на только
                                  # что выбранную строку
}
```

Такая конструкция не будет работать потому, что функция `fetchrow_arrayref()` повторно использует массив, на который делается ссылка. Полученная матрица будет представлять собой массив ссылок, каждая из которых будет указывать на одну и ту же последнюю найденную строку. Поэтому, если необходима построчная выборка, лучше воспользоваться методом `fetchrow_array()`, а не `fetchrow_arrayref()`.

В качестве альтернативы использованию цикла выборки можно воспользоваться одним из методов интерфейса DBI, возвращающим весь результирующий набор. Например, метод `fetchall_arrayref()` возвращает массив ссылок, каждый из которых ссылается на содержимое одной строки результирующего набора. Действительно, возвращаемое значение является ссылкой на матрицу. Перед использованием метода `fetchall_arrayref()` вызовите последовательно `prepare()` и `execute()`, а затем выберите результат следующим образом:

```
# выборка всех строк в ссылку на массив ссылок
my $matrix_ref = $sth->fetchall_arrayref ();
```

Задать размерность массива и получить доступ к элементам массива можно так:

```
# определить размерность матрицы
my $rows = (!defined ($matrix_ref) ? 0 : scalar (@{$matrix_ref}));
my $cols = ($rows == 0 ? 0 : scalar (@{$matrix_ref->[0]}));
```

```
for (my $i = 0; $i < $rows; $i++)  # построчная печать
{
   my $delim = "";
   for (my $j = 0; $j < $cols; $j++)
   {
     $matrix_ref->[$i][$j] = "" if !defined ($matrix_ref->[$i][$j]);
                             # NULL?
     print $delim . $matrix_ref->[$i][$j];
     $delim = ",";
   }
   print "\n";
}
```

Метод `fetchall_arrayref()` возвращает ссылку на пустой массив, если результирующий набор пуст. Метод возвращает значение `undef` в случае ошибки, поэтому, когда параметр `RaiseError` не включен, необходимо проверить возвращаемое значение перед его использованием.

Количество строк и столбцов определяется заполнением матрицы. Для доступа ко всей строке `$i` матрицы как к массиву можно воспользоваться синтаксической конструкцией `@{$matrix_ref->[$i]}`.

Конечно, намного проще для выборки результирующего набора воспользоваться методом `fetchall_arrayref()`, чем писать специальный цикл выборки строк, хотя доступ к элементам массива при этом будет несколько усложнен. Существует метод, во многом аналогичный функциональным возможностям метода `fetchall_arrayref()` в случайном порядке, однако более функциональный. Это метод `selectall_arrayref()`. Он выполняет полную последовательность операций `prepare()`, `execute()` и цикл выборки. Для того чтобы работать с методом `selectall_arrayref()`, необходимо передать запрос непосредственно этому методу с помощью дескриптора базы данных:

```
# выборка всех строк в ссылку на массив ссылок
my $matrix_ref = $dbh->selectall_arrayref ($query);

# определить размерность матрицы
my $rows = (!defined ($matrix_ref ) ? 0 : scalar (@{$matrix_ref }));
my $cols = ($rows == 0 ? 0 : scalar (@{$matrix_ref ->[0]}));

for (my $i = 0; $i < $rows; $i++) # построчная печать
{
   my $delim = "";
   for (my $j = 0; $j < $cols; $j++)
   {
     $matrix_ref->[$i][$j] = "" if !defined ($matrix_ref->[$i][$j]);
                             # NULL?
     print $delim . $matrix_ref ->[$i] [$j];
     $delim = ",";
   }
   print "\n";
}
```

Проверка пустых значений

При выборке данных из базы необходимо различать значения "NULL", нулевые значения столбцов и пустые строки. Это достаточно просто потому, что интерфейс

BI возвращает значения "NULL" столбцов как undef. Однако при этом необходи-о делать соответствующую проверку. Вот фрагмент программного кода, который егда распечатывает "false!":

```
ol_val = undef; if (!$col_val)  { print "false!\n"; }
ol_val =0; if (!$col_val)   { print "false!\n"; }
ol_val = ""; if (!$col_val) { print "false!\n"; }
```

Это покажет, что данная форма проверки не определяет различие между зна-ениями undef, 0 и просто пустой строкой. Следующий фрагмент напечатает alse!" для обоих тестов, показывая, что тест не может отличить значение ndef от пустой строки:

```
ol_val = undef; if ($col_val eq "") { print "false!\n"; }
ol_val = ""; if ($col_val eq "")    { print "false!\n"; }
```

Вот фрагмент, который делает то же самое, из чего можно сделать вывод, что орой тест не может отличить значение 0 от пустой строки:

```
ol_val = "" ;
 ($col_val eq "")  { print "false!\n"; }
 ($col_val == 0)   { print "false!\n"; }
```

Для нахождения различий между столбцами со значением undef ("NULL") и олбцами, имеющими другие значения, воспользуйтесь методом defined(). осле того как вы убедитесь в том, что значение не является пустым, можно при-упить к анализу других значений. Например:

```
 (!defined ($col_val)) { print "NULL\n"; }
.sif ($col_val eq "")   { print "пустая строка \n"; }
.sif ($col_val == 0)    { print "нуль \n"; }
.se{ print "другое \n"; }
```

Здесь очень важно соблюдать правильную очередность проверки. Это важно тому, что если $col_val является пустой строкой, вторая и третья операции авнения дают результат "True". Если изменить порядок следования этих опе-аторов, пустые строки ошибочно будут идентифицированы как нули.

Использование кавычек

До сих пор мы конструировали запросы самым простейшим образом с исполь-ванием кавычек. Проблемы на лексическом уровне языка написания сценариев erl возникают в случае, когда строки, заключенные в кавычки, содержат величи-ы, заключенные в кавычки. Затруднения могут возникнуть и на уровне SQL, ко-а необходимо добавить или выбрать значения, которые содержат кавычки, об-атную косую черту или двоичные данные. При задании запроса в виде строки, ключенной в кавычки, нужно всячески избегать любого появления кавычек утри строки запроса:

```
uery = 'INSERT absence VALUES(14,\'2002-9-16\')';
uery = "INSERT absence VALUES(14,\"2002-9-16\")";
```

Синтаксис Perl и MySQL позволяет заключать строки как в одинарные, так и в ойные кавычки. Таким образом, можно избежать неоднозначности:

```
uery = 'INSERT absence VALUES(14,"2002-9-16")';
uery = "INSERT absence VALUES(14,'2002-9-16')";
```

Однако эти два типа кавычек в Perl не эквивалентны. Ссылки на переменные интерпретируются только внутри двойных кавычек. Поэтому одинарные кавычки при создании запросов не помогут, когда в строке запроса имеется ссылка на переменные. Например, для $var, равного 14, следующие две строки не будут идентичны:

```
"SELECT * FROM member WHERE id = $var"
'SELECT * FROM member WHERE id = $var'
```

Строки интерпретируются следующим образом, и очевидно, что первая строка будет более понятна для сервера MySQL:

```
"SELECT * FROM member WHERE id = 14"
'SELECT * FROM member WHERE id = $var'
```

В качестве альтернативы двойным кавычкам можно назвать конструкцию qq{}, которая позволяет интерпретатору Perl рассматривать все, что находится между "qq{" и "}", как строку, заключенную в двойные кавычки. В названии этой конструкции qq можно рассматривать как аббревиатуру "double-quote" (двойная кавычка). Например, эти две строки эквивалентны:

```
$date = "2002-9-16";
$date = qq{2002-9-16};
```

Конструкция qq{} позволяет совсем не беспокоиться о двойных кавычках и свободно использовать в запросах кавычки (одинарные или двойные). Кроме того, безошибочно интерпретируются ссылки на переменные. Обе эти возможности конструкции qq{} проиллюстрированы следующим оператором INSERT:

```
$id = 14;
$date = "2002-9-16";
$query = qq{INSERT absence VALUES($id,"$date")};
```

В качестве ограничителей конструкции qq можно пользоваться как "{" и "}", так и qq() и qq//. Автор предпочитает форму qq{}, потому что вероятность того, что символ "}" появится в тексте SQL-запроса, значительно ниже, чем вероятность появления там символа ")" или "/". Например, символ ")" появляется внутри только что показанного оператора INSERT. Таким образом, qq() не будет безошибочно интерпретирован при квотировании строки запроса.

Конструкция qq{} позволяет выделить запрос отдельной строкой, если есть необходимость и желание выделить SQL-запрос из программного кода Perl:

```
$id = 14;
$date = "2002-9-16";
$query = qq{
   INSERT INTO absence VALUES($id,"$date")
};
```

Это также имеет смысл делать и с точки зрения придания большей читабельности запросу. Возьмем, например, оператор SELECT из сценария dump_members.pl, имеющий следующий вид:

```
$sth = $dbh->prepare ("SELECT last_name, first_name, suffix, email,"
. "street, city, state, zip, phone FROM member ORDER BY last_name");
```

Применив конструкцию qq{}, придадим оператору более читабельный вид:

```
$sth = $dbh->prepare (qq{
   SELECT
     last_name, first_name, suffix, email,
```

```
    street, city, state, zip, phone
  FROM member
  ORDER BY last_name
});
```

Не редки случаи, когда строки, заключенные в двойные кавычки, тоже могут занимать несколько строк. Но все равно при создании многострочных конструкций я отдают предпочтение конструкции qq{}. Я поймал себя на мысли, что когда вижу непарную кавычку в строке, то думаю, не ошибка ли это? После чего я машинально трачу время зря на поиски закрывающей кавычки.

Конструкция qq{} берет на себя все заботы о работе с кавычками на лексическом уровне интерпретатора Perl. Это позволяет совершенно не заботиться об этой проблеме. Однако необходимо принимать во внимание синтаксис уровня SQL. Рассмотрим такую попытку добавления записи в таблицу member:

```
$last = "O'Malley";
$first = "Brian";
$expiration = "2005-9-1";
$rows = $dbh->do (qq{
  INSERT member (last_name,first_name,expiration)
  VALUES('$last','$first','$expiration')
});
```

Строка, которую метод do() посылает серверу MySQL, будет выглядеть следующим образом:

```
INSERT INTO member (last_name, first_name, expiration)
VALUES('O'Malley','Brian','2005-9-1')
```

В этом SQL-запросе содержится ошибка. Она появилась потому, что одинарная кавычка попадает внутрь строчки, заключенной в одинарные кавычки. В главе 6, "Программный интерфейс MySQL C API", мы уже сталкивались с аналогичной проблемой. Тогда она была вызвана использованием функции mysql_escape_string(). Интерфейс DBI имеет подобный механизм — каждое значение, взятое в кавычки, которое затем будет использоваться в операторе явным образом, вызывает метод quote(), и в дальнейшем используется уже только возвращенное значение. Предыдущий пример лучше переписать следующим образом:

```
$last = $dbh->quote ( "O'Malley");
$first = $dbh->quote ("Brian");
$expiration = $dbh->quote ("2005-9-1");
$rows = $dbh->do (qq{
  INSERT INTO member (last_name,first_name, expiration)
  VALUES($last , $first , $expiration)
});
```

Теперь строка, которую метод do() посылает серверу MySQL, приобретает следующий вид:

```
INSERT INTO member (last_name, first name, expiration)
VALUES('O\'Malley', 'Brian', '2005-9-1')
```

Обратите внимание, что при ссылке на $last и $first в строке запроса нет "окружающих" кавычек — их присваивает метод quote(). При добавлении кавычек в запросе их будет слишком много:

```
$value = "paul";
$quoted_value = $dbh->quote ($value);
```

```
print "...  WHERE name = $quoted_value\n";
print "...  WHERE name =  '$quoted_value' \n";
```

Эти операторы дают следующий результат:

```
... WHERE name = 'paul'
... WHERE name = ' 'paul' '
```

Во втором случае строка содержит слишком много кавычек.

Маркеры и привязка параметров

В предыдущих разделах мы создавали запросы, указывая добавляемые значения и критерии выборки прямо в строке запроса, что совсем не обязательно. Интерфейс DBI позволяет размещать в строке запроса специальные маркеры, а затем во время выполнения запросов подставлять на место этих маркеров реальные значения. Одна цель, которая при этом преследуется, заключается в том, что это позволяет эксплуатировать преимущества метода `quote()` без необходимости явного вызова метода `quote()`. Другая цель — увеличение производительности.

Проиллюстрируем, каким образом работают маркеры. Предположим, что в школе начинается новый семестр и необходимо очистить таблицу `student`, а затем проинициализировать ее, т.е. ввести имена новых учащихся, взяв их из списка имен учащихся, хранящегося в файле. Не пользуясь маркерами, удалить существующие в таблице данные и загрузить новые можно следующим образом:

```
$dbh->do (qq{ DELETE FROM student }); # удалить существующие строки
while (<>)                            # добавить новые строки
{
   chomp;
   $_ = $dbh->quote ($_);
   $dbh->do (qq{ INSERT student SET name = $_ });
}
```

Такой подход требует, чтобы вы обрабатывали специальные символы в данных, вызывая метод `quote()`. Но такая процедура неэффективна потому, что основной запрос `INSERT` статичен, а метод `do()` в цикле вызывает методы `prepare()` и `execute()`. Гораздо более эффективно будет работать процедура, в которой для установки оператора `INSERT` метод `prepare()` вызывается всего один раз перед началом цикла, а в цикле вызывается только метод `execute()`. Это позволяет ограничиться одним вызовом метода `prepare()`. Интерфейс DBI осуществляет это следующим образом:

```
$dbh->do (qq{ DELETE FROM student } ); # удалить существующие строки
my $sth = $dbh->prepare (qq{ INSERT student SET name = ? });
while (<>)                                  # добавить новые строки
{
   chomp;
   $sth->execute ($_);
}
```

Вместо вызова метода `do()` внутри цикла лучше вызывать метод `prepare()` до начала цикла, а внутри него — метод `execute()`. Обратите внимание на знак вопроса в запросе `INSERT`. Это и есть маркер. При выполнении метода `execute()` реальное значение замещает маркер. В таком виде запрос передается

на сервер. В целом, если метод do() находится внутри цикла, гораздо лучше запустить метод prepare() до начала цикла, а внутри цикла оставить только execute(). Вот несколько замечаний, касающихся маркеров.

- Не заключайте символ маркера в кавычки внутри строки запроса. Если это сделать, то данный символ не будет распознан как маркер.
- При определении значений маркеров не используйте метод quote(). В противном случае во вставляемых значениях появятся дополнительные кавычки.
- Можно задавать несколько маркеров в строке запроса, но при этом методу execute() обязательно нужно передавать столько значений, сколько было задано маркеров.
- Каждый маркер должен определять одно значение, а не список значений. Например, такой оператор будет неверным:

```
my $sth = $dbh->prepare (qq{{
        INSERT INTO member last_name, first_name VALUES(?)
});
$sth->execute ("Adams,Bill,2003-09-19");
```

  Чтобы избежать такой ошибки, необходимо поступить следующим образом:

```
my $sth = $dbh->prepare (qq{{
        INSERT INTO member last_name, first_name VALUES(?,?,?)
});
$sth->execute ("Adams","Bill","2003-09-19");
```

- Для определения пустого значения в качестве маркера используйте значение undef.
- Не пытайтесь использовать маркер для ключевых слов или идентификаторов баз данных, таблиц или имен столбцов. Это не даст результата, так как значения маркеров автоматически обрабатываются методом quote(). В таком случае ключевое слово в запросе будет взято в кавычки, и запрос не сработает из-за наличия в нем синтаксической ошибки.

Иногда от использования маркеров можно получить еще один выигрыш. Некоторые базы данных по мере планирования эффективного выполнения запросов кэшируют уже готовые к выполнению запросы. Таким образом, если такой же запрос будет получен повторно, он может быть использован, минуя этап генерации плана выполнения. Кэширование запросов очень эффективно при обработке сложных запросов SELECT, так как генерация хорошего плана выполнения таких запросов может занимать много времени. Маркеры дают больше шансов для создания универсальных запросов и, соответственно, для попадания их в кэш-память. В случае с нашей СУБД MySQL шаблоны не дают такого выигрыша производительности потому, что СУБД MySQL не кэширует запросы. Тем не менее не следует бояться использовать маркеры. Ведь может случиться так, что ваш сценарий, реализующий интерфейс DBI, будет использован для работы с другой СУБД, которая как раз и использует механизм кэширования запросов. И ваш сценарий с маркерами заработает эффективней. (В СУБД MySQL кэширование запросов появилось в версии 4.0.1, но оно выполняет кэширование лексически идентичных строк запроса результирующего набора, а не кэширование плана выполнения. Кэширование запросов обсуждается в главе 4, "Оптимизация запросов".)

Маркеры позволяют подставлять значения в запросы непосредственно во время выполнения запроса. Другими словами, они позволяют параметризировать запрос. Интерфейс DBI также позволяет осуществлять соответствующую операцию вывода, которая называется связыванием параметров. Эта операция позволяет параметризировать вывод и автоматически присваивать выбранные значения столбцов непосредственно переменным, минуя этап принудительного присвоения.

Пустой аргумент undef

Некоторые из методов интерфейса DBI при выполнении строки запроса методом do() и selectrow_array() позволяют привязывать значения маркеров к любому символу '?' из запроса. Например, запись можно обновить так:

```
my $rows = $dbh->do ("UPDATE member SET expiration = ? WHERE member_id = ?",
undef, "2005-01-01", 14);
```

Или выбрать запись следующим образом:

```
my $ref = $dbh->selectrow_arrayref (
"SELECT * FROM member WHERE member_id = ?",
undef, 14);
```

Заметим, что в обоих случаях значениям маркеров предшествует аргумент undef, который, как кажется, ничего не делает. Причина его появления заключается в том, что методы выполнения запросов, позволяющие задавать аргументы с маркерами, предшествующими другим аргументам, могут использоваться для задания атрибутов обработки запросов. Такие атрибуты используются редко (если вообще используются), но аргумент по-прежнему должен присутствовать на своем месте, поэтому зададим его как undef.

Привязка результатов запроса к переменным сценария

Маркеры позволяют подставлять значения в строку запроса во время выполнения запроса. Другими словами, можно параметризировать "вход" запроса. Интерфейс DBI также обеспечивает аналогичную операцию, которая называется привязкой параметров и позволяет параметризировать "вывод" автоматической выборкой значений столбцов в переменные при выборке строки. При этом нет необходимости присваивать значения переменных самим.

Предположим, что у нас есть запрос на выборку имен членов из таблицы member. Можно задать интерфейсу DBI режим, при котором переменным Perl будут назначаться значения выбранных столбцов. При выборке строк переменные будут автоматически инициализированы соответствующими значениями столбцов. Ниже приведен пример, показывающий, каким образом можно связать столбцы с переменными, а затем получить доступ к ним в цикле выборки.

```
my ($last_name, $first_name, $suffix);
my $sth = $dbh->prepare (qq{
    SELECT last_name, first_name, suffix
    FROM member
    ORDER BY last_name, first_name
});
$sth->execute ();
$sth->bind_col (1, \$last_name);
$sth->bind_col (2, \$first_name);
$sth->bind_col (3, \$suffix);
print "$last_name, $first_name, $suffix\n" while $sth->fetch;
```

Каждый вызов метода bind_col() призван определить номер столбца и ссылку на переменную, которая ассоциируется с этим столбцом. Нумерация столбцов начинается с 1. Метод bind_col() должен вызываться после отработ-

ки метода execute() и перед выборкой строк. Существует и другой способ, при котором используется только один вызов метода bind_columns(). В этом случае методу передаются ссылки на переменные:

```
my ($last_name, $first_name, $suffix);
my $sth = $dbh->prepare (qq{
    SELECT last_name, first_name, suffix
    FROM member
    ORDER BY last_name, first_name
});
$sth->execute ();
$sth->bind_columns (\$last_name, \$first_name, \$suffix);
print "$last_name, $first_name, $suffix\n" while $sth->fetch;
```

Метод bind_columns() должен вызываться после отработки метода execute() и перед выборкой строк.

Определение параметров соединения

Наиболее простой способ установки параметров соединения — определение всех параметров в момент выполнения метода connect():

```
my $dsn = "DBI: mysql :db_name:host_name";
my $dbh = DBI->connect ($data_source, user_name, password);
```

При отсутствии параметров соединения происходит следующее.

- Переменная среды DBI_DSN используется в случае, если источник данных не определен или его значение не является пустой строкой. Переменные DBI_USER и DBI_PASS используются, если не определены имя пользователя и пароль (но в этом случае они не могут быть представлены пустой строкой). В ОС Windows в случае неопределенности имени пользователя используется переменная USER.
- По умолчанию принимается имя узла localhost.
- Если имя пользователя имеет значение undef или "пустая строка", по умолчанию используется регистрационное имя UNIX. В ОС Windows имя пользователя имеет значение ODBC.
- Если пароль имеет значение undef или "пустая строка", пароль не передается.

Параметры можно указывать, добавив их в начало выражения и предварив двоеточием. Например, для определения имени конфигурационного файла можно задать параметр mysql_read_default_file:

```
my $dsn = DBI:mysql:sampdb;mysql_read_default_file=/u/paul/.my.cnf";
```

При выполнении сценария открывается файл, содержащий параметры соединения. Предположим, что файл /u/paul/.my.cnf содержит:

```
[client]
host=cobra.snake.net
user=sampadm
password=secret
```

После этого метод connect() попытается подключиться к серверу MySQL, расположенному на узле cobra.snake.net. При успешном соединении будет подключен пользователь sampadm с паролем secret. Чтобы вашим сценарием

мог пользоваться любой пользователь, задайте путь к конфигурационному файлу следующим образом:

```
my $dsn = "DBI:mysql:sampdb;mysql_read_default_file=$ENV{HOME}/.my.cnf";
```

Здесь переменная `$ENV{HOME}` содержит путь к корневому каталогу пользователя, запустившего сценарий. Таким образом, имя узла и имя пользователя, а также его пароль, используемые сценарием, будут взяты из конфигурационного файла пользователя. Для сценария, написанного таким образом, необходимость хранить в своем теле параметры соединения отпадает.

Использование `mysql_read_default_file` позволяет сценарию читать только поименованный конфигурационный файл, что может оказаться нежелательным при необходимости просматривать параметры и в общесистемном конфигурационном файле (таком как `/etc/my.cnf` при работе в ОС UNIX или `C:\my.cnf` при работе с ОС Windows). Для того чтобы этот сценарий просматривал все стандартные конфигурационные файлы на предмет параметров соединения, используется функция `mysql_read_default_group`. Этот вариант позволяет использовать параметры группы `[client]` одновременно с параметрами группы, заданной в вашем конфигурационном файле. Например, если у вас имеются параметры, присущие сценариям, работающим с базой данных `sampdb`, перечислите их в группе `[sampdb]`, а затем воспользуйтесь такими исходными данными:

```
my $dsn = "DBI:mysql:sampdb;mysql_read_default_group= sampdb";
```

Если необходимо пользоваться только группой `[client]` из стандартного конфигурационного файла, задайте параметры следующим образом:

```
my $dsn = "DBI:mysql:sampdb;mysql_read_default_group=client";
```

Работа с параметрами `mysql_read_default_file` и `mysql_read_default_group` возможна только начиная с версии MySQL 3.22.10 и с драйверами DBD::mysql 1.21.06 или выше. Детальную информацию можно найти в приложении Ж, "Программный интерфейс DBI языка написания сценариев Perl". Информация о форматах конфигурационных файлов MySQL приведена в приложении Д, "Программы MySQL".

Одним из "подводных камней", возникающих при использовании `mysql_read_default_file` в ОС Windows, является то, что пути обычно начинаются с обозначения имени устройства и двоеточия. В этом и заключается затруднение, так как интерфейс DBI интерпретирует двоеточие как символ, который разделяет части строки DSN. Эту ситуацию можно обойти следующим способом, хотя этот метод является несколько "искусственным".

1. Разместите конфигурационный файл в корневом каталоге устройства, чтобы пути, заданные без имени устройства, интерпретировались относительно этого устройства.
2. Значением параметра `mysql_read_default_file` задайте имя файла в DSN, не предваряя его при этом именем устройства и двоеточием.
3. При необходимости оставить текущий каталог незадействованным в операции соединения сохраните путь к текущему каталогу перед вызовом функции `connect()` и восстановите его командой `chdir()` после соединения.

В следующем фрагменте программного кода продемонстрировано, как это можно сделать, используя по мере необходимости конфигурационный файл C:\my.cnf. (Заметим, что символ обратной косой черты в описаниях путей ОС Windows в строках переопределяется Perl как обычная косая черта.)

```
# сохранить путь к текущему каталогу
use Cwd;
my $orig_dir = cwd ();
# поменять на путь к корневому каталогу устройства, где расположены файлы
chdir ("C:/ ") or die "Невозможно выполнить команду chdir: $!\n";
# установить соединение с помощью параметров из C:\my.cnf
my $dsn = "DBI:mysql:sampdb:localhost;mysql_read_default_file=/my.cnf";
my $dbh = DBI->connect ($dsn, undef, undef,
                        { RaiseError => 1, PrintError => 0 });
# возвратиться к первоначальному каталогу
chdir ($orig_dir) or die "Невозможно выполнить команду chdir: $!\n";
```

Использование конфигурационных файлов совсем не освобождает вас от необходимости объявления параметров конфигурации при вызове метода connect() (например, в сценарии с подключением определенного пользователя). Любые имена узлов, пользователей и паролей, указанные при вызове метода connect() явным образом, имеют более высокий приоритет, чем параметры, указанные в конфигурационном файле. Например, необходимо, чтобы сценарий анализировал параметры - -host, - -user и - -password, указанные в командной строке, и отдавал приоритет этим значениям перед значениями, указанными в конфигурационном файле. Это достаточно традиционное поведение стандартного пользователя СУБД MySQL. Таким образом, создаваемые вами сценарии DBI должны учитывать такое положение вещей.

В остальных сценариях обработки командной строки, которые приводятся в этой главе, будет использоваться стандартная процедура установки и разрыва соединений. Я только один раз покажу, как это делается, чтобы потом полностью сконцентрироваться на том, для чего был создан каждый конкретный сценарий в отдельности:

```
#! /usr/bin/perl -w
use DBI;
use strict;

# Синтаксический анализ параметров соединения из командной строки, если таковые имеются

use Getopt::Long;
$Getopt::Long::ignorecase = 0; # параметры чувствительны к регистру
$Getopt::Long::bundling = 1; # -uname = -u name, а не -u -n -a -m -e

# параметры по умолчанию — все пропускается

my ($host_name, $password, $port_num, $socket_name, $user_name);
GetOptions (
   # =i обозначает необходимость указания целого числа после параметра
   # =s or :s обозначает необходимость или факультативность указания
строки после параметра
   "host|h=s" => \$host_name,
   "password|p:s" => \$password,
   "port|P=i" => \$port_num,
```

```
   "socket|S=s" => \$socket_name,
   "user|u=s" => \$user_name
) or exit (1);

# запрос пароля, если параметр задан, но значение отсутствует

if (defined ($password) && !$password)
{
   # отключить эхо, не влияя на STDIN
   open (TTY, "/dev/tty") or die "Невозможно открыть терминал \n";
   system ("stty -echo < /dev/tty");
   print STDERR "Вести пароль: ";
   chomp ($password = <TTY>);
   system ("stty echo < /dev/tty");
   close (TTY);
   print STDERR "\n";
}

# создание источника данных

my $dsn = "DBI:mysql:sampdb";
$dsn .= ";host=$host_name" if $host_name;
$dsn .= ";port=$port_num" if $port_num;
$dsn .= "mysql_socket=$socket_name" if $socket_name;
$dsn .= "mysql_read_default_group=client"

# подключение к серверу
my $dbh = DBI->connect ($dsn, $user_name, $password,
{ RaiseError => 1, PrintError => 0 });
```

Этот программный код инициализирует интерфейс DBI, принимает параметры соединения из командной строки, а затем с помощью параметров, полученных из командной строки или заданных в группе `[client]` из стандартного конфигурационного файла `~/.my.cnf` пользователя, запустившего сценарий, осуществляет соединение с сервером MySQL. Это значит, что если в корневом каталоге такого пользователя имеется конфигурационный файл `.my.cnf`, то дополнительно вводить параметры соединения нет необходимости. Только нужно установить такой режим, который обеспечит защиту этого файла от чтения.

Финальная часть нашего сценария будет очень похожа на окончания многих других сценариев. Она просто завершает сеанс связи и работу сценария:

```
$dbh->disconnect ();
exit (0);
```

Когда мы перейдем к разделу, повествующему о программировании в Web, "Работа с интерфейсом DBI в Web-приложениях", код установки соединения будет немного видоизменен, но основная идея останется прежней.

Программный код с подсказкой работает только под управлением ОС UNIX. Для ОС Windows есть альтернатива – поместить пароль СУБД MySQL в группу `[client]` одного из стандартных конфигурационных файлов или указать его непосредственно в командной строке.

Существует одно различие между тем, как стандартные клиенты СУБД MySQL и модуль Getopt обрабатывают параметры, заданные в командной строке. Модуль Getopt не позволяет задавать параметр пароля (`--password` или `-p`) без аргументов,

если этот аргумент не является последним аргументом в командной строке или если сразу же за ним не следует другой параметр. Предположим, что ваш сценарий ожидает имя таблицы после параметров. В этом случае модуль Getopt интерпретирует mytbl как значение пароля, а не как подсказку для введения пароля:

```
% ./myscript.pl -u paul -p mytbl
```

Для того чтобы сценарий был вынужден выводить подсказку на ввод пароля, параметр -p должен предшествовать параметру -u:

```
% ./myscript.pl -p -u paul mytbl
```

Отладка

При отладке сценария DBI, работающего неправильно, прибегают к двум методам, применяя их каждый отдельно или оба сразу. Во-первых, можно разместить в различных местах сценария операторы печати. Это позволяет создавать удобную отладочную диагностику, но вводить эти операторы приходится вручную. Во-вторых, можно использовать встроенные средства трассировки интерфейса DBI. Это более универсальная возможность, но и более систематическая, запускаемая автоматически при ее включении. Трассировка интерфейса DBI отображает информацию о работе самого драйвера, которую невозможно получить иным путем.

Отладка с использованием операторов печати

В вопросах, поступающих в список рассылки СУБД MySQL, одним из самых часто встречающихся является: "У меня есть запрос, который отлично работает при запуске из mysql, но совсем не работает из сценария DBI. Что мне делать?" Очень часто оказывается, что сценарий DBI генерирует запрос, совершенно непохожий на тот, который ожидает от него создавший его программист. И, если запрос распечатать до его отсылки на сервер, будет очень интересно посмотреть, что собственно посылается на сервер. Предположим, что запрос, отсылаемый из mysql, выглядит следующим образом (без точки с запятой в конце):

```
mysql> INSERT INTO member (last_name, first_name, expiration)
    -> VALUES("Brown","Marcia","2005-6-3")
```

Попробуем сделать то же самое из сценария DBI:

```
$last = "Brown";
$first = "Marcia";
$expiration = "2005-6-3";
$query = qq{
   INSERT INTO member (last_name,first_name,expiration)
   VALUES($last, $first, $expiration)
};
$rows = $dbh->do ($query);
```

Несмотря на то что это абсолютно аналогичный запрос, такая конструкция не работает. В чем дело? Попробуем распечатать получившийся запрос:

```
print "$query\n";
```

Вот результат:

```
INSERT INTO member (last_name, first_name, expiration)
VALUES(Brown,Marcia,2005-6-3)
```

Здесь отчетливо видно, что мы забыли заключить в кавычки значения колонок в списке VALUES(). Правильнее здесь будет воспользоваться методом quote():

```
$last = $dbh->quote ("Brown");
$first = $dbh->quote ("Marcia");
$expiration = $dbh->quote ("2002-6-3");
$query =
qq{
   INSERT INTO member (last_name, first_name, expiration)
   VALUES($last, $first, $expiration)
};
$rows = $dbh->do ($query);
```

Есть и другой способ — создать запрос с использованием маркеров и передавать значения с помощью аргументов метода do():

```
$last = "Brown";
$first = "Marcia";
$expiration = "2005-6-3";
$query =
qq{
   INSERT INTO member (last_name, first_name, expiration)
   VALUES(?,?,?)
};
$rows = $dbh->do ($query, undef, $last, $first, $expiration);
```

К сожалению, в этом случае с помощью оператора print нельзя увидеть, как собственно выглядит полный запрос, потому что значения, представленные маркерами, нельзя оценить до запуска метода do(). При использовании маркеров в качестве метода отладки больше подойдет трассировка.

Отладка с применением трассировки

Для определения, почему интерфейс DBI работает неверно, существует режим генерирования трассировочной (отладочной) информации. Есть десять уровней трассировки от 0 (выключено) до 9 (максимальное количество информации). Наиболее удобными являются уровни 1 и 2. На втором уровне отображаются выполняемые запросы (включая запросы, полученные после подстановок), результаты обращений к методу quote() и т.д. Эту помощь трудно переоценить при решении проблем, возникающих в процессе отладки.

Трассировкой можно управлять из отдельного сценария с помощью метода trace() или переменной управления DBI_TRACE для установки режима отладки всех запускаемых вами сценариев DBI.

Для того чтобы воспользоваться вызовом trace(), необходимо передать аргумент уровня и имя файла. Если имя файла не указано, весь вывод трассировки идет на устройство STDERR, если указано — в указанный файл. Вот некоторые примеры:

```
DBI->trace (1);
```

Уровень 1: трассировка производится на устройство STDERR.

```
DBI->trace (2,"trace.out")
```

Уровень 2: трассировка производится в файл "trace.out".

```
DBI->trace (0)
```

Трассировка отключена.

При запуске DBI->trace() все операции с интерфейсом DBI трассируются. Более тонкий метод заключается в возможности трассировки на уровне отдельного дескриптора. Такой подход применим, когда известно, в каком месте сценария возникла проблема, и вам совсем не нужна регистрация того, что происходило до этого места программы. Например, если возникла проблема с запросом SELECT, можно оттрассировать дескриптор оператора, связанного только с этим запросом:

```
$sth = $dbh->prepare (qq{ SELECT ... }); # Создать дескриптор
                                         # оператора
$sth->trace (1); # разрешить трассировку оператора
$sth->execute ();
```

Если при вызове метода trace() указано имя файла, вся диагностика будет записываться в этот файл.

Альтернативой методу trace() является использование атрибута TraceLevel, появившегося в версии 1.21 интерфейса DBI. Он позволяет устанавливать или изменять уровень трассировки для заданного дескриптора:

```
$dbh->{TraceLevel} = 3; # установить уровень трассировки базы данных
my $cur_level = $sth->{TraceLevel}; # получить уровень трассировки дескриптора оператора
```

Для того чтобы включить трассировку для всех запущенных вами сценариев DBI, установите переменную среды DBI_TRACE из оболочки. Синтаксис команды установки полностью зависит от используемой оболочки.

- Для оболочек csh или tcsh:

  ```
  % setenv DBI_TRACE значение
  ```

- Для оболочек sh, ksh или bash:

  ```
  $ export DBI_TRACE=значение
  ```

- Для ОС Windows:

  ```
  C:\> set DBI_TRACE=значение
  ```

Формат *значение* для всех оболочек аналогичен: число *n* для включения трассировки уровня *n* на устройство STDERR; имя файла для включения трассировки второго уровня в указанный файл или *n=file_name*. Вот несколько примеров с использованием синтаксиса оболочки tcsh:

- Трассировка уровня 1 на устройство STDERR:

  ```
  % setenv DBI_TRACE 1
  ```

- Трассировка уровня 1 в "trace.out":

  ```
  % setenv DBI_TRACE 1=trace.out
  ```

- Трассировка уровня 2 в "trace.out":

  ```
  % setenv DBI_TRACE trace.out
  ```

Применение переменной DBI_TRACE предпочтительнее, так как она позволяет производить трассировку DBI-сценариев без внесения каких-либо изменений в сценарии. Включив режим трассировки, не забудьте отключить его после устранения проблемы. Отладочная информация будет добавляться в трассировочный файл. В результате этого файл может вырасти до огромных размеров. Категори-

чески нельзя устанавливать DBI_TRACE в стартовом файле .cshrc, .login или .profile!

В ОС UNIX отключить режим трассировки можно с помощью одной из следующих команд.

- Для оболочек sh, ksh или bash сделайте следующее:

```
% setenv DBI_TRACE 0
% unsetenv DBI_TRACE
```

- В оболочках sh, ksh или bash сделайте следующее:

```
$ DBI_TRACE=0
```

- В ОС Windows отключить трассировку можно с помощью таких команд:

```
C:\> unset DBI_TRACE
C:\> set DBI_TRACE=0
```

Метаданные результирующего набора

Интерфейс DBI позволяет получить доступ к метаданным результирующего набора, т.е. к описанию строк, выбранных по запросу. Это будет доступ к атрибутам дескриптора оператора, ассоциированного с запросом, генерировавшим этот результат. Некоторые из этих атрибутов являются стандартными атрибутами интерфейса DBI, доступ к которым можно получить с помощью любого драйвера базы данных (таких как NUM_OF_FIELDS, т.е. количество столбцов в результирующем наборе). Другие атрибуты присущи только MySQL. Их можно получить с помощью драйвера MySQL для интерфейса DBI DBD::mysql. К числу таких атрибутов можно причислить атрибут mysql_max_length, содержащий информацию о максимальной длине значений в каждом столбце. Такие атрибуты неприменимы для работы с другими базами данных. Чем больше в сценарии используется атрибутов, характерных для работы с СУБД MySQL, тем менее данный сценарий будет переносим на другие базы данных, что позволяет упростить получение нужной информации.

Метаданные нужно запрашивать своевременно. Атрибуты результирующего набора будут недоступны для оператора SELECT до того момента, пока не будут запущены методы prepare() и execute(). Кроме того, атрибуты могут оказаться ошибочными после достижения конца результирующего набора функцией выборки строк или запуска метода finish().

Рассмотрим, как можно использовать один из атрибутов метаданных СУБД MySQL mysql_max_length одновременно с более общим атрибутом NUM_OF_FIELDS и атрибутом уровня DBI NAME, который хранит имена столбцов запроса. Информацию, полученную от этих атрибутов, можно использовать одновременно. Напишем сценарий box_out.pl, который делает вывод из запросов SELECT в том же стиле, что и клиент mysql, работающий в интерактивном режиме. Основное тело сценария box_out.pl имеет следующий вид (здесь оператор SELECT можно заменить любым другим оператором, так как программы ввода-вывода не зависят от конкретного запроса):

```
my ($sth) = $dbh->prepare (qq{
  SELECT last_name , first_name, city, state
  FROM president ORDER BY last_name, first_name
```

```
});
$sth->execute (); # атрибуты становятся доступными
                  # после этого вызова

# реальная максимальная ширина столбца в результирующем наборе
my @wid = @{$sth->{mysql_max_length}};
# количество столбцов в результирующем наборе
my $ncols = $sth->{NUM_OF_FIELDS};

# настроить ширину столбцов,
# если заголовки столбцов шире значений данных
for (my $i = 0; $i < $ncols; $i++)
{
my $name_wid = length ($sth->{NAME}->[$i] ) ;
  $wid[$i] = $name_wid if $wid[$i] < $name_wid;
  $wid[$i] = 4 if $wid[$i] < 4;
}

# печать результата
print_dashes (\@wid, $ncols);              # строка тире
print_row ($sth->{NAME}, \@wid, $ncols);   # заголовки столбцов
print_dashes (\@wid, $ncols);              # строка тире
while (my $ary_ref = $sth->fetchrow_arrayref ())
{
  print_row ($ary_ref, \@wid, $ncols);     # строка данных
}
print_dashes (\@wid, $ncols);              # строка тире
```

Сразу же после инициализации запроса с помощью метода `execute()` мы сможем получить любые необходимые нам метаданные. Выражения `$sth->{NAME}` и `$sth->{mysql_max_length}` дают имена и максимальную длину значения столбцов. Значение каждого атрибута является ссылкой на массив, содержащий по одному значению для каждого столбца результирующего набора, в порядке их перечисления в запросе.

Остальные вычисления очень похожи на аналогичные вычисления, которыми мы пользовались в главе 6, "Программный интерфейс MySQL C API". Например, чтобы не допустить неформатированного вывода, можно увеличить ширину столбца, если имя столбца шире значений данных столбца.

Функции вывода данных `print_dashes()` и `print_row()` аналогичны соответствующим кодам в программах-клиентах:

```
sub print_dashes
{
my ($wid_ary_ref ) = shift;   # ширина столбца
my ($cols) = shift;           # число столбцов

  print "+";
  for (my $i = 0; $i < $cols; $i++)
  {
    print "-" x ($wid_ary_ref->[$i]+2) . "+";
  }
    print "\n";
}

# печать строки данных
# (не выравнивает по правому краю числовые данные)
```

```
sub print_row
{
my $val_ary_ref = shift; # значения столбцов
my $wid_ary_ref = shift; # ширина столбцов
my $cols = shift;        # количество столбцов

  print " | ";
  for (my $i = 0; $i < $cols; $i++)
  {
    printf " %-*s | ", $wid_ary_ref->[$i],
    defined ($val_ary_ref->[$i]) ? $val_ary_ref->[$i] : "NULL";
  }
  print "\n";
}
```

Сценарий box_out.pl выводит следующую информацию:

```
+-----------+------------+--------+------------+-------+
| last_name | first name | suffix |    city    | state |
+-----------+------------+--------+------------+-------+
| Adams     | John       | NULL   | Braintree  | MA    |
| Adams     | John Quincy| NULL   | Braintree  | MA    |
| Arthur    | Chester A. | NULL   | Fairfield  | VT    |
| Buchanan  | James      | NULL   | Mercersburg| PA    |
| Bush      | George H.W.| NULL   | Milton     | MA    |
| Bush      | George W.  | NULL   | New Haven  | CT    |
| Carter    | James E.   | Jr.    | Plains     | GA    |
...
```

Следующий сценарий использует метаданные столбцов для вывода данных в разных форматах. Сценарий show_member.pl позволит быстро просмотреть данные о членах "Исторической Лиги", не требуя ввода запросов. По фамилии члена лиги этот сценарий отобразит выбранную запись:

```
% ./show_member artel
last_name: Artel
first_name: Mike
suffix:
expiration: 2003-04-16
email: mike_artel@venus.org
street: 4264 Lovering Rd.
city: Miami
state: FL
zip: 12777
phone: 075-961-0712
interests: Civil Rights, Education, Revolutionary War
member id: 63
```

Сценарий show_members.pl можно инициировать с помощью идентификационного номера или шаблона по фамилиям. Следующие команды делают выборку данных по идентификационному номеру члена лиги, равному 23, и выборку всех членов лиги, фамилии которых начинаются на "C":

```
% show_member.pl 23
% show_member.pl c%
```

Основное тело программного кода сценария show_member.pl приведено внизу. Здесь атрибут NAME используется для обозначения названий столбцов, а

атрибут NUM_OF_FIELDS — для определения количества столбцов, содержащихся в результирующем наборе:

```
my $count = 0;          # количество распечатываемых строк
my @label = ();         # массив наименований столбцов
my $label_wid = 0;

while (@ARGV)           # выполнить запрос для каждого аргумента
                        # командной строки
{
my $arg = shift (@ARGV);
my ($sth, $clause, $address);

  # умолчание — поиск по фамилии...
  my $clause = "last_name LIKE " . $dbh->quote ($arg);
  $clause = "last_name LIKE " . $dbh->quote ($arg);
  # и по идентификационному номеру, если получен числовой аргумент
  $clause = "member_id = " . $dbh->quote ($arg) if $arg =- /^\d+$/;
  # генерировать запрос
  my $sth = $dbh->prepare (qq{
    SELECT * FROM member
    WHERE $clause
    ORDER BY last_name, first_name
  });
  $sth->execute ();

  # определение имен столбцов для наименования и
  # определение максимальной ширины имени столбца
  # для форматирования
  # (делать это только в начале цикла)
  if ($label_wid == 0)
  {
    @label = @{$sth->{NAME}};
    foreach my $label (@label)
    {
        $label_wid = length ($label) if $label_wid < length ($label);
    }
  }
  # получить и распечатать результаты запроса, а затем очистить

  my $matches = 0;
  while (my @ary = $sth->fetchrow_array ())
  {
  # вывести символ перевода строки перед второй
  # и последующими строками
    print "\n" if ++$count > 1;
    foreach (my $i = 0; $i < $sth->{NUM_OF_FIELDS}; $i++)
    {
        # распечатать название столбца и значение
        printf "%-*s", $label_wid+1 , $label[$i]
        print " " . $ary[$i] if $ary[$i];
        print "\n";
    }

  ++$matches;
  }
  print "\n Для \"$arg\" соответствий не обнаружено \n" if $matches == 0;
}
```

Задачей сценария `show_member.pl` является отображение всего содержимого записи, независимо от количества ее полей. При использовании оператора `SELECT *` для выборки всех столбцов и атрибута `NAME` для определения, что они собой представляют, этот сценарий не потребует модификации, даже если в таблицу позднее будут добавляться новые или удаляться старые столбцы.

Для того чтобы просто узнать, какие столбцы содержит таблица, не выбирая при этом новых строк, необходимо сделать следующий запрос:

```
SELECT * FROM tbl_name WHERE 1 = 0
```

Вызвав методы `prepare()` и `execute()` обычным образом, можно получить имена столбцов из выражения `@{$sth->{NAME}}`. Однако следует помнить, что несмотря на то, что такие трюки с использованием "пустых" запросов работают для MySQL, они не всегда переносимы и работают не во всех базах данных.

Дополнительную информацию об использовании атрибутов интерфейса DBI и `DBD::mysql` можно найти в приложении Ж, "Программный интерфейс DBI языка написания сценариев Perl". Принятие решения о выборе золотой середины между переносимостью и атрибутами, присущими MySQL, полностью возлагается на разработчика.

Выполнение транзакций

Одним из способов выполнения транзакций в DBI-сценариях является задание операторов `SET AUTOCOMMIT`, `BEGIN`, `COMMIT` и `ROLLBACK` явным образом. (Эти операторы описаны в главе 3, "Синтаксис и использование языка SQL".) Однако интерфейс DBI имеет и свои собственные возможности для выполнения операций транзакций. Эти возможности выражаются в виде методов DBI, которые и берут на себя заботу о выпуске соответствующих операторов SQL, автоматически обеспечивающих вам транзакцию.

Эта возможность также переносима на другие поддерживающие работу транзакций базы данных, а операторы SQL могут и не быть переносимыми.

Для того чтобы воспользоваться механизмом транзакций интерфейса DBI, необходимо удовлетворить несколько требований.

- Модуль `DBD::mysql` должен иметь версию 1.2216 или выше.
- Ваш сервер MySQL должен использовать такие поддерживающие механизмы транзакции типы таблиц, как InnoDB или BDB. В главе 3, "Синтаксис и использование языка SQL", описано, как в этом можно удостовериться.
- Ваше приложение должно использовать типы таблиц, которые поддерживают транзакции. Если это не так, для изменения их типа воспользуйтесь оператором `ALTER TABLE`. Например, для того, чтобы изменить тип заданной таблицы *`tbl_name`* на `InnoDB`, можно воспользоваться следующим оператором:

  ```
  ALTER TABLE tbl_name TYPE = INNODB;
  ```

Если предположить, что эти условия удовлетворены, общая процедура обработки транзакций при работе с интерфейсом DBI будет следующей.

1. Отключить (или временно приостановить) режим `auto-commit`, чтобы операторы SQL не выполнялись до тех пор, пока вы не сделаете это сами.

2. Выполнить запросы, которые являются частью транзакции, но сделать это в пределах блока `eval`, который выполняется при активизированном параметре `RaiseError` и отключенном параметре `PrintError`, таким образом, что любая ошибка могла остановить выполнение блока без печати информации об ошибках. Если блок выполнятся успешно, последняя операция будет завершающей операцией выполняемой транзакции.
3. Когда блок `eval` завершается, проверьте состояние завершения. В случае возникновения ошибки вызовите метод `rollback()` для завершения транзакции и, если необходимо выведите сообщение об ошибке.
4. Восстановите режим `auto-commit` и соответствующие значения параметров обработки ошибок.

В следующем примере продемонстрировано, как можно реализовать такой подход. Он базируется на сценарии из главы 3, в котором показано, как задавать операторы, обеспечивающие транзакцию, вручную с клиента `mysql`. Если помните, это тот самый сценарий, в котором была обнаружена ошибка в оценках, выставленных двум учащимся в таблице `score`, и их потребовалось поменять. Учащемуся 8 была присвоена оценка 18, а учащемуся 9 — 13, и эти оценки необходимо поменять местами. Два оператора `UPDATE`, позволяющие исправить эту оценку, выглядят следующим образом:

```
UPDATE score SET score = 13 WHERE event_id = 5 AND student_id = 8;
UPDATE score SET score = 18 WHERE event_id = 5 AND student_id = 9;
```

Нам необходимо обновить обе записи правильными оценками, при этом оба обновления должны выполняться одним блоком. В предыдущей главе операторы обновления были окружены явными операторами SQL, которые осуществляли установку режима `auto-commit`, выполнения или отката транзакции. В сценарии Perl, который использует механизм транзакций интерфейса DBI, операторы обновления будут выполняться следующим образом:

```
my $orig_re = $dbh->{RaiseError}; # сохранить параметры обработки ошибок
my $orig_pe = $dbh->{PrintError};
my $orig_ac = $dbh->{AutoCommit}; # сохранить режим auto-commit
$dbh->{RaiseError} = 1; # ошибки вызывают исключительные ситуации
$dbh->{PrintError} = 0; # но подавляют вывод сообщений
$dbh->{AutoCommit} = 0; # транзакция не завершится, пока не поступит соответствующая команда
eval
{
  # операторы, которые являются частью транзакции
  my $sth = $dbh->prepare (qq{
            UPDATE score SET score = ?
            WHERE event_id = ? AND student_id = ?
        });
  $sth->execute (13, 5, 8);
  $sth->execute (18, 5, 9);
  $dbh->commit(); # выполнить транзакцию
};
if ($@) # транзакция не сработала?
{
  print "При выполнении транзакции возникла ошибка: $@\n";
```

```
    # сделать откат, пользуясь блоком eval для обнаружения ошибки отката
    eval { $dbh->rollback (); }
}
$dbh->{AutoCommit} = $orig_ac; # восстановить режим auto-commit
$dbh->{RaiseError} = $orig_re; # восстановить атрибуты обработки ошибок
$dbh->{PrintError} = $orig_pe;
```

Блок `eval` обрабатывает транзакции, и его статус завершения можно получить в переменной `$@`. При безошибочном выполнении операторов `UPDATE` функция `commit()` предназначается для выполнения транзакции, и значение `$@` будет пустым. При возникновении ошибки блок `eval` не выполняется, а `$@` содержит сообщение об ошибке. В этом случае программа распечатывает диагностическое сообщение и отменяет выполнение транзакции вызовом метода `rollback()`. (Операция отката размещается внутри собственного блока `eval` для того, чтобы предотвратить его остановку в случае сбоя.)

В этой главе в сценариях DBI обычно используется такой режим обработки ошибок, при котором атрибут `RaiseError` включен, а `PrintError` отключен. Это означает, что они уже имеют значения, необходимые для выполнения транзакций, таким образом, не нужно сохранять, устанавливать и восстанавливать эти атрибуты, как это было показано в предыдущем примере. Однако, если это сделать, будет обеспечена правильная работа сценария даже в ситуациях, когда вы заранее не уверены в том, какие значения параметров обработки ошибок установлены.

Подготовка интерфейса к работе

Итак, вы уже познакомились со многими концепциями, положенными в основу процесса программирования интерфейса DBI. Теперь перейдем в практическую плоскость. Наши цели были сформулированы в главе 1, "Знакомство с СУБД MySQL и SQL". Будем следовать им при написании сценариев DBI в этой главе.

В проекте учета успеваемости требуется обеспечить возможность выборки результатов любой викторины или экзамена.

В проекте "Историческая Лига" требуется обеспечить следующие функции.

- Распечатка списка членов лиги в различных форматах. Для создания программы ежегодного банкета нам нужен список членов (только имена) в формате, удобном для печати.
- Выборка членов лиги, которым необходимо срочно возобновить свое членство, и рассылка уведомлений им об этом.
- Редактирование записей. (Модификация даты истечения срока членства после возобновления членства.)
- Поиск членов, имеющих общие научные интересы.
- Размещение списка членов на Web-узле.

Для решения одних задач достаточно сценариев, запускающихся из командной строки. Для решения других задач в следующем разделе будут созданы сценарии, которые можно использовать одновременно с Web-сервером. Но все равно останутся нерешенные задачи. Все они будут решены окончательно в главе 8, "Интерфейс API для языка написания сценариев PHP".

Создание каталога общества "Историческая Лига"

Одной из наших целей является получение данных из списка "Историческая Лига" в различных форматах. Простейшим форматом, который будет нами создан, является список имен приглашенных на ежегодный банкет. Он будет включен в состав большого буклета. Значит, все, что нам нужно, — это некая форма, которую потом можно будет вставить в этот буклет.

Для печати каталога необходимо нечто более "изысканное", чем простой текст. Хорошим решением здесь может быть формат RTF (Rich Text Format), разработанный компанией Microsoft для обеспечения совместимости различных текстовых редакторов. К числу таких редакторов можно отнести редактор MS Word. Но этот формат будет понятен и таким текстовым редакторам, как WordPerfect и AppleWorks. Разные текстовые редакторы поддерживают различные подмножества RTF, но мы воспользуемся основным подмножеством спецификации формата RTF, который будет воспринят любым текстовым процессором, воспринимающим формат RTF даже в минимальном наборе. (Например, прочитать RTF-файл, полученный нами, здесь сможет текстовый редактор TextEdit ОС Mac OS X.)

Процедуры создания приглашений на банкет и каталога в формате RTF принципиально идентичны: генерация запроса поиска записей, затем запуск цикла выборки и форматирование каждой записи. При таком общем подобии лучше всего избегать написания двух отдельных сценариев. Для этого ограничимся написанием универсального сценария `gen_dir.pl`, который может генерировать каталог в различных форматах. Структура сценария будет такой.

1. Инициализация формата. Для каталога участников банкета никакой специальной инициализации не нужно, но необходимо создать некий предварительный код для версии, обрабатывающий формат RTF.
2. Выбрать и распечатать каждую запись, отформатированную под соответствующий тип вывода.
3. После того как все записи будут обработаны, выполнить все необходимые операции по очистке и завершить работу. И опять же, при обработке буклета для банкета такая обработка не нужна, она потребуется нам для RTF-версии.

В будущем этот сценарий можно будет использовать для вывода в других форматах, поэтому предусмотрим такую возможность с помощью расширяемой конструкции языка Perl `switchbox`. Элементы этой конструкции будут определять функцию для различных операций: функцию инициализации, функцию чтения записи из таблицы и функцию очистки.

```
# switchbox, содержащий функции форматирования для каждого формата
my (%switchbox) =
(
   "banquet" =>  # функции для каталога банкета
   {
     "init"    => undef,  # инициализация не требуется
     "entry"   =>\&format_banquet_entry
     "cleanup"  => undef  # очистка не требуется
   },
   "rtf" =>  # функции для обработки формата RTF
```

```
  {
    "init"    => \&rtf_init,
    "entry"   => \&format_rtf_entry,
    "cleanup"  => \&rtf_cleanup
  }
);
```

Элементы конструкции switchbox маркированы названием формата (в данном случае "banquet" и "rtf"). Сценарий позволяет определять нужный формат непосредственно в командной строке:

```
% ./gen_dir.pl banquet
% ./gen_dir.pl rtf
```

Таким образом, всегда можно без особого труда добавить новую возможность. Для этого достаточно выполнить следующее.

1. Написать три форматирующие функции.
2. Добавить в конструкцию switchbox новый элемент, который вызывает эти функции.
3. Для вывода в новом формате вызвать gen_dir.pl с указанием названия формата в командной строке.

Код выбора соответствующей ветви switchbox в зависимости от значения первого аргумента командной строки показан ниже. Это основано на том факте, что ветвление производится на базе названия формата в %switchbox. Если такого ключа в операторе switchbox нет, формат будет признан ошибочным. При этом не отпадает необходимость прописывать имена форматов прямо в коде, так как он определяется автоматически. Если нет названия формата или в командной строке указано ошибочное имя, сценарий выдает сообщение об ошибке и отображает перечень допустимых имен:

```
# проверка на присутствие в командной строке хотя бы одного аргумента
@ARGV == 1
  or die "Использование: gen_dir format_type\n Допустимые форматы: "
    . join (" ", sort (keys (%switchbox) ) ) . "\n";

# определение элемента switchbox, соответствующего аргументу,
# указанному в командной строке
# если не найдено, тип формата неверный
my ($func_hashref ) = $switchbox{$ARGV[0]};

defined ($func_hashref )
  or die "Неизвестный формат: $ARGV[0] \n Допустимые форматы:
    " . join (" ", sort (keys (%switchbox) ) ) . "\n";
```

Если в командной строке указан существующий формат, этот программный код устанавливает переменную $func_hashref. Это значение будет ссылкой на функции обработки выбранного формата. Затем можно запустить запрос выборки нужной записи. После этого вызывается функция инициализации, выборки и печати записей, затем — функция очистки памяти:

```
  # вызов функции инициализации
  &{$func_hashref ->{init}} if defined ($func_hashref ->{init});

# выборка и печать записей, если есть функция форматирования
```

```
if (defined ($func_hashref ->{entry}) )
{
   my ($sth) = $dbh->prepare (qq{
   SELECT * FROM member ORDER BY last_name, first_name
   });
   $sth->execute ();

   while (my $entry_ref = $sth->fetchrow_hashref ("NAME_lc"))
   {
     # передача ссылки на функцию форматирования
     &{$func_hashref ->{entry}} ($entry_ref ) ;
   }
}
```

Метод `fetchrow_hashref()` в цикле выборки используется для того, чтобы функции форматирования имели информацию о порядке следования столбцов. Его можно узнать из атрибута `$sth->{NAME}`, содержащего имена столбцов в порядке, в котором они возвращаются, но зачем это делать? По ссылке функции форматирования могут определить имена нужных столбцов с помощью `$entry_ref->{`*`col_name`*`}`. Это не совсем эффективный, но достаточно простой прием, которым можно пользоваться для любого генерируемого формата, потому что достаточно хорошо известно, какие поля данных нам потребуются.

Теперь только и остается, что написать функции для каждого формата вывода (т.е. функции, указанные в элементах конструкции `switchbox`).

Генерация списка приглашенных на банкет

Для этого формата нам необходимы имена участников, инициализация и очистка памяти не нужны. Потребуется только функция форматирования записи `format_banquet_entry()`:

```
sub format_banquet_entry
{

   # Здесь выводится имя, фамилия и суффикс члена лиги
}
```

Аргументом `format_banquet_entry()` является ссылка на данные в столбцах строки. Эта функция "склеивает" имена и фамилии и добавляет суффикс, если таковой имеется. Особенность этой проблемы заключается в том, что суффиксам, подобным "`Jr.`" или "`Sr.`", должны предшествовать запятая и пробел, в то время как суффиксам, подобным "`II`" или "`III`", должны предшествовать только пробелы:

```
Michael Alvis IV
Clarence Elgar, Jr.
Bill Matthews, Sr.
Mark York II
```

Поскольку букв "I", "V" и "X" достаточно для отображения всех поколений от 1-го до 39-го, определить, надо или не надо добавлять запятую, можно при сравнении со следующим шаблоном:

```
/^[IVX]+$/;
```

Код подпрограммы `format_banquet_entry()`, выполняющий "склеивание" имен и фамилий, нам также пригодится для формирования RTF-версии нашего

каталога. Однако вместо простого дублирования этого кода в функцию format_rtf_entry() преобразуем этот код в функцию:

```
sub_format_name
{
my ($entry_ref) = shift;

   $name = $entry_ref->{first_name} . " " . $entry_ref->{last_name};
   if (defined ($entry_ref->{suffix}))       # вот суффикс имени
   {
     # без запятых в суффиксах типа I, II, III и т.д.
     $name .= "," unless $entry_ref->{suffix} =- /^[IVX]+$/;
     $name .= " " . $entry_ref->{suffix}
   }
   return ($name);
}
```

Перенос в функцию format_name() кода, преобразующего строку имени, сводит функцию format_banquet_entry() к одной строке:

```
sub format_banquet_entry
{
   printf "%s\n", format_name ($_[0]);
}
```

Генерация каталога в формате, пригодном для печати

Создание RTF-версии каталога — задача несколько более сложная, чем создание списка участников банкета. С одной стороны, необходимо получить больше информации из каждой записи. С другой стороны, для того, чтобы достичь желаемого результата, в формате RTF каждую запись необходимо сопровождать управляющим кодом. Минимальный управляющий код, необходимый для управления в RTF-формате, выглядит следующим образом:

```
{\rtf0
{\fonttbl {\f0 Times;}}
\plain \f0 \fs24
   ...здесь идет содержание документа...
}
```

Документ начинается и заканчивается фигурными скобками, { и }. Ключевые слова формата RTF начинаются с обратной косой черты. Таким образом, первым ключевым словом документа должно быть \rtfn, где *n* — это спецификация номера версии формата RTF. Для наших целей вполне подходит версия 0.

Таблица шрифтов определяется внутри документа. Эта таблица будет использоваться при отображении документа. Информация о таблице шрифтов приводится в группе символов, начинающейся с фигурных скобок, включающих ключевое слово \fonttbl и информацию о штифте. Таблица шрифтов, приведенная в данных набросках кода, определяет шрифт Times номер 0. (Здесь можно задать сразу несколько штифтов.)

Несколько директив, следующих за этим, определяют стиль форматирования по умолчанию. Так, \plain выбирает простой формат, \f0 выбирает шрифт 0 (который был определен нами в таблице шрифтов как Times), а \fs24 устанавливает размер шрифта 12 точек (число, следующее за \f, обозначает размер в полу-

точках). Совсем не обязательно устанавливать границы, так как большинство текстовых процессоров имеет приемлемые значения по умолчанию.

Основной код обрамляет функции инициализации и очистки, которые выглядят следующим образом (двойные обратные косые при выводе преобразуются в одинарную обратную косую черту):

```
sub rtf_init
{
   print "{\\rtf0\n";
   print "{\\fonttbl {\\f0 Times;}}\n";
   print "\\plain \\f0 \\fs24\n"
}
sub rtf_cleanup
{
   print "}\n";
}
```

Реализуем простой подход, при котором каждая строка будет распечатана в виде последовательности строк с меткой в каждой строке. Если информация, соответствующая определенной выводимой строке, отсутствует, строка не печатается. (Например, строка "Email:" не будет распечатываться для членов, не имеющих электронного адреса.) Есть и такие строки (как, скажем, строка "Address:"), которые являются составными и состоят из информации, содержащейся сразу в нескольких столбцах (street, city, state, zip). Отсюда видно, что сценарий должен уметь работать с совокупностью недостающих значений. Вот пример формата вывода данных, которым мы здесь воспользуемся:

```
Name: Mike Artel
Address: 4264 Lovering Rd., Miami, FL 12777
Telephone: 075-961-0712
Email: mike_artel@venus.org
Interests: Civil Rights,Education,Revolutionary War
```

Представление в RTF-формате будет иметь вид

```
\b Name: Mike Artel\b0\par
Address: 4264 Lovering Rd., Miami, FL I2777\par
Telephone: 075-961-07!2\par
Email: mike_artel@venus.org\par
Interests: Civil Rights,Education,Revolutionary War\par
```

Для того чтобы строка "Name:" распечатывалась жирным шрифтом, ее необходимо предварить кодом "\b" (с последующим пробелом) для включения жирного шрифта и "\b0" – для отключения жирного шрифта. Строка члена форматируется функцией format_name(), показанной в предыдущем разделе. Каждая строка в конце снабжена маркером абзаца (\par) для того, чтобы текстовый редактор перешел к следующей строке. Основные сложности заключаются в форматировании строки, в которой указывается адрес:

```
sub format_rtf_entry
{
my ($entry_ref) = shift;
   printf "\\b Name: %s\\b0\\par\n", format_name ($entry_ref);
   my $address = "";
   $address .= $entry_ref->{street}
            if defined ($entry_ref->{street});
```

```
    $address .= ", " . $entry_ref->{city}
             if defined ($entry_ref->{city});
    $address .= ", " . $entry_ref->{state}
             if defined ($entry_ref->{state});
    $address .= " " . $entry_ref->{zip}
             if defined ($entry_ref->{zip});
    print "Address: $address\\par\n"
    if $address ne "";
    print "Telephone: $entry_ref->{phone}\\par\n"
             if defined ($entry_ref->{phone});
    print "Email: $entry_ref->{email}\\par\n"
             if defined ($entry_ref->{email});
    print "Interests: $entry_ref->{interests}\\par\n"
             if defined ($entry_ref->{interests});
    print "\\par\n";
}
```

Вы, конечно, не ограничены определенным стилем форматирования и можете вносить свои изменения, модифицировав `format_rtf_entry()`. Такую операцию осуществить не так уж просто с самим документом (документ в текстовом редакторе)!

Теперь сценарий `gen_dir.pl` можно считать завершенным. Каталог можно распечатать в любом из форматов, просто запустив его из командной строки:

```
% ./gen_dir.pl banquet > names.txt
% ./gen_dir.pl rtf > directory.rtf
```

С этого места очень просто считывать список имен и вставлять его в документ, содержащий программу ежегодного банкета, в любом текстовом процессоре, который "понимает" формат RTF.

Интерфейс DBI упрощает процесс получения необходимой информации из СУБД MySQL, а возможности обработки текстов, имеющиеся в языке написания сценариев, легко позволяют преобразовать информацию в нужный нам формат. СУБД MySQL не обладает такими механизмами форматирования выводимой информации. Но, принимая во внимание ту легкость, с которой можно интегрировать возможности обработки данных в СУБД MySQL с возможностями такого языка программирования, как Perl, это уже не так и важно.

Рассылка напоминаний об истечении срока членства

Ведение списка "Исторической Лиги" в виде текстового документа требует много времени и несет в себе большую вероятность ошибок, кроме того, автоматизация процесса рассылки предупреждений в этом случае будет чрезвычайно затруднена. Теперь, когда информация находится в базе данных, посмотрим, каким образом можно автоматизировать этот процесс и рассылать напоминания по электронной почте.

Главной задачей при этом является определение тех членов лиги, которым требуется такое обновление. Для создания такого запроса требуется относительно несложное вычисление относительной даты:

```
SELECT ... FROM member
WHERE expiration < DATE_ADD(CURRENT_DATE, INTERVAL интервал DAY)
```

Параметр *интервал* указывает количество дней гарантированной отсрочки. Данный запрос выбирает перечень членов, которым настало время возобновить членство. В частном случае при значении интервала, равном 0, этот запрос нахо-

дит членов, дата истечения членства которых уже прошла (т.е. срок членства которых истек).

Что нам делать с обнаруженными записями? В качестве одного из вариантов можно предложить послать электронную почту непосредственно из такого сценария, но, может быть, уместно сначала просмотреть полученный список, не посылая никаких сообщений вообще? Для этого воспользуемся процедурой, состоящей из двух этапов.

- **Этап 1.** Запустить сценарий `need_renewal.pl` для определения членов, которым необходимо возобновить членство. Полученный список можно изучить или использовать в качестве источника данных для второго этапа рассылки предупреждений.
- **Этап 2.** Запустить сценарий `renewal_notify.pl`, рассылающий сообщение "пожалуйста, возобновите" по электронной почте. Этот сценарий обязательно должен предупредить о людях, не имеющих электронной почты, с которыми нужно будет связаться с помощью других средств.

На первом этапе данной задачи сценарий `need_renewal.pl` определяет, какому члену нужно такое обновление. Это можно сделать следующим образом.

```
# по умолчанию интервал составляет 30 дней;
my $cutoff = 30;
# изменить в случае получения числового аргумента
$cutoff = shift (@ARGV) if @ARGV && $ARGV[0] =~ /^\d+$/;

warn "интервал составляет $cutoff дней \n";

my ($sth) = $dbh->prepare (qq{
  SELECT
    member_id, email, last_name, first_name, expiration,
    TO_DAYS(expiration) - TO_DAYS(CURRENT_DATE) AS days
  FROM member
  WHERE expiration < DATE_ADD(CURRENT_DATE, INTERVAL ? DAY)
  ORDER BY expiration, last_name, first_name
});
$sth->execute ($cutoff); # передача интервала в виде заглушки

while (my $hash_ref = $sth->fetchrow_hashref ())
  # преобразовать для печати значения undef в пустые строки
  foreach my $key (keys (%{$entry_ref}))
  {
    $entry_ref->{$key} = "" if !defined ($entry_ref->{$key});
  }
  print join ("\t",
              $hash_ref->{member_id},
              $hash_ref->{email},
              $hash_ref->{last_name},
              $hash_ref->{first_name},
              $hash_ref->{expiration},
              $hash_ref->{days} . " дней")
    . "\n";
}
```

Вывод, получаемый из сценария `need_renewal.pl`, имеет следующий вид (вы получите другой результат, так как он определяется относительно текущей даты, которая, конечно же, должна отличаться):

```
89 g.steve@pluto.com       Garner Steve 2002-08-03   -32 days
18 york_mark@earth.com          York      Mark    2002-08-24   -11 days
82 john_edwards@venus.org Edwards  John    2002-09-12   -8 days
```

Обратите внимание, что некоторые члены должны возобновить свое членство через отрицательное количество дней. Это значит, что срок их членства уже истек! (Такое довольно часто случается при ведении данных вручную. Теперь, после того, как учет таких членов был автоматизирован, мы увидели, что кого-то пропустили.)

Вторая часть напоминания о возобновлении членства вызывает сценарий `renewal_notify.pl`, функции которого заключаются в высылке этих напоминаний по электронной почте. Для того чтобы немного облегчить использование сценария `renewal_notify.pl`, необходимо, чтобы он распознавал три типа аргументов командной строки: идентификационные номера, электронные адреса и имена файлов. Цифровые значения обычно являются идентификационными номерами, а аргументы, содержащие символ "@", можно смело отнести к категории электронных адресов. Любое другое значение интерпретируется как имя файла, который необходимо прочесть для получения идентификационных номеров или адреса электронной почты. Таким образом, параметры можно передавать из командной строки или файла. (В частности, можно использовать вывод сценария `need_notify.pl` как ввод сценария `renewal_notify.pl`.)

Сценарий выбирает электронные адреса каждого члена, которому будет разослано напоминание, и рассылает сообщения по этим адресам. Если электронный адрес отсутствует, сценарий `renewal_notify.pl` генерирует диагностическое сообщение, предупреждающее о том, что необходимо связаться с этими людьми каким-то другим образом.

Основной цикл обработки аргументов работает следующим образом. В случае отсутствия аргументов в командной строке входные данные считываются со стандартного устройства ввода. В противном случае аргументы передаются для анализа в функцию `interpret_argument()` для распознавания идентификационного номера, электронного адреса или файла:

```
if (@ARGV == 0)         # аргументы отсутствуют, ожидать ввода
                        # с устройства STDIN
{
  read_file (\*STDIN);
}
else
{
  while (my $arg = shift (@ARGV))
  {
    # интерпретация аргумента с рекурсией по имени файла
    interpret_argument ($arg, 1);
  }
}
```

Функция `read_file()` считывает содержимое файла (предполагается, что они уже открыты) и просматривает первое поле в каждой строке. (При считывании вывода сценария `need_renewal.pl` в качестве ввода `renewal_notify.pl` каждая строка содержит несколько полей, но нам достаточно просмотреть только одну строку.)

```
sub read_file
{
my ($fh) = shift; # дескриптор открытия файла
```

```
my $arg;

  while (defined ($arg = <$fh>))
  {
    # отсекаем все данные, следующие после первого столбца,
    # включая новую строку
    $arg =~ s/\s.*//s;
    # интерпретировать аргумент без рекурсии имени файла
    interpret_argument ($arg, 0);
  }
}
```

Функция `interpret_argument()` определяет, является ли аргумент идентификационным номером, электронным адресом или именем файла. В случае электронного адреса или имени файла эта функция находит соответствующую запись и передает полученные данные функции `notify_member()`. С электронными адресами нужно быть очень осторожными. Вполне возможно, что два члена лиги имеют один и тот же адрес (например, так бывает, если они муж и жена). При этом необходимо следить, чтобы сообщение не попало тому, кому оно не предназначено. Для того чтобы этого избежать, необходимо проверить существование взаимной однозначности между идентификационным номером и электронным адресом. Если адрес соответствует нескольким идентификационным адресам, он игнорируется и печатается предупреждение.

Если аргумент не похож на идентификационный номер или электронный адрес, он принимается за имя файла, который необходимо прочитать для дальнейшего ввода. И здесь необходимо быть предельно осторожными, чтобы избежать опасности неопределенного цикла, так как читать файл повторно не надо:

```
sub interpret_argument
{
my ($arg, $recurse) = @_;

  if ($arg =~ /"\d+$/)   # цифровой идентификационный номер
  {
    notify_member ($arg);
  }
  elseif  ($arg =~ /@/)  # электронный адрес
  {
    # получить идентификационный номер, связанный с адресом
    # (должен быть только один)
    my $query = qq{ SELECT member_id FROM member WHERE email = ? };
    my $ary_ref = $dbh->selectcol_arrayref ($query, undef, $arg);
    if (scalar (@{$ary_ref }) == 0)
    {
        warn "Не найдено соответствия для электронного адреса
                    $arg: проигнорирован \n";
    }
    elseif (scalar (@{$ary_ref }) > 1)
    {
        warn "Электронный адрес $arg соответствует нескольким
                    записям: проигнорирован \n";
    }
    else
    {
        notify_member ($ary_ref ->[0] ) ;
```

```
    }
  }
  else  # имя файла
  {
    if (!$recurse)
    {
        warn "имя файла $arg в файле: проигнорировано \n";
    }
    else
    {
        open (IN, $arg) or die "Нельзя открыть $arg: $!\n";
        read_file (\*IN);
        close (IN);
    }
  }
}
```

Программный код функции `notify_member()`, рассылающей сообщения о возобновлении членства, выполняет следующие функции. Если обнаруживается, что член общества не имеет электронного адреса, то не остается ничего другого, как распечатать с помощью функции `notify_member()` предупреждение с уведомлением о том, что с ним надо будет связаться каким-либо другим способом. Можно вызвать сценарий `show_member.pl`, показывающий в сообщении идентификационный номер, чтобы отобразить всю запись, — прежде всего для определения телефонного номера и адреса, например:

```
sub notify_member
{
my $member_id = shift;

  warn "Ставим в известность $member_id. . . \n" ;
  my $query = qq{ SELECT * FROM member WHERE member_id = ? };
  my $sth = $dbh->prepare ($query);
  $sth->execute ($member_id);
  my @col_name = @{$sth->{NAME}};
  my $entry_ref = $sth->fetchrow_hashref ();
  $sth->finish ();
  if (!$entry_ref) # Член не найден!
  {
    warn "Не найдена ЗАПИСЬ по члену $member_id! \n";
    return;
  }
  if (!defined ($entry_ref->{email})) # номер не найден!
  {
    warn "Член $member_id не имеет электронной почты; сообщение не
отослано \n";
    return;
  }
  open (OUT, "| $sendmail") or die "Программа рассылки
                                    недоступна \n";
  print OUT <<EOF;
To: $entry_ref ->{email}
Subject: Ваше членство в "Исторической Лиге" требует обновления.
Здравствуйте. Срок вашего членства в "Исторической Лиге" скоро
истекает. Мы надеемся, что вы сможете найти несколько
минут, чтобы связаться с нашим офисом и
продлить свое членство. Подробная информация приведена
```

```
внизу. Пожалуйста, обратите внимание на дату истечения
срока вашего членства.

Спасибо.

EOF
   foreach my $col_name (@col_name)
   {
     printf OUT "%s: %s\n", $col_name, $entry_ref->{$col_name};
     printf OUT " $entry_ref->{$col_name}"
     if defined ($entry_ref->{$col_name});
     printf OUT "\n";
   }
   close (OUT);
}
```

Для посылки электронной почты сценарий `notify_member()` открывает канал с программой `sendmail` и направляет сообщение по этому каналу. Путь к каталогу, в котором хранится программа `sendmail`, задается в начале сценария `renewal_notify.pl`. В зависимости от типа системы его может понадобиться изменить:

```
# изменение пути
my $sendmail = "/usr/lib/sendmail -t -oi";
```

При отсутствии программы `sendmail` этот сценарий будет работать неправильно. (Например, ОС Windows не имеет такой программы.) Для такого случая в дистрибутиве `sampdb` содержится модифицированная версия `renewal_notify.pl`, в которой есть модуль `Mail::Sendmail`, работающий без программы `sendmail`.

Эту процедуру можно усовершенствовать. Добавим, например, столбец, который будет содержать дату отсылки самого последнего напоминания. Это поможет не посылать такие предупреждения слишком часто. Самый подходящий интервал такой рассылки — один месяц.

Теперь оба сценария можно считать законченными. Они запускаются на выполнение следующим образом:

```
% ./need_renewal.pl > tmp
```

Посмотрим, как выглядит оповещение в файле `tmp`, чтобы убедиться в том, что сценарий работает. Если это так, воспользуемся им в качестве вводного файла сценария `renewal_notify.pl` для рассылки сообщений об обновлении членства:

```
% ./renewal_notify.pl tmp
```

Для того чтобы оповестить отдельных членов, нужно обозначить их, указав соответствующий идентификационный номер или электронный адрес:

```
% /.renewal_notify 18 g.steve@pluto.com
```

Редактирование записей о членах "Исторической Лиги"

Рассылка сообщений о необходимости обновления членства началась. Теперь необходимо учесть тот факт, что некоторые из членов "Исторической Лиги", которых мы предупредили, возобновили свое членство в этой организации. Когда это происходит, необходимо получить возможность обновлять их записи и мо-

дифицировать даты завершения их срока членства. В следующей главе будет показано, как это можно сделать из Web-броузера. А в этой главе будет написан сценарий `edit_member.pl`, работающий из командной строки и позволяющий модифицировать записи с помощью простого последовательного ввода нового значения для всех частей записи. Он работает следующим образом.

- Когда сценарий `edit_member.pl` запускается без аргументов, предполагается, что пользователь хочет ввести данные о новом члене "Исторической Лиги". В результате будет создана новая запись.
- Когда сценарий `edit_member.pl` вызывается из командной строки с определенным идентификационным номером, осуществляются просмотр существующих компонентов записи и запрос новых данных по каждому столбцу. Введенное значение обновляет текущее значение. После нажатия клавиши <Enter> содержимое столбца не изменится. Ввод слова none вызывает сброс текущего значения столбца. (Если идентификационный номер нам не известен, для его определения можно вызвать сценарий `show_member.pl` *`last_name`*.)

Вероятно, это усложнит работу по изменению одной лишь даты истечения срока членства. Но сценарий, аналогичный этому, позволяет модифицировать любую часть записи, не требуя при этом особого знания языка SQL. (Особо стоит отметить, что сценарий `edit_member.pl` не позволяет вносить изменения в поле `member_id`. Это является следствием того, что оно создается автоматически и его значение после этого менять нельзя.)

Прежде всего, сценарий `edit_member.pl` должен "знать" имена столбцов в таблице `member`:

```
# получить имена столбцов таблицы member
my $sth = $dbh->prepare (qq{ SELECT * FROM member WHERE 0 });
$sth->execute ();
my @col_name = @{$sth->{NAME}};
$sth->finish ();
```

После этого перейдем к главному циклу.

```
if (@ARGV == 0) # аргументы отсутствуют — создать новую запись
{
  # передать ссылку на массив имен столбцов
  new_member (\@col_name);
}
else      # иначе редактировать записи, рассматривая аргументы
          # в качестве идентификаторов членов "Исторической Лиги"
{
  # сохранить @ARGV и затем очистить его, чтобы при чтении
  # с устройства STDIN
  # не использовал их аргументы в качестве имен файлов
  my @id = @ARGV;
  @ARGV = ();
  # найти запись для определенного идентификатора и
  # отредактировать ее
  while (my $id = shift (@id))
  {
    $sth = $dbh->prepare (qq{
        SELECT * FROM member WHERE member_id = ?
            });
```

```
    $sth->execute($id);
    my $entry_ref = $sth->fetchrow_hashref ();
    $sth->finish ();
    if (!$entry_ref)
    {
        warn Член "Исторической Лиги" с номером = $id
             не найден \n";
        next;
    }
    # передать ссылку на массив имен столбцов и ссылку на запись
    edit_member (\@col_name, $entry_ref);
  }
}
```

Программный код для создания новой записи члена "Исторической Лиги" приведен ниже. Он запрашивает новые значения для всех столбцов, а затем генерирует оператор INSERT для того, чтобы добавить новую запись.

```
sub new_member
{
my ($col_name_ref) = shift; # ссылка на массив имен столбцов
my ($entry_ref); # создать новую запись

  return unless prompt ("Создать новую запись? ") =~ /^y/i;
  # подсказка для получения новых значений;
  # пользователь вводит новое значение,
  # для того чтобы ввести пустое значение,
  # введите "none", для преждевременного выхода введите "exit"
  foreach my $col_name (@{$col_name_ref})
  {
    next if $col_name eq "member_id";  # пропустить ключевое поле
    my $col_val = col_prompt ($col_name, undef);
    next if $col_val eq ""; # пользователь нажал клавишу <Enter>
    return if $col_val eq lc /"exit"/i;   # преждевременный выход
    $col_val = undef if $col_val eq lc /"none"/i;
    $entry_ref->{$col_name} = $col_val;
  }
  # показать значения, перед вставкой запросить подтверждение
  show_member ($col_name_ref, $entry_ref);
  return unless prompt ("\nВставить эту запись? ") =~ /^y/i;
  # создать запрос INSERT, а затем запустить его.
  my $query = "INSERT INTO member";
  my $delim = " SET "; # перед первым столбцом вставить "SET",
                       # перед другими — ","
  foreach my $col_name (@{$col_name_ref})
  {
    # определить заданные значения столбцов
    next if !defined ($entry_ref->{$col_name});
    # функция quote() при получении значения undef выводит
    # слово NULL (без кавычек),
    $query .= sprintf ("%s %s=%s", $delim, $col_name,
                $dbh->quote ($entry_ref->{$col_name}));
    $delim = ",";
  }
  $dbh->do ($query) or warn "Предупреждение: запись не введена? \n"
}
```

Подпрограмма вывода подсказок, которые использует `new_member()`, выглядит следующим образом:

```
sub prompt
{
my $str = shift;
   print STDERR $str;
   chomp ($str = <STDIN>);
   return ($str);
}
```

Функция `col_prompt()` принимает в качестве аргумента имя столбца. Он выводит имя как подсказку для ввода нового значения столбца и возвращает значение, введенное пользователем:

```
sub col_prompt
{
my ($col_name, $entry_ref) = @_;
   my $prompt = $col_name;
   if (defined ($entry_ref))
   {
     my $cur_val = $entry_ref->{$col_name};
     $cur_val = "NULL" if !defined ($cur_val);
     $prompt .= " [$cur_val] ";
   }
   $prompt .= ": ";
   print STDERR $prompt;
   my $str = <STDIN>;
   chomp ($str);
   return ($str);
}
```

Второй аргумент `col_prompt()` является ссылкой на хэш, хранящий запись данного члена. Для того чтобы создать новую запись, это значение должно быть `undef`, но при редактировании существующих записей он будет указывать на *текущее содержимое записи. В таком случае* `col_prompt()` *включает текущее значение* столбца, которое находится в подсказке, так, что пользователь сможет увидеть, что он из себя представляет. Пользователь может принять значение, нажав клавишу <Enter>.

Программный код редактирования записи данных существующего члена аналогичен коду создания записи о новом члене. Однако в этом случае мы имеем дело с записью, которая уже существует, и подсказка отображает текущее состояние записи, а метод `edit_member()` выдает запрос `UPDATE`, а не `INSERT`:

```
sub edit_member
{
# ссылка на массив имен столбцов
my ($col_name_ref , $entry_ref) = @_;
   # отобразить текущее значение, запросить подтверждение
   # на дальнейшее выполнение и редактирование
   show_member ($col_name_ref , $entry_ref);
   return unless prompt ("\n Редактировать эту запись? ") =~ /^y/i;
   # подсказка для ввода новых данных;
   # пользователь вводит новое значение для замещения
   # существующего значения, "none" — для ввода пустого значения,.
   # "exit" — для того, чтобы выйти, или нажать
   # клавишу <Enter> для принятия текущего значения.
   foreach my $col_name (@{$col_name_ref })
   {
     next if $col_name eq "member_id";  # пропустить ключевое поле
```

```
    my $col_val = $entry_ref ->{$col_name};

    my $col_val = col_prompt ($col_name, $entry_ref);
    next if $col_val eq ""; # пользователь нажал <Enter>
    return if $col_val eq lc ("exit"); # преждевременный выход
    $col_val = undef if $col_val eq lc ("none");
    $entry_ref->{$col_name} = $col_val;
  }
  # перед модификацией отобразить новые значения и
  # запросить подтверждение
  show_member ($col_name_ref, $entry_ref);
  return unless prompt ("\nМодифицировать эту запись? ") =~ /^y/i;

  # создать запрос UPDATE, а затем сделать его
  my $query = "UPDATE member";
  my $delim = " SET "; # перед первым столбцом поставить "SET",
                       # перед другими — ","
  foreach my $col_name (@{$col_name_ref})

{
    next if $col_name eq "member_id";       # пропустить ключевое
                                            # поле
    # quote() преобразует значение undef в слово NULL (без
    #                                                  кавычек),
    # это то, что нам нужно.
    $query .= sprintf  ("%s %s=%s", $delim, $col_name,
            $dbh->quote  ($entry_ref->{$col_name}));
    $delim = ",";
  }
  $query .= " WHERE member_id = " . $dbh->quote ($entry_ref-
>{member_id});
  $dbh->do ($query) warn "Предупреждение: запись не модифицирована? \n"
}
```

Особенность использования сценария `edit_member.pl` заключается в том, что он не выполняет никакой проверки введенных значений. Для большинства полей таблицы `member` не надо ничего проверять — это просто строковые поля. Но для таких столбцов, как дата истечения срока членства, проверка необходима. Это нужно хотя бы для того, чтобы данные, заносимые в эти поля, напоминали собой даты. В прикладных программах, осуществляющих ввод данных общего назначения, предварительно можно получить данные о структуре таблицы, чтобы на основании этой информации определить типы всех ее столбцов. На основании этих данных определяется область существования данных. Это слишком сложная тема, в которую мы не будем углубляться, просто добавим в функцию `col_prompt()` код проверки формата данных при вводе данных в столбец `expiration`. Простейшая проверка даты может быть сделана следующим образом:

```
sub col_prompt
{
my ($col_name, $entry_ref) = @_;
loop:
  my $prompt = $col_name;
  if (defined ($entry_ref))
  {
    my $cur_val = $entry_ref->{$col_name};
    $cur_val = "NULL" if !defined ($cur_val);
```

```
    $prompt .= " [$cur_val] ";
  }
  $prompt .= " ";
  print STDERR $prompt;
  my $str = <STDIN>;
  chomp ($str);
  # сделать элементарную проверку даты истечения срока членства
  if ($str && $col_name eq "expiration" # проверить формат даты
                                        # истечения срока членства
  {
    if ($str !~ /^\d+\D\d+\D\d+$/)
    {
        warn "$str не является действительной датой, попробуйте еще \n"
        goto loop;
    }
  }
  return ($str);
}
```

Шаблон проверяет три цифровые последовательности, разделенные нецифровыми символами. Но это будет неполная проверка, которая не учитывает такие ошибки в формате даты, как "1999-14-92". Усовершенствование сценария подразумевает более полную проверку адекватности дат и проверку на наличие пробелов в имени и фамилии.

В числе других усовершенствований можно назвать отказ от операции модификации записи, если столбцы не были изменены, или выдавать предупреждение пользователю о том, что пока он редактировал запись, кто-то другой уже внес в нее изменения. Это можно сделать, сохранив первоначальные данные, а затем запрограммировав оператор UPDATE таким образом, чтобы модифицировать только те столбцы записи, которые подвергались изменениям. Если таковых нет, оператор даже не сработает. Кроме того, предложение WHERE можно написать с выражением AND *col_name = col_val* для значения каждого столбца. Это вызовет сбой при выполнении оператора UPDATE, если кто-то еще вносил изменения в запись. Это и создаст обратную связь, сигнализирующую о том, что два пользователя пытаются одновременно внести изменения в одну и ту же запись.

Сценарий `edit_member.pl` имеет еще один недостаток, который, возможно, придется устранять потом. Как видно, сценарий открывает соединение с базой данных до выполнения цикла подсказки и не закрывает его до того, как не сделает запись после завершения цикла. Если пользователю потребуется достаточно длительное время для ввода или модификации записи или происходит что-либо еще, соединение может так и остаться открытым на долгое время. Как можно изменить сценарий `edit_member.pl`, чтобы он держал соединение открытым столько времени, сколько это необходимо?

Подбор членов "Исторической Лиги" по общности интересов

В круг обязанностей секретаря "Исторической Лиги" входит и обработка запросов, поступающих от членов организации по поиску других членов, имеющих аналогичные научные интересы в области истории США. При работе с текстовым редактором нетрудно найти таких членов, область научных интересов кото-

рых — период Великой Депрессии или биография Авраама Линкольна. Но создать список, удовлетворяющий этим критериям, будет уже довольно затруднительно, и для решения такой задачи будет удобнее воспользоваться возможностями СУБД MySQL и сделать такой запрос:

```
SELECT * FROM member WHERE interests LIKE "%lincoln%"
ORDER BY last_name, first_name
```

К сожалению, результаты этого запроса будут иметь неприглядный вид, если запускать его непосредственно из клиентской программы `mysql`. Попробуем усовершенствовать вывод из сценария DBI, выполняющего задачу подбора по интересам. Сначала сценарий проверяет наличие в командной строке хотя бы одного аргумента, так как в противном случае ему вообще не надо ничего делать. Затем сценарий выполняет поиск по столбцу `interests` таблицы `member`:

```
@ARGV or die "Задача: ключевое слово interests.pl \n";
search_members (shift (@ARGV)) while @ARGV;
```

Добавим с обеих сторон к искомой строке символ "%". Это позволит найти строку, являющуюся подстрокой столбца `interests`. После этого распечатаем найденные записи.

```
sub search_members
{
my $interest = shift;
   print "Поиск результатов для ключевого слова: $interest\n\n";
   my $sth = $dbh->prepare (qq{
         SELECT * FROM member WHERE interests LIKE ?
         ORDER BY last_name, first_name
     });
   # поиск строки в любом месте поля interest
   $sth->execute ("%" . $interest . "%");
   my $count = 0;
   while (my $hash_ref = $sth->fetchrow_hashref ())
   {
     format_entry ($hash_ref);
     ++$count;
   }
   print "Найдено $count записей \n\n";
}
```

Здесь не показана функция `format_entry()`, преобразующая запись в ее печатное представление. Она во многом подобна функции `format_rtf_entry()` из сценария `gen_dir.pl`, но без команд управления RTF-формата. Как все это выглядит, можно посмотреть в сценарии `interests.pl` дистрибутива `sampdb`.

Размещение каталога "Исторической Лиги" на Web-узле

В разделе "Работа с интерфейсом DBI в Web-приложениях" мы приступим к написанию сценариев, подключающихся к серверу MySQL для того, чтобы выбирать и записывать информацию в виде Web-страниц с помощью Web-броузера клиента. Эти сценарии динамически генерируют данные в формате HTML в зависимости от запроса клиента. Но до этого попрактикуемся в написании статических HTML-документов, которые будут загружены в дерево документов Web-сервера. Самым

подходящим "кандидатом" является задача создания в формате HTML каталога "Исторической Лиги".

Простейший документ в формате HTML имеет такую структуру:

```
<html>                    <-- начало документа
<head>                    <-- начало заголовка документа
<title> Титульная страница </title>   <-- титул документа
</head>                   <-- конец заголовка документа
<body>                    <-- начало тела документа
<body bgcolor="white"> < начало тела документа
(белый фон)
<h1> Заголовок 1-го уровня </m>   <- заголовок 1-го уровня
... тело документа ...
</body>                   <-- конец тела документа
</html>                   <-- конец документа
```

Для генерации каталога в этом формате нет необходимости писать весь сценарий. Когда писался сценарий `gen_dir.pl`, мы воспользовались расширяемой оболочкой, что дало нам возможность вставлять программный код для печати каталога в дополнительных форматах. Воспользуемся опять преимуществом этой расширяемости, добавив программный код для создания вывода на языкс HTML. Для этого нам потребуется внести в сценарий `gen_dir.pl` следующие изменения:

- написать программный код инициализации документа, функции очистки;
- написать функции форматирования отдельных записей;
- добавить конструкцию `switchbox`, которая идентифицирует имя формата и связывает его с функциями, осуществляющими вывод в этом формате.

Сам документ легко можно разбить на начало и окончание, в которых будут располагаться функции инициализации и очистки, и на основную часть, в которой располагается функция форматирования записей. Функция инициализации формата HTML генерирует управляющий код, находящийся выше заголовка 1-го уровня, а функция очистки генерирует тэги закрытия `</body>` и `</html>`.

```
sub html_init
{
   print "<html>\n";
   print "<head>\n";
   print "<title> Список членов "Исторической Лиги" </title>\n";
   print "</head>\n";
   print "<body bgcolor=\"white\">\n";
   print "<body>\n";
   print "<h1> Список членов "Исторической Лиги" </h1>\n";
}

sub html_cleanup
{
   print "</body>\n";
   print "</html>\n";
}
```

Основная работа, как всегда, заключается в форматировании записей. Скопируем функцию `format_rtf_entry()` и проверим, не осталось ли там специальных символов, заменим управляющие слова формата RTF на тэги формата HTML.

```
sub format_html_entry
{
my ($entry_ref) = shift;

  # Преобразовать управляющие символы <, >, " и & в соответствующие
  # сущности языка HTML (&lt;, &gt;, &quot, &amp;)
  foreach my $key (keys (%{$entry_ref}))
  {
   next unless defined ($entry_ref->{$key});
   $entry_ref ->{$key} =- s/&/&amp;/g;
   $entry_ref ->{$key} =- s/\"/&quot;/g;
   $entry_ref->{$key} =- s/>/&gt;/g;
   $entry_ref->{$key} =- s/</&lt;/g;
  }
  printf "<strong>ФИО: %s</strong><BR>\n",
        format_name ($entry_ref );
  my $address = "";
  $address .= $entry_ref ->{street}
        if defined ($entry_ref ->{street});
  $address .= ", " . $entry_ref ->{city}
        if defined ($entry_ref ->{city});
  $address .= ", " . $entry_ref ->{state}
        if defined ($entry_ref ->{state});
  $address .= " " . $entry_ref ->{zip}
        if defined ($entry_ref ->{zip});
  print "Адрес: $address<br>\n" if $address;
  print "Телефон: $entry_ref ->{phone}<br>\n"
        if $entry_ref ->{phone};
  print "Э/почта: $entry_ref ->{email}<br>\n"
       if $entry_ref ->{email};
  print "Интересы: $entry_ref ->{interests}<br>\n"
        if $entry_ref ->{interests};
  print "<br>\n";
}
```

Эта функция позволяет получить вывод такого вида:

```
<strong>ФИО: Mike Artel</strong><br />
Адрес: 4264 Lovering Rd., Miami, FL 12777<br />
Телефон: 075-961-0712<br />
Э/почта: mike_artel@venus.org<br />
Интересы: Civil Rights,Education,Revolutionary War<br />
<br />
```

Причина использования
 вместо
 кроется в стремлении создать документ, который мог бы работать и с языком XHTML, являющимся более строгим, чем HTML. Несколько различий между языками HTML и XHTML кратко обсуждается в разделе "Вывод результатов".

Теперь всего лишь добавим в конструкцию switchbox элемент, который указывает на функцию создания HTML-страниц. После этого модификацию сценария gen_dir.pl можно считать завершенной:

```
# конструкция switchbox, содержащая функции форматирования
# для всех выводимых форматов
my (%switchbox) =
(
  banquet" =>    # функции оформления списка на банкет
  {
```

```
    "init"       => undef,    # инициализация не требуется
    "entry"      => \&format_banquet_entry,
    "cleanup"    => undef     # очистка не требуется
  },
  "rtf"              =>       # функции оформления RTF-формата
  {
    "init"       => \&rtf_init,
    "entry"      => \&format_rtf_entry,
    "cleanup"        => \&rtf_cleanup
  },
  "html" => # функции оформления HTML-формата
  {
    "init"       => \&html_init,
    "entry"      => \&format_html_entry,
    "cleanup"    => \&html_cleanup
  }
};
```

Получить список в формате HTML можно, запустив команду и установив полученный файл в дереве документов своего Web-сервера:

```
% ./gen_dir html > directory.html
```

После изменения списочного состава "Исторической Лиги" эту команду необходимо повторить и обновить версию на Web-узле. В качестве альтернативной стратегии можно предложить процесс cron, который бы выполнялся периодически. Предположим, что сценарий gen_dir.pl установлен в каталоге /u/paul/bin, а список "Исторической Лиги" — в дереве документов Web-сервера /usr/local/apache/htdocs/ushl. Например, для периодического выполнения сценария gen_dir.pl в 4 часа утра каждые сутки создадим в crontab-файле такую запись:

```
0 4 * * * /u/paul/bin/gen_dir.pl >
/usr/local/apache/htdocs/ushl/directory.html
```

Примечание

Пользователь, выполняющий это задание процесса cron, должен иметь право на выполнение сценариев, расположенных в моем каталоге bin, и на запись файлов в дерево документов Web-сервера.

Работа с интерфейсом DBI в Web-приложениях

DBI-сценарии, написанные нами до сих пор, в основной своей массе предназначались для использования в режиме командной строки, но применение интерфейса DBI имеет свои преимущества и в других контекстах, таких как разработка Web-приложений. Когда вы пишете DBI-сценарии, которые могут быть запущены с помощью Web-броузера, то открываете новые и очень интересные возможности взаимодействия со своими базами данных.

Например, при отображении данных в табличной форме каждое название столбца можно запросто превратить в гиперссылку, которая вызывается для пересортировки данных в этом столбце. Это позволит изменять ракурс просмотра данных одним щелчком, не вводя при этом никаких запросов. Существует воз-

можность задать форму, в которой пользователь сможет сам задавать критерии поиска по базе данных, после чего отобразить страницу, содержащую результаты выборки. Эти и другие подобные функции смогут значительно расширить возможности интерактивного доступа к содержимому вашей базы данных. Кроме того, интерактивные возможности Web-броузеров обычно выше интерактивных возможностей терминальных окон, что также способствует улучшению вывода получаемого на экране.

В этом разделе будут созданы следующие сценарии для Web-приложений.

- **Универсальный броузер таблиц базы данных sampdb.** Эта задача не связана с какой-то узкоспециальной задачей и иллюстрирует несколько концепций Web-программирования, предоставляя удобные средства просмотра содержимого таблиц.
- **Броузер оценок позволит просматривать любую оценку за любой тест или викторину.** Такой броузер удобен как средство быстрого просмотра результатов оценочных тестов и очень удобен при необходимости построения оценочной кривой экзамена, позволяющей определить ранг самого экзамена.
- **Сценарий подбора членов "Исторической Лиги" по научным интересам.** Задача подбора осуществляется самим пользователем с помощью поисковой фразы. Версия такого сценария, работающая из командной строки, нами уже создана, но Web-версия будет очень наглядной и просто для сравнения двух подходов к решению одной задачи. (В действительности мы разработаем два варианта Web-броузера. Один из них будет основываться на поиске по соответствующему шаблону, аналогично `interests.pl`, а другой будет осуществлять поиск по индексу FULLTEXT.)

Напишем эти сценарии с применением модуля CGI.pm Perl, который является простейшим способом связи DBI-сценариев с Web-сервером. (Как получить модуль, рассказывается в приложении А, "Получение и инсталляция программного обеспечения".) Модуль CGI.pm называется так потому, что он позволяет создавать сценарии, использующие CGI-протокол. Этот протокол определяет связь Web-сервера с другими программами. Модуль CGI.pm обрабатывает многочисленные детали интерфейса между Web-сервером и создаваемым сценарием (например, сбор значений параметров, передаваемых как ввод в ваш сценарий). Модуль CGI.pm также обеспечивает удобные методы генерации вывода HTML, которые уменьшают вероятность создания ошибочной HTML-страницы по сравнению с написанием HTML-страницы вручную.

В этой главе вы узнаете довольно много о самом модуле CGI.pm. Этих знаний может быть достаточно для создания Web-приложений, но, конечно, это далеко не все возможности модуля CGI.pm. Более глубокое изложение материала об этом модуле можно найти в других изданиях или на Web-узле `http://stein.cshl.org/WWW/software/CGI/`.

Среди прочего справочного материала, повествующего об использовании модуля CGI.pm совместно с СУБД MySQL и интерфейсом DBI, можно назвать книгу *Применение MySQL и Perl в Web-приложениях* (ИД "Вильямс", 2002), написанную самим автором. Сценарии, которые могут работать с Web-серверами, описаны в частях этой главы, которые можно найти в каталоге `perlapi/web` дистрибутива `sampdb`.

Настройка сервера Apache для работы с CGI-сценариями

Кроме интерфейса DBI и модуля CGI.pm, для написания сценариев, работающих с Web, нам потребуется еще один компонент: Web-сервер. Все приведенные здесь сценарии будут разрабатываться на примере Web-сервера Apache. Для работы с другими серверами команды придется немного изменить.

Различные части инсталляции сервера Apache обычно размещаются в каталоге `/usr/local/apache` для ОС UNIX и в каталоге `C:\Apache` для ОС Windows. В нашем случае нас будут интересовать больше всего следующие подкаталоги этого каталога: `htdocs` (каталог размещения дерева документов HTML), `cgi-bin` (каталог размещения сценариев и программ, которые запускаются Web-сервером) и `conf` (каталог размещения конфигурационных файлов). Эти каталоги могут располагаться и в другом месте, но при этом потребуется соответствующая настройка.

Необходимо проверить, что каталог `cgi-bin` не находится внутри дерева документов сервера Apache. В противном случае сценарии нельзя запрашивать как простой текстовый файл. Это вопрос безопасности. Нельзя позволить пользователям запросто просматривать ваши сценарии, пользуясь "дырами", имеющимися в защите.

Для того чтобы установить CGI-сценарий для работы с сервером Apache, разместите его в каталоге `cgi-bin`. В ОС UNIX этот сценарий должен начинаться со строки `#!` и должен быть исполняемым, аналогично тому, как это делалось в сценарии обработки командной строки. Кроме того, неплохо изменить права доступа, чтобы его мог запускать на выполнение и модифицировать только сам пользователь. Например, если сервер Apache работает как пользователь `www`, можно воспользоваться следующими командами:

```
% chown www myscript.pl
% chmod 500 myscript.pl
```

Эти команды можно выполнить, работая только с правами пользователя `root`. Если у вас нет таких прав, попросите системного администратора сделать это за вас.

В ОС Windows команды `chown` и `chmod` не нужны, но сценарий по-прежнему должен начинаться со строки `#!`. В этой строке указан полный путь к программе Perl. Например, если Perl установлена как `C:\Perl\bin\perl.exe`, строка `#!` может иметь следующий вид:

```
#! C:/Perl/bin/perl -w
```

С другой стороны, в Windows NT-системах строку можно упростить, так как при установленной переменной среды `PATH` в ней задастся полный путь к Perl:

```
#! perl -w
```

Все сценарии Perl, имеющиеся в дистрибутиве `sampdb`, задают путь к Perl в строке `#!` в виде `/usr/bin/perl`. При необходимости можно внести изменения в соответствии с конкретными установками вашей системы.

После того как этот сценарий установлен, нужно указать со своего броузера соответствующий URL на Web-сервере. Например, если Web-сервер расположен на узле `www.snake.net`, запрос сценария `myscript.pl` из него можно сделать с помощью такого URL:

```
http://www.snake.net/cgi-bin/myscript.pl
```

Вызов сценария из Web-броузера инициирует его выполнение Web-сервером. Вывод, полученный от работы сценария, возвращается обратно и отображается в виде Web-страницы на броузере.

При работе с DBI-сценариями из командной строки предупреждения и сообщения об ошибках выводятся непосредственно на терминал пользователя. В Web-среде терминала как такового не существует, и эти сообщения выводятся в журнал регистрации ошибок Web-сервера Apache. Нужно определиться с местом размещения этого сценария, так как он содержит много информации, полезной при отладке.

В моей системе это файл `error_log`, размещенный в каталоге `logs` в корневом каталоге Apache `/usr/local/apache`. Но он может размещаться и в любом другом каталоге. Размещение журнала задается директивой `ErrorLog` в конфигурационном файле `httpd.conf`, который размещается в каталоге `conf` Web-сервера Apache.

Знакомство с модулем CGI.pm

В начале сценария Perl, работающего с модулем CGI.pm, размещается оператор `use`, после чего создается объект CGI, предоставляющий доступ к методам и переменным CGI.pm:

```
use CGI qw(:standard);
```

После этого для создания различных структур HTML можно вызывать функции модуля CGI.pm. Обычно имена функций совпадают с соответствующими элементами языка HTML. Например, для генерации заголовка 1-го уровня вызываются функции `h1()` и `p()`:

```
print h1 ("Это заголовок");
print p ("Это абзац");
```

Модуль CGI.pm также поддерживает стиль использования, который позволяет вызывать методы как функции, без предшествующего суффикса "`$cgi->`". Этот синтаксис здесь не используется, так как нотация вида "`$cgi->`" более привычна для работы с интерфейсом DBI и эта форма позволяет избежать конфликта между именами функций модуля CGI.pm и любыми другими функциями, которые будут созданы в процессе разработки сценария. Например, CGI.pm имеет функцию `tr()`, которая создает тэги ограничения ячеек в строках HTML-таблиц (тэги `<tr>` и `</tr>`). Имя этой функции конфликтует со встроенной функцией транслитерации `tr`. Для того чтобы решить эту проблему, при использовании интерфейса вызова функций CGI.pm сделаем вызов `tr()` как `Tr()` или `TR()`. При использовании объектно-ориентированного интерфейса эта проблема не возникает, так как в этом случае метод `tr()` вызывается как метод вашего объекта `$cgi` (т.е. с использованием синтаксиса `$cgi->tr()`), что ясно показывает, что вы не обращаетесь к встроенной функции.

Проверка введенных параметров

Одна из важнейших задач модуля CGI.pm — выполнение рутинной работы по накоплению информации, которую поставляет сценарию Web-сервер. Для этого нужно только запустить функцию `param()`. Получить имена имеющихся параметров можно с помощью выражения

```
my (@param) = $cgi->param ();
```

Для получения значения конкретного параметра укажите его имя, как показано ниже. Если параметр установлен, функция `param()` возвратит его значение или в противном случае возвратит значение `undef`.

```
my $my_param = param ("my_param");
if (defined ($my_param))
{
   printf "(значение my_param: %s\n", $cgi->param ("my_param");
}
else
{
   print "my_param не установлен \n";
}
```

Вывод результатов

Модуль CGI.pm также предоставляет в ваше распоряжение выводимую информацию, которая отсылается на броузер клиента. Рассмотрим следующий документ в формате HTML:

```
<html>
<head>
<title> Моя страница </title>
</head>
<body bgcolor="white">
<h1> Заголовок страницы </h1>
<p>1-й абзац.</p>
<p>2-й абзац.</p>
</body>
</html>
```

Следующий код использует соответствующие функции вывода модуля CGI.pm для того, чтобы создать аналогичный документ:

```
#! /usr/bin/perl -w
# simple_doc.pl - создать простейшую HTML-страницу
use strict;
use CGI qw(:standard);
print header ();
print start_html (-title => "Заголовок страницы", -bgcolor => "white");
print h1 ("Заголовок страницы");
print p ("1-й абзац.");
print p ("2-й абзац.");
print end_html ();
```

Функция `header()` генерирует заголовок `Content-Type:`, который предшествует содержимому страницы. Этот заголовок необходим при создании Web-страниц из сценариев для того, чтобы предупредить броузер о типе ожидаемого документа. (Это контрастирует со стилем написания статических HTML-страниц, для которых не надо создавать заголовок, так как Web-сервер сам создает их автоматически.) По умолчанию функция `header()` создает заголовок, который имеет следующий вид:

```
Content-Type: text/html
```

Сразу же за вызовом функции `header()` следуют вызовы функций, которые генерируют содержимое страниц. Так, функция `start_html()` создает тэги, на-

чиная с открывающего тэга <html> и заканчивая тэгом <body>. Функции h1() и p() записывают элементы заголовка и абзаца, а функция end_html() добавляет тэги, закрывающие документ.

Как показано на примере вызова функции start_html(), многие функции модуля CGI.pm позволяют задавать именованные параметры в формате *name=>value*. Это является большим преимуществом для функций, принимающих много параметров, большинство из которых является факультативным. Это позволяет задавать в произвольном порядке только необходимые параметры.

Использование функций вывода модуля CGI.pm совсем не означает, что разработчик не может пользоваться обычными операторами языка HTML. Эти два подхода можно совмещать, комбинируя вызовы функций CGI.pm и обычные литеральные тэги. Однако одно из основных преимуществ использования модуля CGI.pm при генерации вывода перед написанием HTML-кода вручную заключается в том, что в первом случае разработчик может мыслить в терминах логических единиц, а не в терминах отдельных тэгов, а это снижает вероятность ошибок в исходном коде. (Упоминание о "вероятности" напоминает нам о том, что модуль CGI.pm не может предотвратить появление таких ошибок, как включение списка в заголовок.)

Модуль CGI.pm также обеспечивает преимущества переносимости, которых вы никогда не добьетесь, написав свой собственный HTML-код. Например, начиная с версии 2.69 модуль CGI.pm автоматически создает XHTML-код. Если вы пользуетесь более старой версией модуля CGI.pm, которая создает обычный HTML-код, то, для того, чтобы модифицировать свои сценарии и создать XHTML-код, необходимо только обновить версию самого модуля CGI.pm.

XHTML-код подобен HTML-коду, но его формат лучше проработан. HTML-код прост в изучении и использовании, но одной из сложностей является то, что различные броузеры интерпретируют его по-разному. Например, они могут обрабатывать один и тот же ошибочный HTML-код различными способами. Это значит, что не совсем "правильная" страница может нормально работать в одном броузере и не работать в другом. Требования, предъявляемые к XHTML-коду, значительно строже. Ниже описаны некоторые различия между кодами HTML и XHTML.

- В отличие от языка HTML, любой открывающий тэг языка XHTML должен иметь соответствующий ему закрывающий тэг. Например, абзацы пишутся с использованием тэгов и , но в HTML-документах закрывающий тэг обычно отсутствует. В XHTML-документах тэг обязателен. Многие тэги HTML (
 и <hr>) не имеют никакого тела. Требования языка XHTML таковы, что все тэги должны закрываться (
</br> и <hr></hr>). Для этих целей язык XHTML разрешает использование кратких форм (
, <hr/>), служащих как для открытия, так и для закрытия. Однако старые броузеры, встречая такие тэги, иногда могут путать их имена. Возникновение таких проблем можно предотвратить, вставив пробел перед косой чертой (
 и <hr />).
- В языке HTML имена тэгов и атрибутов не чувствительны к регистру. Например, <tr> и <TR> — это одно и то же. В XHTML имена тэгов и атрибутов должны задаваться только в нижнем регистре, таким образом, допустимо только написание <tr>.

- Значения атрибутов в языке HTML могут не браться в кавычки или пропускаться. Например, в языке HTML допустима следующая конструкция ячейки данных:

  ```
  <td width=40 nowrap>какой-то текст</td>
  ```

  В языке XHTML атрибуты обязательно должны снабжаться значениями, которые должны заключаться в кавычки. Общее соглашение для атрибутов HTML, которые обычно используются без значения, является значением. В языке XHTML эквивалент предшествующего элемента `<tr>` выглядит следующим образом:

  ```
  <td width="40" nowrap="nowrap">какой-то текст</td>
  ```

Все Web-сценарии, приведенные в этой книге, генерируют код, удовлетворяющий правилам XHTML. В этой главе всю ответственность за создание безошибочного XHTML-кода мы возложим на модуль CGI.pm. В сценариях, которые обсуждаются в главе 8, "Интерфейс API для языка написания сценариев PHP", тэги создаются вручную, так как язык PHP не имеет таких же функций генерирования тэгов, как модуль CGI.pm.

Управляющие HTML-последовательности и текст URL

Для обработки специальных символов на Web-страницах, которые должны обрабатываться соответствующим образом, предназначается метод `escapeHTML()`. Это также справедливо и в отношении URL, которые могут содержать управляющие символы, хотя для этого случая предназначена функция `escape()`. Здесь очень важно применить правильный метод кодировки, поскольку каждый метод применяет разные методы кодировки и кодирует специальные символы в совершенно различных форматах. Рассмотрим следующий короткий сценарий Perl:

```
#! /usr/bin/perl -w
# escape_demo.pl - демонстрация функций кодировки вывода модуля CGI.pm
use CGI qw(escapeHTML escape); # импортировать escapeHTML() и escape()

$s = "x<=y, right?";
print escapeHTML ($s) . "\n"; #код HTML-текста
print escape ($s) . "\n";     #код URL
```

При запуске этого сценария можно увидеть, что он выводит результат, который позволяет сделать вывод, что кодирование текста в формате HTML совсем не одно и то же, что кодирование в формате URL:

```
x&lt;=y, right?
x%3C%3Dy%2C%20right%3F
```

Сценарий `escape_demo.pl` импортирует имена функций кодировки в операторе `use CGI`. Они не включены в стандартный набор функций, поэтому их необходимо импортировать, даже если при этом импортируется стандартный набор:

```
use CGI qw (:standard escapeHTML escape);
```

Создание многоцелевых Web-страниц

Одной из главных причин, для чего нужно создавать сценарии, генерирующие динамические HTML-страницы, а не статические HTML-документы, является то, что сценарий должен иметь возможность создавать разные страницы в зависимо-

сти от способа их вызова. Все создаваемые нами CGI-сценарии имеют это свойство и работают по такому алгоритму.

- При первом запросе сценария из броузера он генерирует начальную страницу, позволяющую выбрать тип информации, которая вам требуется.
- После того как этот выбор сделан, сценарий будет вызван повторно, но на этот раз он будет выбирать и отображать вторую страницу, содержащую конкретную информацию, которую вы выбрали на предыдущем этапе.

Главной особенностью здесь является то, что выборка, сделанная на первом этапе, определяет содержимое второй страницы. Но если не производить каких-то предварительных операций, Web-страницы не будут зависеть одна от другой. Решением будет являться сценарий, генерирующий страницы, устанавливающие значения параметров, управляющие повторным запуском сценария.

При первом запуске сценария параметр еще не имеет значения, что означает, что сценарий находится на первой странице. После того как информация задана, эта страница запускает сценарий повторно, но теперь уже параметр имеет какое-то значение, указывающее сценарию, что нужно делать дальше.

Есть несколько способов передачи команд из страницы обратно сценарию. Одним из них является вывод формы, которую должен заполнять сам пользователь. И когда пользователь подтверждает заполненную форму, ее содержимое пересылается на Web-сервер. Сервер передает информацию сценарию, который различает, что было заказано, вызвав функцию `param()`. Такие манипуляции позволяет производить наш CGI-сценарий (сценарий, позволяющий пользователю вводить ключевое слово для поиска списка членов "Исторической Лиги").

Другим способом указания команд сценарию является передача параметров в конце URL, посылаемого на Web-сервер при запросе выполнения сценария. Воспользуемся этим методом в броузере таблиц базы данных `sampdb` и в сценариях броузера оценок учащихся. Этот метод реализуется сценарием, генерирующим страницу, содержащую гиперссылки. Выбор ссылки не только запускает сценарий повторно, но и задает значение параметра, которое указывает сценарию, что надо делать. На самом деле сценарий запускается автоматически в различных режимах работы, выдавая различные результаты в зависимости от выбранной гиперссылки.

Такой сценарий позволяет производить рекурсивный вызов самого себя посылкой адреса страницы, содержащей гиперссылку на свой собственный URL. Например, сценарий `myscript.pl` возвращает управление на самого себя с помощью такой ссылки:

```
<a href="/cgi-bin/myscript.pl">Click Me!</a>
```

И когда пользователь щелкает на тексте "Click Me!", его броузер возвращает на Web-сервер запрос на инициализацию сценария `myscript.pl` (т.е. самого себя). Конечно, в этом случае активизируется та же страница, так как никакой другой информации нет. Однако если к URL прибавить параметр, то при повторной инициализации сценарий может вызвать метод `param()`, чтобы проанализировать параметр и в зависимости от его значения предпринять определенные действия.

Параметр может быть добавлен в конец URL, если добавить символ "?" с последующей парой "*name*=*value*". Например, для того, чтобы добавить параметр, определяющий размер со значением `large`, напишите следующий URL:

```
/cgi-bin/myscript.pl?size=large
```

Для добавления нескольких параметров такие пары необходимо разделять символами ";"[1]:

```
/cgi-bin/myscript.pl?size=large;color=blue
```

Сценарий CGI рекурсивного URL с присоединенными параметрами должен начинаться с вызова метода `url()`. Он позволит получить свой собственный URL, а затем добавить параметры следующим образом:

```
$url = url ();                  # получить URL сценария
$url .= "?size=large";    # добавить первый параметр
$url .= ";color=blue";    # добавить второй параметр
```

Пользуясь методом CGI.pm `url()`, после того как URL создан, можно сгенерировать тэг гиперссылки `<a>`:

```
print a ({-href => $url}, "Click Me!");
```

Оператор `print` выдаст гиперссылку, имеющую такой вид:

```
<a href="/cgi-bin/url.pl?size=large;color=blue">Click Me!</a>
```

В этом примере значение `$url` конструируется в несколько "легкомысленном" варианте, так как при этом не учитывается вероятность присутствия в нем специальных символов. До тех пор, пока разработчик не убежден на все 100% в том, что значения или гиперссылки не содержат специальных символов, лучше всего пользоваться функциями кодировки модуля CGI.pm. Функция `escape()` кодирует значения, которые будут присоединяться в URL, а `escapeHTML()` кодирует обычный HTML-код. Например, если значение метки гиперссылки хранится в `$label` и значения для параметров `size` и `color` хранятся в переменных `$size` и `$color`, кодировку можно выполнить следующим образом:

```
$url = sprintf ("%s?size=%s;color=%s"
        url (), escape ($size), escape ($color));
print a ({-href => $url}, escapeHTML ($label));
```

Посмотреть, как это работает, лучше всего на примере короткого CGI-сценария. Будучи вызванным, сценарий `flip_flop.pl` отображает страницу Page A, которая содержит единственную гиперссылку. Выбор этой ссылки повторно активизирует сценарий, но при этом параметр установлен таким образом, что сценарий отобразит уже страницу Page B, которая также содержит гиперссылку на сценарий, но на этот раз без значения в параметре. Поэтому выбор ссылки на странице Page B возвращает нас на исходную страницу. Все последующие вызовы сценария будут переключать управление между страницами Page A и Page B:

```
#! /usr/bin/perl -w
# flip_flop.pl - простой сценарий с выводом на нескольких страницах
CGI.pm
use CGI qw(:standard);
my $url;
# определить, какую страницу необходимо отобразить на основании при-
```

[1] Модуль CGI.pm также в качестве разделителя параметров принимает символ '&'. Интерфейсы API других языков для программирования Web-страниц варьируются в своих соглашениях, поэтому при программировании необходимо знать, чего ожидать, ';' или '&', и соответствующим образом конструировать URL.

```
сутствия или отсутствия
# параметра pageb
if (!defined (param ("pageb"))) # отобразить страницу A без ссылки на
страницу B
{
   $this_page = "A";
   $next_page = "B";
   $url = url () . "?pageb=1";
}
else # отобразить страницу B без ссылки на страницу A
{
   $this_page = "B";
   $next_page = "A";
   $url = url ();
}
print header ();
print start_html (-title => "Flip-Flop: Страница $this_page",
-bgcolor => "white");
print p ("Это страница $this_page. Для выборки страницы $next_page, "
    . a ({-href => $url}, "щелкните здесь"));
print end_html ();
```

Установите этот сценарий в каталоге `cgi-bin`, а затем запросите его из своего броузера, воспользовавшись таким URL:

```
http://www.snake.net/cgi-bin/flip_flop.pl
```

Выберите ссылку на странице несколько раз, чтобы узнать, как сценарий меняет страницы.

Теперь предположим, что другой клиент получает доступ и делает запрос `flip_flop.pl`. Что при этом происходит? Помешаете ли вы друг другу? Нет! Так как первоначальный запрос, поступивший от любого пользователя, будет включать параметр pageb и сценарий ответит стартовой страницей. После этого запросы, поступившие от любого пользователя, будут включать или опускать параметр, в соответствии с которым вы просматривали определенную таблицу. Сценарий `flip_flop.pl` генерирует различные последовательности страниц для каждого клиента, совершенно независимо от действий другого клиента.

Подключение к серверу MySQL из Web-сценария

Сценарии, работающие с командной строкой, разработанные нами в разделе "Подготовка интерфейса к работе", имеют общую преамбулу, которая устанавливает связь с сервером MySQL. Наши CGI-сценарии также имеют общий код:

```
#! /usr/bin/perl -w
use strict;
use DBI;
use CGI qw(:standard);
use Cwd;

# конфигурационный файл, который должен содержать параметры соединения
для ОС UNIX
my $option_file = "/usr/local/apache/conf/sampdb.cnf";
my $option_drive_root;
# пропустить значения из файла для ОС Windows
if ($^O =~ /^MSWin/i || $^O =~ /^dos/)
```

```
{
   $option_drive_root = "C:/ ";
   $option_file = "/Apache/conf/sampdb.cnf";
}
# собрать информацию об источнике данных и соединиться с сервером
(в Windows сначала необходимо сохранить
# текущий рабочий каталог, перейти на устройство, содержащее конфигу-
рационный файл,
# соединиться, а затем восстановить текущий каталог)
my $orig_dir;
if (defined ($option_drive_root))
{
   $orig_dir = cwd ();
   chdir ($option_drive_root)
        or die "Нельзя перейти к каталогу $option_drive_root: $!\n";
}
my $dsn =
DBI:mysql:sampdb;mysql_read_default_file=$option_file";
my $dbh = DBI->connect ($dsn, undef, undef,
{ RaiseError => 1, PrintError => 0 });
    if (defined ($option_drive_root))
{
chdir ($orig_dir)
or die "Нельзя поменять каталог на $orig_dir: $!\n";
}
```

Эта преамбула несколько отличается от той, которую мы использовали в сценариях, работающих с командной строкой.

- Первый раздел теперь содержит операторы `use CGI` и `use Cwd`. Первый предназначен для модуля CGI.pm, а второй для модуля, возвращающего имя пути текущего рабочего каталога; он используется при работе сценария под управлением ОС Windows.
- Аргументы, принимаемые из командной строки, больше не анализируются. Предполагается, что все они перечислены в конфигурационном файле.
- Вместо `mysql_read_default_group` для чтения стандартного конфигурационного файла воспользуемся командой `mysql_read_default_file` для чтения единственного файла, содержащего параметры, которые будут использоваться сценариями при доступе к базе данных `sampdb`. Как видно, программа ищет параметры, хранящиеся в файле `/usr/local/apache/conf/sampdb.cnf` (ОС UNIX) или в файле `C:\Apache\conf\sampdb.cnf` (ОС Windows). Заметим, что в ОС Windows программа переходит в корневой каталог устройства, где конфигурационный файл был размещен до соединения, и обратно в исходный каталог после этого. Причина этого "обходного маневра" объяснялась ранее, в разделе "Определение параметров соединения".

Дистрибутив `sampdb` содержит конфигурационный файл `sampdb.cnf`, который можно использовать Web-сценарием, разработанным на интерфейсе DBI. Он выглядит следующим образом:

```
[client]
host=cobra.snake.net
user=sampadm
password=secret
```

Для того чтобы воспользоваться Web-сценариями, разработанными в вашей системе, при использовании конфигурационного файла с другим именем в преамбуле достаточно поменять имя конфигурационного файла. Кроме того, придется в конфигурационном файле перечислить такие значения, отражающие имя узла MySQL сервера, как имя пользователя MySQL и его пароль.

В ОС UNIX в качестве владельца конфигурационного файла следует задать пользователя, который запускает Web-сервер Apache, и установить код доступа 400 или 600, чтобы никто другой не смог его прочитать. Это не позволит другим пользователям, которые имеют право доступа на узел Web-сервера, непосредственно считывать конфигурационный файл и обеспечит одну из степеней защиты.

К сожалению, этот файл будет доступен для чтения любому, кто имеет право устанавливать сценарии для выполнения на Web-сервере. Все, что им надо сделать, — это записать сценарий, который явным образом открывает конфигурационный файл и отображает его содержимое на Web-странице. Так как его сценарий запущен пользователем Web-сервера, он будет иметь все права на чтение конфигурационного файла и, следовательно, сможет узнать все пароли и параметры. Этого ему будет вполне достаточно для доступа к базе данных `sampdb`. Поэтому может оказаться целесообразным создать отдельного пользователя MySQL, который имеет право только на чтение (`SELECT`) из базы `sampdb`, а затем поместить в конфигурационный файл базы данных `sampdb.cnf` не свое собственное имя и пароль, а имя и пароль этого пользователя. Таким образом, можно избежать риска подключения сценариев к базе данных в качестве пользователя с правами модификации таблиц. В главе 11, "Общее администрирование MySQL", будет показано, каким образом создается учетная запись пользователя MySQL, который будет иметь права, ограниченные только чтением.

В качестве альтернативы можно настроить выполнение сценариев под управлением механизма suEXEC сервера Apache. Это позволит начать выполнение сценария под управлением привилегированного пользователя и задать сценарию получение параметров соединения из конфигурационного файла, права на чтение которого будут только у этого пользователя. Так, например, можно поступить со сценариями, которые должны иметь право на запись в базу данных. Другой подход можно реализовать с помощью Web-сервера Apache 2.x, который позволяет запускать сценарии, размещающиеся в определенных каталогах, и работать с правами определенных пользователей. Это позволит произвести установку владельца и режима конфигурационного файла `sampdb.cnf`, и ваши сценарии будут доступны только вам; другие пользователи ими воспользоваться не смогут.

В качестве примера другого подхода можно привести создание сценария, который сам запрашивает имя и пароль пользователя с тем, чтобы использовать полученные значения для подключения к серверу MySQL. Такой подход приемлем для сценариев, предназначенных для выполнения задач администрирования. Но в любом случае нужно хорошо понимать, что отдельные методы опроса имени и пароля пользователя могут быть предметом атаки любого взломщика, имеющего возможность установить дешифратор в сети между вами и сервером. Однако эта тема выходит за пределы материала данной книги.

Как понятно из всего предыдущего материала, безопасность Web-сервера может быть связана с большими проблемами. К сожалению, объем этой книги не

позволяет уделить этой теме достаточно внимания. Можно посоветовать обратиться к документации по безопасности сервера Apache. Кроме того, желательно поинтересоваться вопросами и ответами по проблеме безопасности WWW, которые можно найти по адресу

`http://www.w3.org/Security/Faq/`.

Броузер базы данных sampdb

Разработаем для нашего первого Web-приложения простейший сценарий – `db_browse.pl`. В его функции будет входить получение перечня таблиц, имеющихся в базе данных `sampdb`, и интерактивный просмотр содержимого любой из этих таблиц непосредственно с вашего Web-броузера. Сценарий `db_browse` работает следующим образом.

- Первый запуск сценария `db_browse.pl` из вашего броузера подключает вас к MySQL-серверу и выбирает перечень таблиц из базы данных `sampdb`, который отсылается на ваш броузер в виде страницы, на которой каждая таблица представлена в виде доступной гиперссылки. При выборе какой-либо таблицы броузер посылает запрос на Web-сервер сценарию `db_browse.pl` на отображение содержимого этой таблицы.
- Если в момент запуска сценарий `db_browse.pl` получает имя таблицы с Web-сервера, он выбирает содержимое таблицы и отображает полученную информацию в Web-броузере. Заголовок каждого столбца данных является именем столбца в таблице. Заголовки являются ссылками, щелчок на которых задает сортировку по данному столбцу.

Но следует предостеречь вас: таблицы базы данных `sampdb` относительно невелики, поэтому не составляет труда послать на ваш броузер содержимое всей таблицы. Но если отредактировать сценарий `db_browse.pl` для того, чтобы он мог отображать данные из самых различных баз данных, в которых могут встречаться таблицы достаточно большого размера, нужно будет подумать и о добавлении в операторы выборки предложения `LIMIT`.

Перед тем как продолжить изложение материала, необходимо напомнить, что несмотря на то, что пример сценария `db_browse.pl` очень поучителен и удобен для демонстрации некоторых весьма полезных концепций программирования Web-приложений, он также содержит определенные "дыры" в системе безопасности, которыми могут воспользоваться злоумышленники. Этот сценарий очень просто обмануть и заставить его отображать любую таблицу, которая доступна пользователю, указанному в конфигурационном файле `sampdb.cnf`. Об этом мы поговорим чуть позже, но с этого момента вам следует запомнить, что этот сценарий можно устанавливать только в случае, если эта учетная запись имеет доступ к малозначительным данным. Но лучшим решением будет удаление этого сценария из вашего каталога `cgi-bin` после того, как будут проверены все его возможности. (Или установите его на сервере, к которому нет доступа из сети, или на сервере, к которому не имеют доступа ненадежные пользователи.) Вот конкретный пример того, почему сценарий `db_browse.pl` может создавать проблемы. В главе 8, "Интерфейс API для языка написания сценариев PHP", мы создадим сценарий, позволяющий чле-

нам "Исторической Лиги" редактировать свои регистрационные данные через Internet. Доступ к записям контролируется с помощью паролей, хранящихся в таблице member_pass. Сценарий db_browse.pl может позволить любому желающему просмотреть таблицу с паролями и получить доступ к информации, необходимой для редактирования любой записи таблицы member.

Предположим, что мы не слышали этих предупреждений, и посмотрим на сценарий db_browse.pl в работе. Создадим объект CGI в основном теле сценария db_browse.pl и разместим начальную часть Web-страницы. Затем на основании параметра tbl_name рассмотрим, нужно ли отображать структуру определенной таблицы:

```
#! /usr/bin/perl -w
# db_browse.pl - Разрешить просмотр базы данных sampdb через Web
use strict;
use DBI;
use CGI qw (:standard escapeHTML escape);
# ... установить соединение с базой данных (не показано) ...
my $db_name = "sampdb";
# вывести начальную часть страницы
my $title = "Броузер базы данных $db_name";
print header ();
print start_html (-title => $title, -bgcolor => "white");
print h1 ($title);
# параметры для поиска URL
my $tbl_name = param ("tbl_name");
my $sort_col = param ("sort_col");
# если переменная $tbl_name не имеет значения,
# отобразить перечень таблиц для выбора
# В противном случае отобразить содержимое заданной таблицы,
# если установлено значение $sort_column,
# сортировать по заданному столбцу.
if (!defined ($tbl_name))
{
  display_table_names ($dbh, $db_name)
}
else
{
  display_table_contents ($dbh, $tbl_name, $sort_col);
}
print end_html ();
```

Процедура определения значения параметра, полученного от Web-сервера, максимально упрощена потому, что всю эту работу выполняет модуль CGI.pm. Для этого достаточно вызвать метод param(), указав имя параметра, который нас интересует. В сценарии db_browse.pl этим параметром будет tbl_name. Если он не определен или является пустым, то это — первоначальная инициализация сценария. В таком случае сценарий отображает список таблиц. В противном случае отображается содержимое таблицы, указанной в параметре tbl_name, отсортированной по содержимому столбца, имя которого указано в параметре sort_column.

Функция display_table_list() генерирует начальную страницу. Она делает выборку списка таблиц и выводит таблицу в HTML-формате, состоящую из одного столбца, с именем таблицы базы данных в каждой ячейке.

```
sub display_table_names
{
```

```
  my ($dbh, $db_name) = @_;
  print p ("Для выбора таблицы щелкните на ее имени:");
  # ссылка на массив имен таблиц
  my $ary_ref = $dbh->selectcol_arrayref (qq{ SHOW TABLES FROM $db_name
});
  # Создать список с помощью функций ul() (несортированный список) и
  # li() (перечень объектов). Все объекты являются гиперссылками, которые
  # вызывает сценарий для отображения соответствующей таблицы.
  my @item;
  foreach my $tbl_name (@{$ary_ref})
  {
    my $url = sprintf ("%s?tbl_name=%s", url (), escape ($tbl_name));
    my $link = a ({-href => $url}, escapeHTML ($tbl_name));
    push (@item, li ($link));
  }
  print ul (@item);
}
```

Функция `li()` вставляет вокруг каждого распечатываемого значения тэги `<li>` и `</li>`, а функция `ul()` добавляет тэги `<ul>` и `</ul>` вокруг набора величин. Все имена таблиц в списке представляются гиперссылками, которые вызывают рекурсивно сценарий для отображения содержимого указанной таблицы. Полученный список, созданный функцией `display_table_names()`, имеет следующий вид:

```
<ul>
<li><a href="/cgibin/
localhost/db_browse.pl?tbl_name=absence">absence</a></li>
<li><a href="/cgi-
bin/localhost/db_browse.pl?tbl_name=event">event</a></li>
<li><a href="/cgi-
bin/localhost/db_browse.pl?tbl_name=member">member</a></li>
...
</ul>
```

Если при запуске сценария `db_browse.pl` параметр `tbl_name` содержит определенное значение, сценарий передает это значение функции `display_table_contents()` вместе с именем столбца, по которому будет производиться сортировка:

```
sub display_table_contents
{
my ($dbh, $tbl_name, $sort_col) = @_;
my @rows;
my @cells;
# если столбец, по которому будет производиться сортировка, не задан, ис-
пользовать первый столбец
$sort_col = "1" if !defined ($sort_col);

# вывести ссылку, которая возвращает пользователя на первую страницу таб-
лицы
print p (a ({-href => url ()},"Отобразить список таблиц"));

print p (strong ("Содержимое таблицы $tbl_name:"));

my $sth = $dbh->prepare (qq{
            SELECT * FROM $tbl_name ORDER BY $sort_col
            LIMIT 200
```

```
        });
$sth->execute ();

# Использовать в качестве заголовков имена столбцов таблицы базы данных
# Заголовки HTML-таблицы. Каждое имя должно быть гиперссылкой,
# которая должна вызываться повторно для повторного отображения таблицы,
# с сортировкой по указанному столбцу.

foreach my $col_name (@{$sth->{NAME}})
{
  my $url = sprintf ("%s?tbl_name=%s;sort_col=%s",
                     url (),
                     escape ($tbl_name),
                     escape ($col_name));
  my $link = a ({-href => $url}, escapeHTML ($col_name));
  push (@cells, th ($link));
}

push (@rows, Tr (@cells));

# display table rows
while (my @ary = $sth->fetchrow_array ())
{
  @cells = ();
  foreach my $val (@ary)
  {
    # Если непустое - отобразить значение, если пустое - неразрывный про-
бел
    if (defined ($val) && $val ne "")
    {
        $val = escapeHTML ($val);
    }
    else
    {
        $val = " ";
    }
    push (@cells, td ($val));
  }
  push (@rows, Tr (@cells));
}

# Отобразить таблицу с рамкой
print table ({-border => "1"}, @rows);
}
```

Если столбец не указан, сортировка будет производиться по первому столбцу (столбцы можно сортировать по их порядковому номеру с помощью предложения `ORDER BY 1`). Этот запрос также включает предложение `LIMIT 200`, чтобы предусмотреть посылку сценарием большого количества данных на ваш броузер. (Маловероятно, что это случится с таблицами из базы данных `sampdb`, но вполне может произойти, если вы адаптируете сценарий для отображения содержимого таблиц из других баз данных.) Функция `display_table_contents()` отображает строки из таблицы как HTML-таблицу, пользуясь функциями `th()` и `td()` для того, чтобы создать заголовок таблицы и ячейки данных в ней. Функция `Tr()` здесь используется для группировки данных в строки и `table()`, чтобы создать тэги `<table>`, окружающие строки.

HTML-таблица представляет заголовки столбцов как гиперссылки, которые позволяют повторно отображать в таблицу базы данных; эти ссылки включают параметр sort_col, который определяет явным образом столбец, по которому будет производиться сортировка. Например, для страницы, которая отображает содержимое таблицы event, ссылки заголовков столбцов выглядят следующим образом:

```
<a href="/cgi-bin/db_browse.pl?tbl_name=event&sort_col=date">date</a>
<a href="/cgi-bin/db_browse.pl?tbl_name=event&sort_col=type">type</a>
<a href="/cgi-bin/db_browse.pl?tbl_name=event&sort_col=event_id">
event_id</a>
```

Функция display_table_contents() использует приемы превращения пустых значений в неразрывные пробелы ( ). В таблице с рамками некоторые броузеры будут неправильно отображать границы для пустых ячеек; такие проблемы решаются путем добавления в ячейки неразрывных пробелов.

Сценарием db_browse.pl можно воспользоваться для создания и более универсальных сценариев. Его можно видоизменить для работы с несколькими базами данных. Например, с помощью сценария можно просматривать список баз данных, имеющихся на сервере, и выбрать из этого списка базу данных для дальнейшего подключения к ней и просмотра ее таблиц.

В начале этого раздела уже упоминалось о том, что сценарий db_browse.pl можно легко приспособить для отображения любой таблицы, доступной для учетной записи sampadm, посредством которой сценарий подключается к серверу MySQL.

Предположим, что эта учетная запись имеет право доступа не только к базе данных sampdb, но и к базе данных hr, которая содержит кадровую информацию компании. Это приводит к нарушению системы безопасности.

Для этого сценарий db_browse.pl включает имена таблиц, которые предполагается отыскать в базе данных sampdb, но нет ничего, что могло бы остановить злоумышленника, непосредственно отсылающего запросы, использующие аналогичный URL, который содержит перечень таблиц из базы данных hr:

```
http://www.snake.net/cgi-bin/db_browse.pl?tbl_name=hr.employee
```

В этом случае сценарий подключается к СУБД MySQL и делает базу данных sampdb базой данных по умолчанию, но оператор SELECT, который он создаст, обратится напрямую к таблице базы данных hr:

```
SELECT * FROM hr.employee ORDER BY 1
```

В результате сценарий db_browse.pl предоставляет содержимое таблиц, содержащих закрытые данные, любому, кто имеет доступ к вашему Web-серверу.

Броузер проекта "Учет успеваемости"

Наш следующий Web-сценарий, score_browse.pl, разработан для отображения оценок, которые были сохранены в проекте "Учет успеваемости". Строго говоря, сначала необходимо разработать способ ввода оценок, а затем разрабатывать способ их выборки. Но не будем спешить — собственно сценарий ввода оценок будет описан в следующей главе. А пока воспользуемся наборами оценок за начальный оценочный период, которые были занесены нами в базу данных ранее. Этот сценарий можно использовать для отображения этих оценок даже при отсутствии удобного метода ввода

оценок. Данный сценарий отображает упорядоченный список оценок, полученных учащимися за тесты или викторины, которые полезны для определения кривой оценок и присвоения аттестационных оценок.

Наш сценарий просмотра оценок score_browse.pl имеет много общего со сценарием db_browse.pl (оба работают как информационные броузеры), но при этом он предназначен для решения более специфической задачи просмотра оценок, поставленных за определенную викторину или тест. Начальная страница отображает весь перечень экзаменов, из которого необходимо делать выбор, и позволяет выбрать любой из них, чтобы просмотреть оценки по данному экзамену. Результаты сортируются по убыванию. Это позволяет легко построить оценочную кривую. Сценарий score_browse.pl проверяет только один параметр — event_id. Если конкретный экзамен не задан, сценарий score_browse.pl отображает строки таблицы event. Отсюда пользователь может выбирать. В противном случае он отображает результаты, связанные с выбранным экзаменом:

```
# ... установить соединение с базой данных (не отражено) ...
# вывести начальную часть страницы
my $title = "Проект "Учет успеваемости" - броузер оценок";
print header ();
print start_html (-title => $title, -bgcolor => "white");
print h1 ($title);
# параметр, задающий экзамен, оценки которого необходимо отобразить
my $event_id = param ("event_id");
# если переменная $event_id не имеет значения,
# отобразить перечень экзаменов.
# в противном случае отобразить оценки заданного экзамена
if (!defined ($event_id))
{
  display_events ($dbh)
}
else
{
  display_scores ($dbh, $event_id);
}
print end_html ();
```

Функция display_events() выбирает информацию из таблицы event и отображает ее в табличной форме, используя имена столбцов в качестве заголовков. Значение event_id играет роль ссылки, запускающей выполнение запроса по поиску оценок, соответствующих данному экзамену. URL каждого экзамена просто является путем к сценарию score_browse.pl с присоединенным параметром, определяющим номер экзамена:

```
/cgi-bin/score_browse.pl?event_id=n
```

Функция display_events() написана следующим образом:

```
sub display_events
{
my $dbh = shift;
my @rows;
my @cells;
  print p ("Выберите экзамен, щелкнув на его номере: ");
  # получить список экзаменов
  my $sth = $dbh->prepare (qq{
    SELECT event_id, date, type
```

```
    FROM event
    ORDER BY event_id
  });
  $sth->execute ();
  # имена столбцов используются как заголовки
  for (my $i = 0; $i < $sth->{NUM_OF_FIELDS}; $i++)
  {
    push (@cells, th (escapeHTML ($sth->{NAME}->[$i])));
  }
  push (@rows, Tr (@cells));
  # отобразить информацию по каждому экзамену как строку таблицы
  while (my ($event_id, $date, $type) = $sth-fetchrow_array ())
  {
    @cells = ();
    # Связать идентификатор каждого события со ссылкой,
    # которая покажет оценки по экзамену
    my $url = sprintf ("%s?event_id=%s", url (), escape ($event_id));
    my $link = a ({-href => $url}, escapeHTML ($event_id));
        push (@cells, td ($link));
        # отобразить дату и тип экзамена
        push (@cells, td (escapeHTML ($date)));
        push (@cells, td (escapeHTML ($type)));
        push (@rows, Tr (@cells));
    }
    # отобразить таблицу в рамке
    print table ({-border => "1"}, @rows);
}
```

Последовательность работы такова: пользователь выбирает экзамен, броузер посылает запрос с выбранным значением идентификатора экзамена сценарию `score_browse.pl`. Сценарий `score_browse.pl` находит набор данных по параметру `event_id` и вызывает функцию `display_scores()` для получения списка оценок заданного экзамена. Страница также имеет ссылку "Список экзаменов" на первую страницу, что позволяет вернуться на нее. URL этой ссылки ссылается на сценарий `score_browse.pl`, но не определяет значения параметра `event_id`. Функция `display_scores()` представлена в следующей распечатке:

```
sub display_scores
{
my ($dbh, $event_id) = @_;
my @rows;
my @cells;
   # URL без параметра event_id
   # вызовет отображение списка экзаменов.
   print p (a ({-href => url ()}, "Список экзаменов"));
   # select scores for the given event
   my $sth = $dbh->prepare (qq{
     SELECT
         student.name, event.date, score.score, event.type
     FROM
         student, score, event
     WHERE
         student.student_id = score.student_id
         AND score.event_id = event.event_id
         AND event.event_id = ?
     ORDER BY
```

```
        event.date ASC, event.type ASC, score.score DESC
    });
    $sth->execute ($event_id); # передать идентификатор экзамена в запрос
    print p (strong ("Оценки по экзамену $event_id"));
    # использовать имена столбцов в качестве заголовков
    for (my $i = 0; $i < $sth->{NUM_OF_FIELDS}; $i++)
    {
      push (@cells, th (escapeHTML ($sth->{NAME}->[$i])));
    }
    push (@rows, Tr (@cells));
    while (my @ary = $sth->fetchrow_array ())
    {
      @cells = ();
      foreach my $val (@ary)
      {
          # отобразить значение, а если пустое - неразрывный пробел
          if (defined ($val) && $val ne "")
          {
              $val = escapeHTML ($val);
          }
          else
          {
              $val = "  ";
          }
          push (@cells, td ($val));
      }
      push (@rows, Tr (@cells));
    }
    # отобразить таблицу в рамке
    print table ({-border => "1" @rows);
}
```

Создаваемый сценарием `display_scores()` запрос во многом аналогичен запросу, разработанному нами в главе 1, "Знакомство с СУБД MySQL и SQL", в которой объясняется процедура создания объединения. Там мы запрашивали оценки для заданных дат, так как даты более информативны, чем идентификаторы экзаменов. С другой стороны, использование сценария `score_browse.pl` позволяет нам узнать точный идентификатор экзамена. И совсем не потому, что мы мыслим в терминах идентификаторов экзаменов (это не так), а потому, что этот сценарий снабжает нас списком для их выбора. Очевидно, что этот тип интерфейса уменьшает необходимость в конкретных деталях. Совсем не надо знать идентификатор события, нам только нужно распознать необходимый нам экзамен.

Подбор членов "Исторической Лиги" по общности научных интересов

Сценарии `db_browse.pl` и `score_browse.pl` позволяют пользователю делать осознанный выбор из списка на начальной странице, где каждый элемент выбора является ссылкой, вызывающей сценарий с определенным значением параметра. Существует еще один способ сделать выбор: сделать это в форме страницы, содержащей редактируемое поле. Это уместно в том случае, когда диапазон возможных значений не ограничивается достаточно просто определяемым набором значений. Наш следующий сценарий демонстрирует метод запроса таких значений.

В разделе "Подготовка интерфейса к работе" был создан сценарий `interests.pl`, работающий из командной строки, главной задачей которого являлся подбор членов "Исторической Лиги" по их научным интересам. Но сами члены "Исторической Лиги" не могут иметь доступа к такому сценарию. Принцип работы такого сценария заключается в другом: секретарь сам запускает этот сценарий, а затем отсылает по почте результат члену "Исторической Лиги", запросившему такой список. Конечно, гораздо лучше сделать эту возможность более доступной, чтобы члены этой организации сами могли производить такой выбор. И единственный способ — создать соответствующий Web-сценарий. Первый базируется на поиске по шаблону, а второй использует поисковые возможности индекса FULLTEXT, имеющегося в СУБД MySQL.

Поиск с использованием соответствия по шаблону

Первый сценарий поиска, `ushl_browse.pl`, отображает форму, в которую пользователь должен ввести ключевое слово. Когда пользователь запускает форму, сценарий повторно вызывается для поиска имен в таблице `member` и отображения результатов. Поиск производится за счет добавления групповых символов '`%`' с обеих сторон ключевого слова и выполнения поиска по шаблону с помощью команды `LIKE`, которая ищет записи, содержащие ключевое слово в любом месте столбца `interests`. Основная часть сценария отображает форму с ключевым словом. Он также проверяет задание ключевого слова и выполняет поиск:

```
my $title = "Поиск по интересам в "Исторической Лиге США";
print header ();
print start_html (-title => $title, -bgcolor => "white");
print h1 ($title);
# parameter to look for
my $keyword = param ("keyword")
# Вывод формы ввода ключевого слова. Кроме того, если задано $keyword,
# произвести поиск и отобразить список членов, которые имеют такой же
научный интерес.
print start_form (-method => "POST")
print p ("Ввести ключевое слово для поиска: ");
print textfield (-name => "keyword", -value => "" -size => 40);
print submit (-name => "button", -value => "Search"
print end_form ();
# соединиться с сервером и запустить поиск, если ключевое слово было
задано
if (defined ($keyword) && $keyword !~ /^\s*$/)
{
   # ... установить соединение с базой данных (не показано) ...
   search_members ($dbh, $keyword);
   # ... отключиться от базы данных (не показано) ...
}
```

Этот сценарий передает информацию самому себе иначе, чем это делают сценарии `db_browse.pl` и `score_browse.pl`. Параметр `interest` не добавляется в конец URL, а информация, полученная из формы, кодируется броузером и передается как часть запроса POST. Однако модулю CGI.pm безразлично, каким образом посылается информация, значение параметра по-прежнему можно получить с помощью вызова `param()`.

Поиск по ключу выполняется функцией `search_members()`. Он принимает в качестве аргументов дескриптор базы данных и ключевое слово, после этого выполняет запрос и выводит список членов, соответствующих искомому ключу:

```
sub search_members
{
my ($dbh, $interest) = @_;
   print p ("Поиск результатов по ключу: " . escapeHTML ($interest));
            my $sth = $dbh->prepare (qq{
            SELECT * FROM member WHERE interests LIKE ?
            ORDER BY last_name, first_name
         });
   # поиск строки как подстроки поля interest
   $sth->execute ("%" . $interest . "%");
   my $count = 0;
   while (my $ref = $sth->fetchrow_hashref ())
   {
     format_html_entry ($ref);
     ++$count;
   }
   print p ("Найдено $count записей");
}
```

Запустив сценарий `ushl_browse.pl`, вы определенно заметите, что каждый раз после ввода ключевое значение будет отображено в форме на следующей странице. Это происходит даже тогда, когда сценарий задает при генерировании формы в качестве значения `keyword` пустую строку. Причина заключается в том, что модуль CGI.pm автоматически заполняет поля формы значениями, которые он берет в среде выполнения сценария, если они заданы. Для того чтобы преодолеть такое поведение программы и оставить это поле пустым, в вызов функции `textfield()` необходимо добавить параметр `override`:

```
print textfield (-name => "keyword",
                                -value => "",
                                -override => 1,
                                -size => 40);
```

Функция `search_members()` для отображения отдельных записей использует вспомогательную функцию `format_html_entry()`. Эта функция во многом аналогична функции с тем же именем, которая была написана ранее для сценария `gen_dir.pl` (см. раздел "Создание каталога общества "Историческая Лига"). Однако если ранняя версия функции генерировала HTML-формат с непосредственной печатью тэгов, версия, использованная сценарием `ushl_browse.pl` для создания HTML-тэгов, использует функции модуля CGI.pm:

```
sub format_html_entry
{
my $entry_ref = shift;
   # закодировать специальные символы HTML
   foreach my $key (keys (%{$entry_ref}))
   {
     next unless defined ($entry_ref->{$key});
     $entry_ref->{$key} = escapeHTML ($entry_ref->{$key});
   }
   print strong ("Имя: " . format_name ($entry_ref)) . br ();
   my $address = "";
```

```
    $address .= $entry_ref->{street}
    if defined ($entry_ref->{street});
    $address .= ", " . $entry_ref->{city}
    if defined ($entry_ref->{city});
    $address .= ", " . $entry_ref->{state}
    if defined ($entry_ref->{state});
    $address .= " " . $entry_ref->{zip}
    if defined ($entry_ref->{zip});
    print "Адрес: $address" . br ()
    if $address ne "";
    print "Телефон: $entry_ref->{phone}" . br ()
    if defined ($entry_ref->{phone});
    print "Э/почта: $entry_ref->{email}" . br ()
    if defined ($entry_ref->{email});
    print "Интересы: $entry_ref->{interests}" . br ()
    if defined ($entry_ref->{interests});
    print br ();
}
```

Для "склеивания" значений столбцов `first_name`, `last_name` и `suffix` функция `format_html_entry()` использует функцию `format_name()`. Она полностью аналогична такой же функции, имеющейся в сценарии `gen_dir.pl`.

Выполнение поиска с использованием индекса FULLTEXT

Члены "Исторической Лиги" могут иметь интересы сразу в нескольких областях. При этом в столбце `interests` таблицы `member` они отделяются запятыми, например:

```
Revolutionary War,Spanish-American War,Colonial period,Gold
rush,Lincoln
```

Подходит ли сценарий `ushl_browse.pl` для поиска записей, которые отражают сразу несколько интересов? Да, но не совсем. Можно ввести несколько слов в поисковую форму, но записи не будут найдены, если слова не разделить запятыми или если в записях интересы не перечислены в порядке, в котором они указываются при поиске. Более гибкий подход к решению задачи поиска можно осуществить при использовании индекса FULLTEXT[2]. В этом разделе этот тип поиска показан на примере сценария `ushl_ft_browse.pl`. Это совсем не трудно.

Требования к использованию `ushl_ft_browse.pl` сводятся к тому, что в вашем распоряжении должна быть СУБД MySQL версии 3.23.23 или выше и таблица `member` должна иметь тип MyISAM. Если таблица `member` уже имеет какой-то другой тип, его можно преобразовать с помощью оператора `ALTER TABLE`:

```
mysql> ALTER TABLE member TYPE = MYISAM;
```

Затем таблицу `member` необходимо соответствующим образом проиндексировать. Для этого воспользуйтесь оператором

```
mysql> ALTER TABLE member ADD FULLTEXT (interests);
```

Это позволит использовать столбец `interests` при полнотекстном поиске. Сценарий `ushl_ft_browse.pl`, имеющийся в дистрибутиве `sampdb`, опирается на сценарий `ushl_browse.pl` и отличается от него только функцией

[2] Возможности типа данных FULLTEXT описываются в главе 3.

search_members(), которая составляет поисковый запрос. Видоизмененная версия функции выглядит следующим образом:

```
sub search_members
{
my ($dbh, $interest) = @_;
  print p ("Поиск результатов по ключу: " . escapeHTML ($interest));
  my $sth = $dbh->prepare (qq{
            SELECT * FROM member WHERE MATCH(interests) AGAINST(?)
            ORDER BY last_name, first_name
  });
  # поиск строки как подстроки поля interest
  $sth->execute ($interest);
  my $count = 0;
  while (my $ref = $sth->fetchrow_hashref ())
  {
    format_html_entry ($ref);
    ++$count;
  }
  print p ("Найдено $count записей ");
}
```

Эта версия функции search_members() имеет следующие отличия от предыдущей:

- в запросе конструкция LIKE заменена конструкцией MATCH() ...AGAINST();
- групповые символы "%" не добавляются.

После внесения этих изменений можно вызывать сценарий ushl_ft_browse.pl из своего Web-броузера и вводить в поисковую форму сразу несколько ключевых слов (с запятыми или без них). Сценарий найдет записи, соответствующие любой из них.

Конечно, этот сценарий позволяет реализовать и более интересные варианты поиска. Одним из них является возможность извлечь преимущество из того факта, что поиск типа FULLTEXT может одновременно производить поиск в нескольких столбцах, включив в индекс сразу несколько столбцов и после этого модифицируя сценарий ushl_ft_browse.pl для осуществления поиска по всем ним. Например, можно удалить первоначальный индекс FULLTEXT и добавить другой, в котором кроме столбца interests используются столбцы last_name и full_name:

```
mysql> ALTER TABLE member DROP INDEX interests;
mysql> ALTER TABLE member ADD FULLTEXT
(interests,last_name,first_name);
```

Для того чтобы использовать новый индекс, в запрос SELECT в функции search_members() потребуется внести изменения и заменить конструкцию MATCH(interests) конструкцией MATCH(interests,last_name,first_name).

Другое изменение, которое необходимо внести в сценарий ushl_ft_browse.pl, — это добавить в форму несколько радиокнопок, чтобы позволить пользователю сделать выбор между режимами "Выбрать любое ключевое слово" и "Выбрать все ключевые слова". Первый режим используется сценарием в данный момент. Для того чтобы заработал режим "Выбрать все ключевые слова", запрос необходимо настроить на использование типа IN BOOLEAN MODE поиска

FULLTEXT и предварить каждое ключевое слово символом '+', чтобы он присутствовал в найденных записях. Подробнее этот материал изложен в главе 3, "Синтаксис и использование языка SQL".

8

Интерфейс API для языка написания сценариев PHP

В этой главе...

Основы языка PHP
Работа с PHP

PHP – это язык написания Web-страниц, содержащих код, который исполняется во время обращения к странице. Кроме того, этот код в состоянии генерировать динамическое содержание как часть результата, посылаемого на Web-броузер клиента. В этой главе описываются принципы использования языка PHP при создании Web-приложений, работающих с СУБД MySQL. Для сравнения программного интерфейса API, разработанного для языка PHP, с интерфейсами DBI API, созданными для языков программирования C и Perl, обратитесь к главе 5, "Введение в программирование в СУБД MySQL".

Все примеры, приведенные в этой главе, используют хорошо известную тестовую базу данных `sampdb`. При этом используются таблицы, созданные нами для проектов "Учет успеваемости" и "Исторической Лиги" в главе 1, "Знакомство с СУБД MySQL и SQL". При этом мы будем работать с языком PHP 3 или PHP 4, хотя из соображений производительности и безопасности рекомендуется использовать PHP 4.

В данной главе предполагается, что программирование на PHP производится одновременно с работой Web-сервера Apache. При этом также должны быть установлены клиентская библиотека MySQL C и файлы заголовков. Эти файлы нам понадобятся при построении PHP. В противном случае PHP не сможет получить доступ к базам данных MySQL. Если у вас такое программное обеспечение отсутствует, его можно получить в соответствии с инструкциями, изложенными в при-

ложении А, "Получение и инсталляция программного обеспечения". Кроме всего прочего, там вы найдете инструкции по получению текстов сценариев, разработанных в этой главе. Их можно будет загрузить и не заниматься рутинным вводом текстов этих сценариев с клавиатуры. Сценарии, задействованные в этой главе, можно найти в каталоге phpapi этого дистрибутива.

При работе под управлением ОС UNIX PHP может функционировать с сервером Apache как встроенный модуль, связанный с двоичным исполняемым кодом сервера, или в виде автономного интерпретатора, работающего как традиционная программа CGI. При работе под управлением ОС Windows язык PHP может функционировать только в качестве автономной программы, если речь не идет о PHP 4 и Apache 2.x под управлением ОС Windows NT. На любой платформе предпочтительнее работа PHP в виде модуля, поскольку это дает выигрыш в производительности.

В основном в этой главе функции языка PHP описываются по мере их необходимости в излагаемом материале. Более полный перечень функций языка PHP, имеющих отношение к интерфейсу с СУБД MySQL, можно найти в приложении 3, "Программный интерфейс языка PHP". Вероятно, потребуется обратиться и к руководству по языку PHP, содержащему полное описание функций языка, включая даже те функции, которые используют другие СУБД. Это руководство можно найти на Web-узле по адресу http://www.php.net/. В качестве еще одного полезного источника информации можно назвать книгу *PHP Functions Essential Reference*, Грант и др. (New Riders, 2002).

Имена файлов для сценариев PHP обычно имеют расширение, которое легко позволяет Web-серверу распознать их и запустить интерпретатор PHP для их выполнения. При использовании других расширений PHP-сценарии будут обработаны как текстовые файлы. В этой главе используется расширение .php. Другим часто используемым расширением является .phtml. Как настроить сервер Apache, чтобы он распознавал эти расширения, рассказывается в приложении А, "Получение и инсталляция программного обеспечения". Если управление инсталляцией сервера Apache не входит в круг ваших прав и обязанностей, проконсультируйтесь с системным администратором, каким расширением вам будет удобней воспользоваться.

Для того чтобы разработанные здесь сценарии заработали, потребуется установить их в каталоге, в котором ваш Web-сервер может иметь к ним доступ. В этой главе будем придерживаться соглашения, в соответствии с которым проекты "Историческая Лига США" и "Учет успеваемости" имеют отдельные каталоги ushl и gp соответственно на уровне дерева документов Web-сервера Apache. Для того чтобы ваш Web-сервер мог работать таким образом, необходимо создать эти каталоги. Рядом с именем Web-узла www.snake.net страницы, размещенные в этих каталогах, будут иметь URL, которые начинаются следующим образом:

```
http://www.snake.net/ushl/...
http://www.snake.net/gp/...
```

Например, главные страницы во всех каталогах могут иметь имя index.php и будут доступными командам:

```
http://www.snake.net/ushl/index.php
http://www.snake.net/gp/index.php
```

При настройке Web-сервера Apache таким образом, чтобы в качестве стандартной страницы использовалась index.php, следующие URL будут эквивалентны предыдущим:

```
http://www.snake.net/ush1/
http://www.snake.net/gp/
```

Основы языка PHP

Основной задачей языка PHP является интерпретация сценария для генерации Web-страницы, которая отсылается программе-клиенту. Сценарий содержит HTML-код, который высылается в литеральном представлении, и PHP-код, который выполняется как программа. Клиентской программе посылаются результаты работы кода. Таким образом, клиент никогда не видит самого кода, он видит только результат его работы[1].

Начав чтение файла, PHP копирует его содержимое с допущением, что все содержимое файла представляет собой литеральный HTML-код. Когда интерпретатор PHP находит тэг открытия, он переключается из режима HTML в режим интерпретации кода PHP и начинает интерпретировать файл как исполнимый код PHP. Окончание кодовой части идентифицируется другим специальным тэгом, встретив который интерпретатор переключается обратно в режим HTML. Это позволяет объединять статический HTML-текст (HTML-часть) с динамически генерируемыми результатами (результат выполнения части PHP-кода) для генерации страниц, содержимое которых изменяется в зависимости от условий, при которых они активизируются.

Например, сценарий PHP можно использовать для обработки результата ввода формы, в которую пользователь ввел параметры для осуществления поиска в базе данных. Параметры поиска могут варьироваться в зависимости от заполнения формы, таким образом, каждый раз, когда сценарий осуществляет поиск, результирующая страница будет отображать разные результаты.

Посмотрим, как это работает на примере простейшего сценария PHP:

```
<html>
<body>
<p>hello, world</p>
</body>
</html>
```

Этот сценарий не представляет особого интереса, так как он не содержит PHP-кода. Что же тогда есть примечательного в этом сценарии? Это резонный вопрос. Ответ заключается в том, что иногда очень полезно начать с создания простой заготовки, содержащей только HTML-код, а потом, по мере необходимости, добавлять в нее код PHP. Этот метод имеет "право на жизнь", ведь интерпретатор PHP работает безошибочно и с таким кодом.

Для включения кода PHP его необходимо отличить от окружающего его текста. Для этого применяются два специальных тэга: "<?php" для обозначения начала сце-

---

[1] PHP-сценарии, разработанные в этой главе, генерируют страницы, которые могут работать не только как HTML-страницы, но и как XHTML-страницы. Краткое описание XHTML-формата содержится в приложении B, "Операторы и функции".

нария и "?>" для завершения сценария. Когда интерпретатор PHP обнаруживает открывающий тэг "<?php", он переключается из режима HTML в режим PHP и интерпретирует все, что попадается на его пути, как код PHP, до тех пор, пока не встретится закрывающий тэг "?>". Текст сценария, находящийся между тэгами, определенным образом интерпретируется и замещается результатом его работы. Предыдущий пример можно переписать таким образом, чтобы он включал небольшой код PHP:

```
<html>
<body>
<p><?php print ("hello, world"); ?></p>
?>
</body>
</html>
```

В этом случае кодовая часть минимальна и содержит одну строку. Интерпретация этого кода генерирует ставший классикой текст `hello, world`. Он и будет послан на броузер клиента. Таким образом, Web-страница, генерируемая этим сценарием, идентична результату, сгенерированному кодом, приведенным в предыдущем примере, в котором сценарий состоял только из HTML-кода.

Код PHP можно применять для генерирования любой части Web-страницы. Мы уже приводили пример, в котором полностью отсутствовала PHP-часть, где весь сценарий состоял только из литерального HTML-кода. Вот другая крайность — пример, когда есть только PHP-код и полностью отсутствует литеральная HTML-часть:

```
<?php
print ("<html>\n");
print ("<body>\n");
print ("<p>hello, world</p>\n");
print ("</body>\n");
print ("</html>\n");
?>
```

Это свидетельствует о том, что PHP предоставляет возможность оперировать методами получения результатов: как их применять и в какой пропорции прибегать к помощи кодов HTML и PHP. Язык PHP проявляет свою гибкость еще и в том, что отсутствует требование размещать весь код компактно в одном месте. Таким образом, переключение между кодами HTML и PHP может производиться многократно.

PHP разрешает применение стилей тэгов, отличных от <?php и ?>, которые и используются в примерах, приведенных в этой главе. Описание различных стилей написания тэгов и инициализирующих их команд содержится в приложении 3, "Программный интерфейс языка PHP".

Автономные сценарии PHP

Примеры сценариев в этой главе написаны в расчете на то, что они вызываются Web-сервером для создания Web-страниц. Существует возможность создавать автономные сценарии PHP, вызываемые из командной строки, так как это делается со сценарием оболочки или сценарием Perl. Например:

```
<?php print ("hello, world\n"); ?>
```

Чтобы выполнить сценарий из командной строки, воспользуйтесь следующей командой:

```
% php -q hello.php
hello, world
```

Иногда очень полезно работать со сценарием непосредственно, поскольку в таком случае вы можете увидеть, что в нем есть синтаксические ошибки или другие неточности, без необходимости вызова сценария из броузера после каждого внесенного изменения. (По этой причине может потребоваться использовать автономную версию PHP, даже если обычно он вызывается как модуль из Web-сервера Apache.)

Этот сценарий можно сделать непосредственно исполнимым (аналогично сценарию оболочки или Perl), добавив в начало строку "#!", которая задает путь к PHP. Предположим, что PHP установлен в каталоге /usr/local/bin. Сценарий можно модифицировать следующим образом:

```
#! /usr/local/bin/php -q
<?php print ("hello, world\n"); ?>
```

Сделайте его исполнимым с помощью команды chmod +x и вызовите его следующим образом:

```
% chmod +x hello.php
% ./hello.php
hello, world
```

Следующие сценарии немного более функциональны, но все же по-прежнему относительно коротки. Они демонстрируют легкость, с которой PHP позволяет осуществлять доступ к СУБД MySQL и отображать результаты запросов на Web-странице. Вот сценарий, который был очень кратко представлен в главе 5, "Введение в программирование в СУБД MySQL". Он играет роль простой заготовки для создания Web-узла "Исторической Лиги". По мере повествования этот сценарий будет усложняться, но пока в его функции входит печать короткого приветствия и список членов "Исторической Лиги" на данный момент:

```
<html>
<head>
<title>"Историческая Лига США"</title>
</head>
<body bgcolor="white">
<p> Добро пожаловать на Web-узел "Исторической Лиги".</p>
<?php
# Главная страница "Исторической Лиги США"
$conn_id = @mysql_connect ("cobra.snake.net", "sampadm", "secret")
  or exit ();
mysql_select_db ("sampdb")
  or exit ();
$result_id = mysql_query ("SELECT COUNT(*) FROM member")
  or exit ();
if ($row = mysql_fetch_row ($result_id))
  print ("<p> В настоящее время "Историческая Лига" насчитывает " .
$row[0] . " членов. </p>");
mysql_free_result ($result_id);
?>
</body>
</html>
```

Текст приветствия является статическим текстом, поэтому его легче всего отобразить литералом в формате HTML. Однако счетчик членов динамический и его значение время от времени изменяются, таким образом, он должен вычисляться на лету после запроса к таблице member из базы данных sampdb.

Текст самой программы, расположенный между открывающим и закрывающим тэгом, выполняет простую задачу.

1. Осуществляет соединение с сервером MySQL и делает базу данных sampdb базой данных по умолчанию.

2. Отсылает запрос на сервер, чтобы определить, сколько членов насчитывает "Историческая Лига" на данный момент (который мы получаем как количество строк в таблице member).
3. Результат запроса отображается как часть сообщения, содержащего данные о количестве членов.

В случае возникновения ошибки в любом месте сценария на протяжении процесса его выполнения он просто завершает свое выполнение безо всякой диагностики[2]. Этот сценарий, index.php, находится в каталоге phpapi/ushl дистрибутива sampdb. По необходимости внесите изменения в параметры соединения, поместите копию файла index.php в каталог ushl дерева документов вашего Web-сервера и запросите его из своего броузера с помощью одного из следующих URL:

```
http://www.snake.net/ushl/
http://www.snake.net/ushl/index.php
```

Разобьем сценарий на отдельные части. Это поможет проверить правильность их работы. Первым шагом является подключение к серверу с помощью функции mysql_pconnect():

```
$conn_id = @mysql_connect ("cobra.snake.net", "sampadm", "secret")
  or exit ();
```

Функция mysql_pconnect() принимает в качестве аргументов имена узла и пользователя, а также пароль пользователя. Она возвращает идентификатор связи при успешном соединении и значение "FALSE" – в случае ошибки. Когда попытка соединения оказывается неудачной, сценарий вызывает функцию exit(), которая немедленно завершает работу сценария. Никакой информации больше не выдается.

Вас может насторожить выставление имени и пароля в этом сценарии на всеобщее обозрение. Да, это так. Имя и пароль не появляются на Web-странице, которая отсылается клиенту, так как содержимое сценария замещается его же выводом. Тем не менее, если в конфигурации Web-сервера есть ошибки или он почему-то не распознает, что ваш сценарий должен быть обработан интерпретатором PHP, он передаст весь сценарий как простой текст, и параметры вашего соединения будут выставлены на всеобщее обозрение. Об этом мы поговорим вкратце ниже, в разделе "Работа с функциями и включаемыми файлами".

Что означает символ @, расположенный перед вызовом функции mysql_pconnect()? Это оператор "Shut up, please" ("Замолчите, пожалуйста"). Одни функции PHP в случае сбоя, кроме кода состояния, выдают сообщение об ошибке. В случае с функцией mysql_pconnect(), если при соединении произошел сбой, то на броузер клиента отсылается сообщение следующего содержания:

```
Warning: MySQL Connection Failed: Access denied for user:
'sampadm@cobra.snake.net' (Using password: YES)
```

---

[2] При создании всей страницы с помощью кода PHP такое поведение сценария, как выход по ошибке без диагностирования возникшей ситуации, будет раздражать посетителей будущей Web-страницы, так как некоторые броузеры в этом случае выводят диалоговое окно "this page contained no data" ("эта страница не содержит данных"), которое в данном случае будет отсутствовать. В этом случае лучше отобразить страницу, содержащую как минимум сообщение о том, что запрос не может быть обработан.

```
Внимание: неудачное соединение с MySQL. Отказ в доступе для пользователя:
sampadm@cobra.snake.net' (Использование пароля: ДА)
```

Сообщение — "неприглядное", и посетитель вашего узла может не понять, что же ему следует предпринять, когда он его увидит. Размещение символа @ перед вызовом функции `mysql_pconnect()` подавляет вывод этого сообщения. Таким образом, это позволяет программировать обработку сбойных ситуаций собственноручно, пользуясь только возвращенным значением. В нашем случае лучше всего вообще не делать никакого вывода, относящегося к счетчику количества членов "Исторической Лиги". В таком случае страница будет содержать только приветствие.

Символ @ можно использовать с любой функцией PHP, но мой опыт подсказывает, что первоначальный вызов функции `mysql_pconnect()` является самым вероятным кандидатом на сбой, следовательно, во всех примерах данной главы подавляется диагностика только этих функций.

Сравнение функций `mysql_pconnect()` и `mysql_connect()`

Существует функция, аналогичная функции `mysql_pconnect()`, это — `mysql_connect()`. Обе принимают в качестве аргументов имена узла и пользователя и пароль пользователя, а возвращают идентификатор связи или логическое значение "FALSE". Разница между ними заключается в том, что функция *`mysql_pconnect()` устанавливает устойчивое соединение*, а `mysql_connect()` — неустойчивое. Устойчивое соединение отличается от неустойчивого тем, что оно не закрывается после завершения работы сценария. Если другой PHP-сценарий будет вызван тем же порожденным процессом сервера Apache и при этом он вызовет функцию `mysql_pconnect()` с аналогичными аргументами, это же устойчивое соединение будет использовано повторно. Это более эффективно, чем снова устанавливать соединение. В действительности использование устойчивых соединений может иметь некоторые отрицательные эффекты. На перегруженных Web-узлах, поддерживающих работу сразу нескольких PHP-сценариев, может случиться так, что все процессы Web-сервера Apache держат открытыми так много соединений с сервером MySQL, что все имеющиеся в наличии разъемы становятся задействованными. Эту проблему можно решить, изменив значение переменной сервера `max_connections` (см. раздел "Настройка сервера" в главе 11, "Общее администрирование MySQL"), но есть еще один выход — использовать неустойчивые соединения.

Возвращаемый функцией `mysql_pconnect()` идентификатор связи может быть передан некоторым другим вызовам, работающим с MySQL. Но для таких вызовов идентификатор будет всегда необязательным параметром. Например, функцию `mysql_select_db()` можно вызвать, воспользовавшись одним из следующих форматов:

```
ysql_select_db ($db_name, $ conn_id);
ysql_select_db ($db_name);
```

Аргумент связи можно и опустить. В этом случае будет использовано самое последнее соединение. Таким образом, если сценарий открывает только одно соединение, в обращениях к СУБД MySQL нет необходимости указывать явным образом аргумент соединения — это соединение становится соединением по умолчанию. Это совсем не похоже на программирование интерфейса MySQL с помощью C API или DBI API, где такой механизм не предусмотрен.

Здесь соединение запрограммировано с помощью переменной `$conn_id`, чтобы однозначно показать, какое значение возвращает функция `mysql_connect()`:

```
conn_id = @mysql_connect ("cobra.snake.net", "sampadm", "secret")
  or exit ();
```

Однако в дальнейшем переменная $conn_id в сценарии нигде не используется, что позволяет переписать эту строку так:

```
@mysql_connect ("cobra.snake.net", "sampadm", "secret")
  or exit ();
```

Предположим, что если соединение было успешным, то следующим шагом будет выбор базы данных:

```
mysql_select_db ("sampdb")
  or exit ();
```

Если функция mysql_select_db() выполняется с ошибкой, сценарий завершает свою работу без вывода диагностического сообщения. Но на этой стадии ошибка маловероятна, так как мы уже подключились к серверу и база данных существует, но все равно никогда не помешает проверить исходный текст на наличие ошибок и предпринять соответствующие меры. После того как база данных выбрана, на сервер можно послать запрос, выбрать результат и очистить результирующий набор:

```
$result_id = mysql_query ("SELECT COUNT(*) FROM member")
  or exit ();
if ($row = mysql_fetch_array ($result_id))
  print "<p> В настоящее время Историческая Лига насчитывает " .
$row[0] . " членов. </p>";
mysql_free_result ($result);
```

Функция mysql_query() отсылает запрос на сервер на выполнение. Запрос не надо завершать точкой с запятой или "\g". Функция mysql_query() возвращает значение "FALSE", если запрос содержит ошибку или по какой-то причине не может быть выполнен. В противном случае она возвращает идентификатор результирующего набора. Это значение, которое можно использовать для получения информации о результирующем наборе. В нашем запросе результирующий набор состоит из одной строки с одним столбцом, содержащим счетчик членов. Для того чтобы получить это значение, идентификатор результирующего набора передается для выборки строки функции mysql_fetch_row(), строка присваивается переменной $row. Доступ к первому элементу строки (который в нашем случае является и единственным элементом) осуществляется как $row[0].

После того как все операции с результирующим набором будут завершены, его очищают с помощью функции mysql_free_result(). Вообще-то, вызов этой функции здесь не очень нужен, так как PHP автоматически очищает активный результирующий набор по завершении работы сценария. Функция mysql_free_result() имеет смысл прежде всего в сценариях, обрабатывающих очень большие запросы или большое количество запросов. Это предотвращает перерасход памяти.

Переменные языка PHP

В языке PHP не надо объявлять переменные. В нашем сценарии используются три переменные: $conn_id, $result_id и $row. Ни одна из них нигде не была объявлена. (Конечно, в определенных контекстах объявления переменной не избежать. Таким примером может служить ссылка на глобальную переменную внутри функции, но об этом — позже.)

Переменные обозначаются идентификаторами с предшествующим знаком доллара ($). Это справедливо для всех типов переменных, хотя для массивов приходится задавать дополнительную информацию, чтобы получить доступ к отдельным элементам. Если переменная $x

представляет одно-единственное значение, такое как число или строка, получить ее можно с помощью выражения $x. Но если $x является массивом с числовыми индексами, доступ к его элементам можно получить с помощью выражений $x[0], $x[1] и т.д. Если $x является массивом с ассоциативными индексами, такими как "yellow" или "large", получить доступ к его элементам можно с помощью выражения $x["yellow"] или $x["large"]. (Массивы PHP могут также иметь смешанные элементы, как числовые, так и ассоциативные.) Например, $x[1] и $x["large"] могут быть одновременно членами одного и того же массива. Если $x представляет собой объект, к его свойствам можно получить доступ с помощью выражения $x->property_name. Например, $x->yellow и $x->large являются свойствами $x. В качестве имен свойств числа недопустимы. Так, конструкция $x->1 в PHP недопустима.

Заимствования языка PHP

Если вы имеете опыт программирования на языке C, то, вероятно, заметили, что многие синтаксические конструкции в нашем сценарии очень похожи на те, которые используются для программирования на языке C. Действительно, синтаксис языка PHP в основном заимствован из языка C, таким образом, это совпадение не случайно. Ваш опыт работы с языком C очень поможет при работе с PHP. И действительно, когда вы не уверены, как написать выражение или управляющую конструкцию в PHP, сделайте это так, как вы обычно делаете это на языке C. Скорее всего это будет правильно.

Но несмотря на то что PHP имеет "свои корни" главным образом в C, имеют место также элементы, специфические только для языка Perl. Это хорошо видно в синтаксисе написания комментариев:

```
# Комментарий в стиле Perl от символа '#' до конца строки
// Комментарий в стиле C++ или Java от символа '//' до конца строки
/* Комментарий в стиле C от пары символов косая черта звездочка до пары символов звездочка-косая черта */
```

Другие аналогии с языком Perl включают оператор конкатенации строк "." (включая символы ".=" или аддитивную конкатенацию) и способ интерпретации ссылок на переменные и управляющие последовательности в двойных, а не в одинарных кавычках.

Работа с функциями и включаемыми файлами

Сценарии PHP отличаются от сценариев DBI прежде всего тем, что располагаются в дереве документов Web-сервера, тогда как сценарии DBI обычно размещаются в каталоге cgi-bin, который расположен вне дерева документов. Это вызывает проблемы с безопасностью: ошибка в конфигурации сервера может привести к тому, что страницы, расположенные в дереве документов, станут доступными для клиентов в виде простых текстовых файлов. Это значит, что опасность того, что имена пользователей и пароли станут легкодоступными, для сценариев PHP выше, чем для сценариев DBI.

Первоначальный сценарий страницы "Исторической Лиги" подвержен этому риску, так как он содержит литеральные значения имени пользователя СУБД MySQL и его пароль. Попробуем замаскировать эти параметры соединения, воспользовавшись двумя возможностями языка PHP: функциями и включаемыми файлами. Для подключения напишем функцию sampdb_connect() и поместим ее во *включаемый файл*, т.е. в файл, который не является частью главного сценария, но на который можно делать ссылки из главного сценария. Вот несколько очевидных преимуществ использования включаемых файлов.

- **Упрощается программирование подключения к базе данных.** Не нужно прописывать все параметры. Таким образом, процедура sampdb_connect() производит выборку из базы данных после подключения к ней. Вот так одна функ-

ция выполняет работу сразу двух функций PHP. Это позволяет сделать сценарии более понятными и сконцентрировать все внимание на алгоритме самого сценария, а не на программировании подключения.

- **Включаемый файл можно использовать сразу в нескольких сценариях.** Это способствует возможности неоднократного использования таких файлов. Это также позволяет легко вносить глобальные изменения во все сценарии, которые пользуются этим включаемым файлом. Например, при переносе базы данных `sampdb` из узла `cobra` на узел `boa` нет необходимости перебирать отдельные сценарии, достаточно поменять имя узла в вызове `mysql_pconnect()`, который содержит функция `sampdb_connect()`.
- **Включаемый файл становится доступным из сценария и выносится за пределы дерева документов сервера Apache.** Это делает недоступным его содержание для клиентов, даже в случае нарушения конфигурации Web-сервера. Использование включаемых файлов является хорошим способом скрыть важную информацию, которую вы не хотите выпускать за пределы своего Web-сервера.

  Это не дает 100%-ной гарантии, но если не предпринять каких-то предварительных мер, посетители узла Web-сервера смогут прочесть включаемые файлы. Прочитайте раздел "Подключение к серверу MySQL Server из Web-сценария" главы 7, "Программный интерфейс Perl API", и обратите внимание на предостережения относительно конфигурационных файлов интерфейса DBI. Аналогичное предостережение справедливо и для включаемых файлов PHP.

Для того чтобы воспользоваться включаемым файлом, необходимо указать ссылку на него, чтобы интерпретатор PHP знал, что к нему нужно обратиться. Если система имеет определенное место, где хранятся включаемые файлы, им можно воспользоваться, если нет — выполните следующие действия, чтобы создать место для файлов включения.

1. Создайте каталог, в котором интерпретатор PHP будет хранить включаемые файлы. Этот каталог не должен находиться внутри дерева документов Web-сервера! Автор для этой цели обычно использует каталог включаемых файлов PHP `/usr/local/apache/php`, который имеет тот же уровень, что и дерево документов (`/usr/local/apache/htdocs`).
2. Ссылайтесь на включаемые файлы по полному пути и имени файла или сообщайте интерпретатору PHP, какие каталоги следует просматривать при поиске файлов включения. Последний способ более удобен, так как для использования файла достаточно указать только имя файла[3]. Для того чтобы в таком случае интерпретатор PHP имел информацию, где производить поиск, достаточно изменить значение переменной `include_path` в файле инициализации PHP `php.ini` (`/usr/local/lib/`). Если она такого значения не имеет, установите полный путь к новому каталогу файлов включения:

   ```
   include_path = "/usr/local/apache/lib/php"
   ```

[3] Включаемые файлы PHP являются неким аналогом файлов заголовков языка C, даже в том смысле, что интерпретатор PHP ищет такой файл в нескольких каталогах, аналогично тому, как препроцессор C ищет файлы заголовков в каталогах C, предназначенных для хранения файлов заголовков.

Если переменная include_path уже имеет какое-то значение, просто добавьте новый путь:

```
include_path = "текущее_значение:/usr/local/apache/lib/php"
```

В ОС UNIX каталоги, перечисленные в include_path, разделяются двоеточиями, в ОС Windows для этих целей используется точка с запятой.

3. Создайте включаемый файл и поместите его в каталог для включаемых файлов. Сам включаемый файл должен иметь характерное название, например название sampdb.php вполне подойдет. Его содержимое показано в следующем листинге. Во всех примерах для подключения будем использовать базу данных sampdb, но функция соединения sampdb_connect() имеет возможность подключать любую базу данных. Постепенно этот файл будет наполняться другими функциями, но для начала он будет содержать только функцию sampdb_connect():

```
<?php
# sampdb.php - sampdb общие функции для баз данных
# Подключение к серверу MySQL с помощью секретного имени
# и пароля
function sampdb_connect ()
{
   $conn_id = @mysql_pconnect ("cobra.snake.net", "sampadm",
"secret");
   if ($conn_id && mysql_select_db ("sampdb"))
       return ($conn_id);
   return (FALSE);
}
>?
```

Функция sampdb_connect() в случае успешного соединения возвращает идентификатор соединения, при ошибке она возвращает значение "FALSE". При ошибке она не выдает диагностического сообщения или печатает предупреждение.

Содержимое файла sampdb.php ограничено тэгами "<?php" и "?>". Это сделано потому, что интерпретатор PHP начинает считывать (анализировать) включаемые файлы в режиме HTML. Без этих тэгов интерпретатор PHP разошлет файл как простой текст, не интерпретируя его как код PHP. В эти файлы можно включать литеральный HTML-код. Но если файл все же содержит PHP-код, его необходимо заключать в тэги.

4. Обратиться к файлу из сценария можно с помощью строки

```
include ("sampdb.php");
```

Когда интерпретатор PHP встречает такую строку, он ищет названный файл и считывает его содержимое. Все содержимое файла становится доступным следующим частям сценария.

Вариант sampdb.php имеется в каталоге phpapi дистрибутива sampdb. Скопируйте его в каталог include, а затем установите режим файла и его привилегии такими, чтобы его мог прочитать ваш Web-сервер. Кроме того, необходимо модифицировать параметры соединения, чтобы отобразить те, которые вы используете для подключения к СУБД MySQL.

Создав и установив наш включаемый файл sampdb.php, мы можем модифицировать страницу "Исторической Лиги", вставив включаемый файл, и подключаться к серверу MySQL с помощью вызова функции sampdb_connect():

```
<html>
<head>
<title>"Историческая Лига США"</title>
</head>
<body bgcolor="white">
<p>Добро пожаловать на Web-узел "Исторической Лиги США".</p>
<?php
# Стартовая страница "Исторической Лиги США" - версия 2
include "sampdb.php";
sampdb_connect ()
  or exit ();
$result_id = mysql_query ("SELECT COUNT(*) FROM member")
  or exit ();
if ($row = mysql_fetch_row ($result_id))
  print ("<p> В настоящее время "Историческая Лига" насчитывает " .
$row[0] . " членов.</p>");
mysql_free_result ($result_id);
?>
</body>
</html>
```

Этот сценарий можно найти под именем index2.php в каталоге phpapi/ush1 дистрибутива sampdb. Скопируйте его в каталог ush1 дерева документов вашего Web-сервера, присвойте ему имя index.php для того, чтобы заменить им файл, находящийся там сейчас. При этом менее защищенная версия файла заменяется более защищенной, так как новая версия не содержит литерального представления имени или пароля.

Сравнение операторов include и require

Язык PHP имеет в своем арсенале оператор require, который аналогичен оператору include. Различие заключается в том, что при выполнении оператора include файл считывается и интерпретируется при каждом вызове оператора include. При выполнении оператора require файл обрабатывается только один раз (в действительности происходит замещение оператора require содержимым файла). Это значит, что применение оператора require будет более эффективным, если в сценарии эта операция выполняется несколько раз. Однако если каждый раз будет считываться другой файл или есть цикл, в котором считывается набор файлов, лучше использовать оператор include, так как это позволит присвоить имя файла какой-то переменной и потом ею пользоваться.

В PHP 4 добавлены два оператора — include_once и require_once. Они аналогичны операторам include и require за исключением того, что если указанный файл уже был однажды прочитан, он повторно читаться не будет. Это свойство может пригодиться в ситуации, когда включаемые файлы содержат другие файлы, предотвращая возможность включения файла несколько раз, что может стать причиной возникновения ошибок перераспределения.

На первый взгляд может показаться, что во включаемый файл можно поместить и побольше информации. Наберитесь терпения. Файл sampdb.php может нам пригодиться также и для выполнения других функций, и мы воспользуемся им для хранения другой информации. Действительно, в этот файл даже сейчас уже можно поместить другую информацию. В каждом сценарии есть достаточно много повторяющихся частей программного кода, таких как HTML-тэги начала и конца страницы. Чтобы не писать их в каждом сценарии, напишем функции html_begin() и html_end(). Функ-

ция html_begin() может принять несколько аргументов, определяющих титул страницы и заголовок. Вот программный код этих функций:

```
function html_begin ($title, $header)
{
  print ("<html>\n");
  print ("<head>\n");
  if ($title != "")
    print ("<title>$title</title>\n");
  print ("</head>\n");
  print ("<body bgcolor=\"white\">\n");
  if ($header != "")
    print ("<h2>$header</h2>\n");
}

function html_end ()
{
  print ("</body></html>\n");
}
```

Теперь можно изменить стартовую страницу следующим образом:

```
<?php
# Стартовая страница проекта "Историческая Лига США" - версия 3
include "sampdb.php";

$title = "Историческая Лига США";
html_begin ($title, $title);
?>
<p>Добро пожаловать на Web-узел "Исторической Лиги США".</p>

<?php
sampdb_connect ()
  or exit ();
$result_id = mysql_query ("SELECT COUNT(*) FROM member")
  or exit ();
if ($row = mysql_fetch_row ($result_id))
  print ("<p>В настоящее время "Историческая Лига" насчитывает " .
$row[0] . " членов.</p>");
mysql_free_result ($result_id);

html_end ();
?>
```

Обратите внимание, что литеральный HTML-код разбит на две части. Приветственное сообщение располагается между этими частями.

Использование части, генерирующей начальную и финальную части страницы, предоставляет нам одну очень удобную возможность. Изменение кода функции позволит изменить вид заголовков и окончаний всех страниц, использующих эту функцию. Любой сценарий, который использует эту функцию, будет задействован автоматически. Например, таким образом можно поместить вверху каждой Web-страницы "Исторической Лиги" сообщение "Copyright USHL". И в этом легко может помочь такая функция, как html_end().

Web-страница с простейшим запросом

Сценарий, помещенный на Web-странице "Исторической Лиги", запускает запрос, возвращающий всего одну строку. Следующий сценарий демонстрирует обработку

многострочного результирующего набора. Он делает выборку и отображение содержимого таблицы member. Это аналог DBI-сценария dump_members.pl, разработанного нами в главе 7, "Программный интерфейс Perl API". Поэтому назовем его dump_members.php. Он отличается от версии DBI тем, что предназначен для работы в Web-окружении, а не из командной строки. Поэтому он должен генерировать вывод в формате HTML, а не простую текстовую таблицу, столбцы которой разделены табуляцией. Этот сценарий имеет следующий вид:

```
# dump_members.php - распечатка списка членов "Исторической Лиги"
include "sampdb.php";
$title = "Список членов "Исторической Лиги США";
html_begin ($title, $title);
sampdb_connect ()
   or die ("Нельзя подключиться к серверу");
# issue query
$query = "SELECT last_name, first_name, suffix, email, "
   . "street, city, state, zip, phone FROM member ORDER BY last_name";
$result_id = mysql_query ($query)
   or die ("Нельзя выполнить запрос");
print ("<table>\n"); # начало таблицы
# прочесть результаты запроса, потом освободить ресурсы
while ($row = mysql_fetch_row ($result_id))
{
   print ("<tr>\n"); # начало строки таблицы
   for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
   {
     # убрать все специальные символы и напечатать ячейку таблицы
     printf ("<td>%s</td>\n", htmlspecialchars ($row[$i]));
   }
   print ("</tr>\n"); # конец строки таблицы
}
mysql_free_result ($result_id);
print ("</table>\n"); # конец таблицы
html_end ();
?>
```

Для печати сообщения о выходе в случае возникновения ошибки вызывается функция die()[4]. Такой подход к процедуре обработки ошибочных ситуаций отличается от того, когда в случае выхода по ошибке это делалось без каких-либо диагностических сообщений. От сценария dump_members.php уже ожидается конкретный результат; таким образом, о возникновении сбоя свидетельствует диагностическое сообщение.

Сценарий устанавливается в каталоге ushl, и доступ к нему осуществляется как http://www.snake.net/ushl/dump_members.php. Таким образом, добавим ссылку из Web-страницы "Исторической Лиги":

```
<?php
# Стартовая страница проекта "Историческая Лига США" - версия 4
include "sampdb.php";
$title = "Историческая Лига США";
html_begin ($title, $title);
?>
```

[4] Функция die() аналогична функции exit(), но перед выходом печатает диагностическое сообщение.

```
<p> Добро пожаловать на Web-узел "Исторической Лиги США".</p>
<?php
sampdb_connect ()
  or exit ();
$result_id = mysql_query ("SELECT COUNT(*) FROM member")
  or exit ();
if ($row = mysql_fetch_row ($result_id))
  print ("<p> В настоящее время Историческая Лига" насчитывает " .
$row[0] . " членов. </p>");
mysql_free_result ($result_id);
?>
<p>
Можно просмотреть список членов <a href="dump_members.php">here</a>.
</p>
<?php
html_end ();
?>
```

Сценарий dump_members.php служит для демонстрации возможностей получения PHP-сценарием информации из СУБД MySQL и преобразования ее в содержимое Web-страницы. По необходимости этот сценарий можно видоизменить и получить с его помощью более сложные результаты. Одно из таких изменений предназначено для того, чтобы значения столбца email были гиперссылками, а не статическим текстом, что существенно облегчит посетителям узла рассылку почты членам лиги. Дистрибутив sampdb содержит сценарий dump_members2.php, на который возложена эта задача. Он отличается от сценария dump_members.php самым малым: циклом выборки и отображения записей о членах. Первоначальный цикл выглядит следующим образом:

```
while ($row = mysql_fetch_row ($result_id))
{
  print ("<tr>\n"); # начало строки таблицы
  for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
  {
    # убрать все специальные символы и напечатать ячейку таблицы
    printf ("<td>%s</td>\n", htmlspecialchars ($row[$i]));
  }
  print ("</tr>\n"); # конец строки таблицы
}
```

Адреса электронной почты размещаются в четвертом столбце результата запроса. Таким образом, dump_members2.php обрабатывает этот столбец несколько иным способом, чем все остальные, печатая гиперссылку, если строка не пустая:

```
while ($row = mysql_fetch_row ($result_id))
{
  print ("<tr>\n"); # начало строки таблицы
  for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
  {
    print ("<td>");
    if ($i == 3 && $row[$i] != "") # электронная почта находится в 4-м
столбце результата
    {
        printf ("<a href=\"mailto:%s\">%s</a>",
                $row[$i],
                htmlspecialchars ($row[$i]));
    }
```

```
    else
    {
        # убрать все специальные символы и напечатать ячейку таблицы
        print (htmlspecialchars ($row[$i]));
    }
    print ("</td>\n");
  }
  print ("</tr>\n"); # конец строки таблицы
}
```

Обработка результатов запроса

Рассмотрим подробнее процесс выполнения запросов СУБД MySQL и обработки результирующих наборов. В языке PHP все запросы можно выполнить, вызвав функцию `mysql_query()`, которая принимает в качестве аргументов строку запроса и идентификатор соединения, причем идентификатор задавать совсем не обязательно. Функцию `mysql_query()` можно вызвать следующим образом:

```
$result_id = mysql_query ($query, $conn_id); # задать соединение явным способом
$result_id = mysql_query ($query); # задать соединение по умолчанию
```

Функция `mysql_query()` возвращает идентификатор результата, который должен интерпретироваться в соответствии с типом оператора. В случае запроса, не возвращающего результирующего набора (это такие запросы, как `DELETE`, `INSERT`, `REPLACE` и `UPDATE`), функция `mysql_query()` возвращает значение "`TRUE`" или "`FALSE`", показывающее, как завершилась обработка запроса. В случае успешного завершения можно вызвать функцию `mysql_affected_rows()`. Она покажет, сколько строк было обработано (удалено, добавлено, замещено или модифицировано).

Для операторов типа `SELECT` функция `mysql_query()` возвращает идентификатор набора или значение "`FALSE`", соответственно отражающие успех или сбой в работе запроса. Идентификатор результирующего набора используется для получения информации о результирующем наборе. Например, можно получить информацию о том, сколько строк содержит результирующий набор, или сделать выборку строк набора.

Возвращение функцией `mysql_query()` значения "`FALSE`" (т.е. нуля) означает, что запрос не сработал — другими словами, возникла какая-то ошибочная ситуация и запрос даже не выполнялся. Запрос может не отработать по различным причинам.

- Запрос может содержать синтаксическую ошибку.
- Запрос может не иметь синтаксической ошибки, но семантически быть бессмысленным. Такая ситуация возникает, когда, к примеру, делается попытка выбрать несуществующий столбец.
- У вас нет достаточных привилегий на выполнение запроса.
- Из-за сетевых проблем мог стать недоступным узел сервера MySQL.

В любом из этих случаев (могут иметь место и другие) функция `mysql_query()` возвращает значение "`FALSE`". Можно узнать и конкретную причину, вызвавшую ошибку. Вызовите для этого функцию `mysql_error()` или `mysql_errno()`. С их

помощью можно получить строку сообщения или числовой код ошибки. (Подробно об этом рассказывается ниже, в разделе "Обработка ошибок".)

Не рассчитывайте априори, что `mysql_query()` отрабатывает успешно

Очень часто начинающие пользователи PHP задают вопрос о происхождении такого сообщения в момент выполнения сценария:

```
Warning: 0 is not a MySQL result index in file on line n
Предупреждение: 0 не является результирующим индексом в строке n файла
```

Это сообщение свидетельствует о том, что какой-то функции было передано значение идентификатора результирующего набора, равное 0 (такой, например, как функции выборки строк). И эта функция ожидает получения правильного результирующего набора. Это означает, что предыдущий вызов функции `mysql_query()` вернул 0, т.е. значение "FALSE". Другими словами, функция `mysql_query()` завершилась с какой-то ошибкой, и автор сценария не позаботился о том, чтобы проверить полученный от нее результат перед вызовом другой функции. При работе с функцией `mysql_query()` всегда проверяйте возвращаемое значение.

При работе с функцией `mysql_query()` можно наблюдать две самые распространенные ошибки: считают, что возвращаемое значение является счетчиком строк и что это данные, которые возвращает запрос. Ни одно из этих мнений неверно.

Обработка запросов, не возвращающих результирующего набора

Для операторов, модифицирующих строки, возвращаемое функцией `mysql_query()` значение не является счетчиком строк. Оно является лишь индикатором успешного или неуспешного завершения его работы.

А для того чтобы получить счетчик строк, необходимо вызвать `mysql_affected_rows()`. Предположим, что вы хотите удалить запись о члене 149 в таблице `member` и вывести полученный результат.

Вот как это не надо делать:

```
$result_id = mysql_query ("DELETE FROM member WHERE member_id = 149");
if (!$result_id)
  print ("член 149 не удален \n");
else
  printf ("член 149 удален \n");
```

Этот код содержит грубую ошибку, так как в нем `$result_id` рассматривается как счетчик возвращаемых строк. При этом считается, которое значение 0 ("FALSE") означает, что запрос выполнился успешно, но не удалил ни одной строки. А в действительности это значит, что запрос не был выполнен вообще. Вот другой пример ошибочной интерпретации результата работы функции `mysql_query()`:

```
$result_id = mysql_query ("DELETE FROM member WHERE member_id = 149");
if (!$result_id)
  print ("Ошибка при выполнении запроса \n");
else
  print ("член 149 удален \n");
```

Этот код правильно отличает сбой запроса от успешного завершения, но по-прежнему содержит очевидные ошибки, так как он предполагает, что если запрос успешно отработал, он действительно удалил запись. Что же не так? Оператору

DELETE совсем не надо что-либо удалять, чтобы успешно выполниться. Если член с идентификатором, равным 149, существует, MySQL удаляет запись и функция mysql_query() возвращает значение "TRUE". Но если такого члена нет, функция mysql_query() ничего не удаляет и также возвращает значение "TRUE". Для того чтобы определить, был ли в действительности успешным запрос и действительно ли он удалил что-нибудь, вызовите функцию mysql_affected_rows(). Вот пример, в котором изложена правильная интерпретация результата:

```
$result_id = mysql_query ("DELETE FROM member WHERE member_id = 149");
if (!$result_id)
   print ("Ошибка при выполнении запроса \n");
else if (mysql_affected_rows () < 1)
   print ("запись с номером 149 не найдена \n");
else
   print ("член 149 удален \n");
```

Обработка запросов, возвращающих результирующий набор

Для обработки запроса, возвращающего строки, необходимо сначала выполнить запрос, а затем, если он отрабатывает успешно, выбрать содержимое результирующего набора. Несложно забыть, что этот процесс происходит в два этапа, особенно если он возвращает единственное значение.

Рассмотрим следующую попытку определить, сколько записей считывает таблица member:

```
$result_id = mysql_query ("SELECT COUNT(*) FROM member");
print ("В таблице member имеется $result_id записей \n");
```

Этот код содержит ошибку. Предполагается, что потому, что результирующий набор содержит только одно значение, оно и должно быть тем, что возвращает функция mysql_query(). Но это не так. Значение $result_id позволяет получить доступ к результирующему запросу, но никогда само по себе не является результатом. Даже если результат состоит из одного значения, его необходимо выбирать после выполнения запроса. Следующий код иллюстрирует один из приемов, как это делается. Он проверяет правильность выполнения функции mysql_query(), а потом производит выборку записи в $row с помощью функции mysql_fetch_row(). Только в случае успешного завершения обеих функций этот код распечатывает значение, полученное операцией COUNT(*):

```
$result_id = mysql_query ("SELECT COUNT(*) FROM member");
if (!$result_id || !($row = mysql_fetch_row ($result_id)))
   print ("Ошибка при выполнении запроса \n");
else
   print ("В таблице member имеется $row[0] записей \n");
```

Аналогичный подход можно использовать, если ожидается получить несколько записей, хотя в этом случае для выборки строк обычно используется цикл. В следующем примере показан один из способов, как это можно сделать:

```
$result_id = mysql_query ("SELECT * FROM member");
if (!$result_id)
   print ("Ошибка при выполнении запроса \n");
else
{
   printf ("количество возвращенных строк: %d\n", mysql_num_rows
```

```
($result_id));
  # выбрать каждую строку в результирующий набор
  while ($row = mysql_fetch_row ($result_id))
  {
    # напечатать значения в строчку, разделив их запятыми
    for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
    {
        if ($i > 0)
            print (",");
        print ($row[$i]);
    }
    print ("\n");
  }
  mysql_free_result ($result_id);
}
```

Если этот запрос завершается неуспешно, будет получено значение "FALSE", о чем будет выдано диагностическое сообщение. (В зависимости от обстоятельств возможна и другая реакция на ошибку.) Если запрос отрабатывает успешно, функция `mysql_query()` возвращает идентификатор результирующего набора. Он нам может очень пригодиться в различных аспектах (но только не как счетчик!). Идентификатор результирующего набора можно использовать следующим образом.

- Передать функции `mysql_num_rows()` для определения количества строк в результирующем наборе.
- Передать функции `mysql_num_fields()` для определения количества столбцов в результирующем наборе.
- Передать процедуре выборки строк результирующего набора. В примере для этой цели используется функция `mysql_fetch_row()`, но существуют и другие варианты. Они будут показаны ниже.
- Передать функции `mysql_free_result()` для того, чтобы очистить результирующий набор и все ресурсы, связанные с ним.

Язык PHP в своем арсенале имеет несколько функций выборки данных из результирующего набора после успешного завершения функцией `mysql_query()` запроса `SELECT` (табл. 8.1). Любая из этих функций принимает в качестве аргумента идентификатор результирующего набора и возвращает значение "FALSE", если больше строк нет.

Таблица 8.1. Функции выборки строк языка PHP

Имя функции	Возвращаемое значение
`mysql_fetch_row()`	Массив, доступ к элементам которого осуществляется с помощью числового индекса
`mysql_fetch_assoc()`	Массив, доступ к элементам которого осуществляется с помощью ассоциативного индекса
`mysql_fetch_array()`	Массив, доступ к элементам которого осуществляется с помощью числового или ассоциативного индекса
`mysql_fetch_object()`	Объект, доступ к элементам которого осуществляется как к свойствам

Основной функцией является функция `mysql_fetch_row()`, возвращающая как массив следующую строку результирующего набора. Доступ к элементам мас-

сива осуществляется с помощью числового индекса в диапазоне чисел от 0 до mysql_num_fields()-1. Покажем, как функция mysql_fetch_row() используется в простейшем цикле построчной выборки и печати:

```
$query = "SELECT * FROM president";
$result_id = mysql_query ($query)
  or die ("Ошибка запроса");
while ($row = mysql_fetch_row ($result_id))
{
  for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
  {
    if ($i > 0)
        print ("\t");
    print ($row[$i]);
  }
  print ("\n");
}
mysql_free_result ($result_id);
```

Переменная $row является массивом. Доступ к его элементам производится как $row[$i], где $i — числовой индекс столбца. Если вам известна функция count(), вы можете захотеть воспользоваться ею для определения количества элементов в каждой строке вместо функции mysql_num_fields(). Но ни в коем случае не делайте этого. Функция count() считает количество установленных элементов в массив, но PHP не устанавливает элементы, которые соответствуют столбцам, имеющим значения "NULL". Поэтому функция count() будет не самым лучшим решением для возвращенных столбцов, так как она и не предназначалась для выполнения этой задачи. Это справедливо и для двух других функций выборки.

Другим способом выборки массива является присвоение результата списку переменных. Например, для выборки столбцов last_name и first_name непосредственно в переменные $ln и $fn, чтобы напечатать имена в последовательности имя (first_name) и фамилия (last_name), необходимо сделать следующее:

```
$query = "SELECT last_name, first_name FROM president";
$result_id = mysql_query ($query)
  or die ("Ошибка запроса");
while (list ($ln, $fn) = mysql_fetch_row ($result_id))
  printf ("%s %s\n", $fn, $ln);
mysql_free_result ($result_id);
```

Переменным могут присваиваться любые допустимые значения, но их порядок в list() должен соответствовать порядку столбцов, выбранных по запросу.

Вторая функция выборки, приведенная в табл. 8.1, mysql_fetch_assoc(), аналогична функции mysql_fetch_row(), но доступ к ее элементам можно осуществить с помощью ассоциативного индекса. Другими словами, доступ к элементам массива можно производить по имени выбранного столбца:

```
$query = "SELECT last_name, first_name FROM president";
$result_id = mysql_query ($query)
  or die ("Ошибка запроса");
while ($row = mysql_fetch_assoc ($result_id))
  printf ("%s %s\n", $row["first_name"], $row["last_name"]);
mysql_free_result ($result_id);
```

Функция mysql_fetch_assoc() — новая функция выборки строк; она появилась только в PHP 4.0.3.

Третья функция выборки, приведенная в табл. 8.1, mysql_fetch_array(), аналогична функции mysql_fetch_row(), но доступ к ее элементам можно осуществить с помощью цифрового или ассоциативного индекса. Другими словами, доступ к элементам массива можно производить по номеру или по имени:

```
$query = "SELECT last_name, first_name FROM president";
$result_id = mysql_query ($query)
  or die ("Ошибка запроса");
while ($row = mysql_fetch_array ($result_id))
{
  printf ("%s %s\n", $row[0], $row[1]);
  printf ("%s%s\n", $row["first_name"], $row["last_name"]);
}
mysql_free_result ($result_id);
```

Функция mysql_fetch_array() по сравнению с функцией mysql_fetch_row() возвращает расширенный набор информации. Несмотря на это, различие между двумя функциями в производительности мизерно. Для этих целей можно воспользоваться функцией mysql_fetch_array(). Ни к каким существенным потерям производительности это не приведет.

Последняя функция выборки — mysql_fetch_object() — возвращает следующую строку результирующего набора как объект. Это значит, что доступ к элементам может производиться в соответствии с синтаксисом $row->*col_name*. Например, для того, чтобы выбрать значения last_name и first_name из таблицы president, к ним необходимо обратиться как $row->last_name и $row->first:

```
$query = "SELECT last_name, first_name FROM president";
$result = mysql_query ($query)
  or die ("Ошибка запроса");
while ($row = mysql_fetch_object ($result))
  printf ("%s %s\n", $row->first_name, $row->last_name);
mysql_free_result ($result_id);
```

Что если запрос содержит вычисляемые выражения? Например, запрос может возвращать значения, которые вычисляются в выражениях:

```
SELECT CONCAT(first_name, ' ', last_name) FROM president
```

Такой запрос нельзя обрабатывать с помощью функции mysql_fetch_object(). Именем выбранного столбца будет имя выражения, которое не является допустимым именем свойства. Однако допустимое имя можно присвоить с помощью псевдонима. Вот запрос, который присваивает псевдоним full_name, позволяющий осуществить доступ с использованием функции $row->full_name при выборке результатов с помощью функции mysql_fetch_object():

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name FROM president
```

В случае выборки результатов этого запроса с помощью функции mysql_fetch_object() псевдоним позволяет обращаться к такому столбцу как к $row->full_name.

Проверка результатов запроса на значения "NULL"

Язык PHP интерпретирует значения "NULL" как неустановленные значения. Проверка на значение "NULL" значений, возвращаемых запросом SELECT, произ-

водится с помощью функции isset(). Ниже приведен пример, в котором производится выборка и печать имен и электронных адресов из таблицы member, печатая сообщение "Электронный адрес отсутствует", если столбец имеет значение "NULL".

```
$query = "SELECT last_name, first_name, email FROM member";
$result_id = mysql_query ($query)
  or die ("Ошибка запроса");
while (list ($last_name, $first_name, $email) = mysql_fetch_row
($result_id))
{
  printf ("Имя: %s %s, Электронная почта: ", $first_name, $last_name);
  if (!isset ($email))
    print ("Электронный адрес отсутствует");
  else
    print ($email);
  print ("\n");
}
mysql_free_result ($result_id);
```

Есть еще одна подобная функция empty(), но она возвращает один и тот же результат как для значений "NULL", так и для пустых строк. Поэтому при тестировании строк на пустое значение эта функция мало в чем может помочь.

В PHP 4 можно проверять величины на пустые значения с помощью константы "NULL" и оператора "===" (идентично равный):

```
$query = "SELECT last_name, first_name, email FROM member";
$result_id = mysql_query ($query)
  or die ("Ошибка запроса");
while (list ($last_name, $first_name, $email) = mysql_fetch_row
($result_id))
{
  printf ("Имя: %s %s, Электронная почта: ", $first_name, $last_name);
  if ($email === NULL)
    print ("адрес отсутствует");
  else
    print ($email);
  print ("\n");
}
mysql_free_result ($result_id);
```

Обработка ошибок

Язык PHP предоставляет в распоряжение разработчика три средства обработки ошибочных ситуаций. Два из них применимы ко всем типам ошибок, а одна имеет только отношение к операциям с СУБД MySQL. Во-первых, для подавления сообщений об ошибках можно использовать символ отмены вывода — @. Этот символ можно использовать с любой функцией, которая выводит сообщение. Это уже было нами проделано при вызове функции mysql_connect() для предотвращения рассылки сообщений от этой функции клиенту.

```
$conn_id = @mysql_connect ("cobra.snake.net", "sampadm", "secret");
```

Во-вторых, для отключения или включения диагностирования ошибок на любом из описанных в табл. 8.2 уровней можно воспользоваться функцией error_reporting().

Таблица 8.2. Уровни обработки ошибок языка PHP

Уровень ошибки	Тип диагностируемой ошибки
E_ERROR	Ошибки обычных функций
E_WARNING	Обычные предупреждения
E_PARSE	Ошибки синтаксического анализатора
E_NOTICE	Замечания
E_CORE_ERROR	Ошибки обработчика
E_CORE_WARNING	Предупреждения обработчика
E_COMPILE_ERROR	Ошибки компилятора
E_COMPILE_WARNING	Предупреждения компилятора
E_USER_ERROR	Ошибки пользователей
E_USER_WARNING	Предупреждение пользователей
E_USER_NOTICE	Уведомление пользователей
E_ALL	Все ошибки

Для того чтобы управлять диагностированием ошибок с помощью функции `error_reporting()`, вызовите ее с аргументом, равным побитовому "ИЛИ" всех уровней, которые вы хотите установить. Для подавления вывода ошибок функций СУБД MySQL достаточно отключить уровни `E_ERROR` и `E_WARNING`:

```
error_reporting (E_PARSE | E_NOTICE);
```

Отключать диагностирование уровня `E_PARSE` (ошибки синтаксического анализатора) не рекомендуется, так как это может затруднить отладку любых изменений, вносимых в SQL-запросы. Предупреждения уровня `E_NOTICE` зачастую можно игнорировать, но иногда они могут помочь определить ошибки в сценарии, на которые следует обратить внимание. Таким образом, диагностирование ошибок этого уровня также можно оставить включенным. Ошибки других уровней, за исключением, пожалуй, уровня `E_ALL`, можно не брать в расчет.

В PHP 3 имеются только четыре уровня диагностирования ошибок, и на них можно ссылаться с помощью значений, а не символических констант (табл. 8.3).

Таблица 8.3. Уровни обработки ошибок языка PHP 3

Уровень ошибки	Тип диагностируемой ошибки
1	Ошибки обычных функций
2	Обычные предупреждения
4	Ошибки синтаксического анализатора
8	Замечания

Третий метод обработки ошибок заключается в использовании функций `mysql_error()` и `mysql_errno()`. Эти функции диагностируют ошибки, которые возвращаются сервером MySQL. Они аналогичны соответствующим вызовам C API. Функция `mysql_error()` возвращает сообщение об ошибке в виде строки (пустая строка свидетельствует об отсутствии ошибки). Функция `mysql_errno()` возвращает номер ошибки (0 – если ошибки нет). Обе функции принимают идентификатор соединения в качестве аргумента, определяющего соединение с сервером MySQL, и обе воз-

вращают информацию об ошибке самой последней вызванной функции MySQL. Идентификатор соединения можно не указывать; если он отсутствует, диагностируется последнее соединение. Например, диагностировать ошибку, произошедшую при работе `mysql_query()`, можно следующим образом:

```
$result_id = mysql_query ($query);
if (!$result_id)
{
   print ("errno: " . mysql_errno() . "\n");
   print ("error: " . mysql_error() . "\n");
}
```

Интерфейс C API позволяет получить информацию об ошибке от функций `mysql_error()` и `mysql_errno()`, даже если попытка подключиться к серверу будет неудачной. Это справедливо и для языка PHP, но только начиная с версии PHP 4.0.6. До этой версии вызов этих функций в PHP не возвращал никакой полезной информации до тех пор, пока соединение не было удачным. Другими словами, если соединение не удалось, использовать функцию `mysql_error()` или `mysql_errno()` для получения ответа на вопрос "Почему?" бесполезно. Для того чтобы определить причину неудачного соединения, необходимо предпринять какие-то специальные действия. Как это сделать, вы узнаете в приложении 3, "Программный интерфейс языка PHP".

Сценарии, представленные в этой главе, при возникновении ошибочной ситуации выдают достаточно общие сведения об ошибках наподобие "Ошибка запроса". Однако по мере разработки сценария вы поймете, что лучше все же вызвать функцию `mysql_error()`, чтобы определить конкретную причину возникновения ошибки.

Проблема кавычек

Как в C и Perl, при конструировании строк SQL-запроса на языке PHP необходимо быть очень осторожным с кавычками. Методы работы с кавычками тоже аналогичны, хотя имена функций в этих языках несколько отличаются. Предположим, что необходимо создать запрос, добавляющий запись в таблицу. Значение, которое будет добавлено в таблицу, можно заключить в кавычки:

```
$last = "O'Malley";
$first = "Brian";
$expiration = "2002-9-1";
$query = "INSERT member (last_name, first_name, expiration)"
    .  " VALUES('$last','$first','$expiration')"
```

Проблема здесь заключается в том, что одно из значений, заключенных в кавычки, содержит одинарную кавычку (`O'Malley`). Это является причиной синтаксической ошибкой при посылке такого запроса на сервер MySQL. Для того чтобы решить эту проблему в языке C, нужно прибегнуть к помощи функции `mysql_escape_string()`. В сценарии Perl DBI можно воспользоваться методом `quote()`. Для решения аналогичной проблемы в языке PHP используется функция `addslashes()`, выполняющая аналогичную функцию. Например, вызов `addslashes("O'Malley")` возвращает значение `"O\'Malley"`. Предыдущий пример можно переписать следующим образом:

```
$last = addslashes ("O'Malley");
$first = addslashes ("Brian");
$expiration = addslashes ("2002-9-1");
$query = "INSERT member (last_name, first_name, expiration)"
    .  " VALUES('$last','$first','$expiration')"
```

Если метод quote() интерфейса DBI добавляет к строке ограничивающие кавычки, то функция addslashes() этого не делает. Таким образом, их необходимо добавить явным образом, ограничивая добавляемые ими значения. Этот метод хорошо работает со всеми значениями, за исключением неустановленных, для которых в строку запроса необходимо вставить слово "NULL" без кавычек. Эта проблема имеет решение, о котором вы скоро узнаете.

Альтернативой функции addslashes() для 4.0.3 являются функции mysql_escape_string() и mysql_real_escape_string() — для PHP 4.3.0:

```
$escaped_str = mysql_escape_string ($str);
$escaped_str = mysql_real_escape_string ($str, $conn_id);
```

Обе эти функции основываются на функции с таким же именем из MySQL C API. При выполнении кодирования функция mysql_real_escape_string() принимает во внимание кодировку соединения и поэтому принимает в качестве аргумента идентификатор соединения. При отсутствии аргумента функция использует кодировку текущего соединения.

Принимая во внимание трудность выбора сразу из трех функций по обработке управляющих символов в строках, будет очень затруднительно выбирать каждый раз функцию, кодирующую строку запроса. Поэтому напишем отдельную функцию quote_value() и поместим ее в библиотечный файл sampdb.php. Функция quote_value() будет обрабатывать управляющие символы в строках с помощью новейшей из существующих функций кодировки, проверяя существование этой функции с помощью функции function_exists():

```
function quote_value ($str)
{
  if (!isset ($str))
    return ("NULL");
  if (function_exists ("mysql_real_escape_string"))
    return ("'" . mysql_real_escape_string ($str) . "'");
  if (function_exists ("mysql_escape_string"))
    return ("'" . mysql_escape_string ($str) . "'");
  return ("'" . addslashes ($str) . "'");
}
```

В дополнение к выбору из имеющихся функций кодировки функция quote_value() также решает проблему обработки неустановленных значений (NULL), которая только что упоминалось. Если аргумент представлен неустановленным значением, quote_value() возвращает слово NULL без кавычек, иначе он обрабатывает аргумент и возвращает результат, заключенный в одинарные кавычки. Таким образом, использовав quote_value() для создания запроса, вы вставляете значение, которое он возвращает непосредственно в строку запроса, не добавляя при этом самостоятельно никаких кавычек:

```
$last = quote_value ("O'Malley");
$first = quote_value ("Brian");
$expiration = quote_value ("2002-9-1");
```

```
$query = "INSERT INTO member (last_name,first_name,expiration)"
  . " VALUES($last,$first,$expiration)";
```

Предшествующее обсуждение тоже имеет отношение к проблемам заключения в кавычки SQL-запросов, которые готовятся для посылки на сервер MySQL. В процессе написания Web-страниц всегда возникают аналогичные проблемы. Строки, которые записываются в HTML или как часть URL, лучше всего кодировать, если строка может содержать специальные символы в HTML-коде или URL. Для этого можно использовать функции PHP `htmlspecialchars()` и `urlencode()`. Они подобны методам модуля `CGI.pm` `escapeHTML()` и `escape()`.

Работа с PHP

В конце этой главы мы попытаемся достичь целей, поставленных в главе 1, "Знакомство с СУБД MySQL и SQL", которые нами все еще не достигнуты.

- Написать сценарий, который позволит вводить и редактировать результаты экзаменов и викторин в проекте учета успеваемости учащихся.
- Разработать для проекта "Историческая Лига" викторину, посвященную президентам США. При этом ее необходимо сделать интерактивной, чтобы вопросы к посетителям Web-узла генерировались на ходу.
- Кроме того, необходимо предложить членам "Исторической Лиги" возможность редактировать информацию о себе, чтобы она всегда оставалась актуальной, и тем самым, уменьшить работу по редактированию записей.

Каждый из этих сценариев генерирует несколько взаимосвязанных Web-страниц, которые переключаются на основании информации, содержащейся на создаваемых страницах. Если концепция межстраничного взаимодействия вам не знакома, обратитесь к разделу "Создание многоцелевых Web-страниц" главы 7, "Программный интерфейс Perl API".

Ввод оценок учащихся

Обратимся к проекту "Учет успеваемости" и напишем сценарий `score_entry.php`, который позволит нам ввести новый набор оценок за тесты или викторины или редактировать уже существующие оценки. Web-каталог для этого проекта имеет имя `gp` и располагается в дереве документов Web-сервера Apache, что соответствует такому URL для нашего Web-узла:

```
http://www.snake.net/gp/
```

Этот каталог в настоящее время пуст, поэтому посетители, которые задают этот URL, могут получить только сообщение об ошибке "Page not found" или страницу с пустым листингом каталога. Для решения этой проблемы мы напишем небольшую Web-страницу `index.php` и поместим ее в каталог `gp`, чтобы она стала стартовой страницей нашего узла. Следующего сценария, который содержит две ссылки, будет достаточно. Одна ссылка будет на сценарий `score_browse.pl`, который был создан нами в главе 7, "Программный интерфейс Perl API". Другая ссылка сделана на сценарий `score_entry.php`, который мы сейчас и напишем:

```
<?php
# Стартовая страница проекта "Учет успеваемости"
```

```
include "sampdb.php";
$title = "Проект "Учет успеваемости";
html_begin ($title, $title);
?>
<p>
<a href="/cgi-bin/score_browse.pl">View</a> Оценки за викторины и тес-
ты
</p>
<p>
<a href="score_entry.pl">Введите или отредактируйте</a> оценки за эк-
замены и тесты
</p>
<?php
html_end ();
?>
```

Посмотрим теперь, каким образом можно разработать и реализовать сценарий `score_entry.php`, который позволил бы нам вводить новые оценки по экзаменам или редактировать уже существующие оценки. Последняя возможность необходима для обработки результатов учащихся, которые сдали экзамен позже остальных учащихся класса по болезни или по какой-то другой причине (или, о ужас, для исправления ошибки, допущенной при первоначальном вводе оценки). Общий вид этого сценария будет таким.

- Стартовая страница предоставляет список известных зачетных экзаменов и позволяет вам произвести выбор одного из них или создать запись о новом экзамене.
- Если выбрано создание нового экзамена, сценарий перейдет на страницу, позволяющую указать дату и тип экзамена. После того как запись будет создана, сценарий повторно отобразит страницу, содержащую перечень экзаменов, содержащий уже новый экзамен.
- Выбор определенного экзамена вызовет на экран страницу ввода результатов. Информация об экзамене хранится в верхней части этой страницы (идентификатор экзамена, дата и тип экзамена). За этой информацией следует построчное отображение оценок учащихся. По одной строке на каждого. Для новых экзаменов записи будут пустыми. Для уже существующих экзаменов записи будут содержать оценку каждого учащегося. Щелкнув на кнопке Submit, введем оценки в таблицу `score`.

Этот сценарий должен быть рассчитан на выполнение ряда разнообразных действий, которые будут передавать переменную состояния между страницами, чтобы передать сценарию информацию о том, что от него требуется сделать. Это очень просто сделать в PHP, так как PHP обрабатывает переменные как параметры URL и преобразует их в переменные с именами параметров. Например, можно запрограммировать параметры, добавив их в конец URL:

```
http://www.snake.net/gp/score_entry.php?action=value
```

Значения параметров могут также задаваться пользователем в самой форме. Каждое поле в форме имеет свое имя и значение. Язык PHP позволяет передавать значения параметров через специальные массивы. Параметры, добавленные в конец URL и, соответственно, посланные через запрос GET, помещаются в гло-

бальный массив $HTTP_GET_VARS. Для параметров, полученных в запросе POST (это может быть содержимое формы, имеющее значение атрибута method, равное POST), параметры размещаются в глобальном массиве $HTTP_POST_VARS.

Это ассоциативные массивы, элементы которых связаны с именами параметров. Например, параметр action, посланный через URL, будет доступен сценарию PHP как значение $HTTP_GET_VARS["action"]. Если форма содержит поле address и форма активизируется с помощью запроса POST, значение можно найти в элементе $HTTP_POST_VARS["address"].

Значения параметров можно также получить и через поля формы. Предположим, что форма содержит поля с именами name и address. Когда клиент подтверждает форму, Web-сервер вызывает этот сценарий для обработки содержимого формы. Если форма выдается через запрос GET, сценарий может определить, какие значения были введены в форму, проверив значения переменных $HTTP_GET_VARS["name"] и $HTTP_GET_VARS["address"]. Если форма выдается через запрос POST, сценарий может определить, какие значения были введены в форму, проверив значения переменных $HTTP_POST_VARS["name"] и $HTTP_POST_VARS["address"]. Очевидно, что при наличии множества полей в форме присваивать уникальное имя каждому будет неудобно. Язык PHP позволяет передавать переменные из формы или в форму. Поля, имеющие имена x[0], x[1] и т.д., PHP будет сохранять в массиве $HTTP_GET_VARS["x"] или $HTTP_POST_VARS["x"]. Если присвоить значение массива переменной $x, доступ к элементам может осуществляться как к $x[0], $x[1] и т.д.

В большинстве случаев вы не захотите заботиться о том, каким запросом (GET или POST) был передан параметр. Поэтому достаточно написать утилиту (script_param()), которая принимает имя параметра и проверяет оба массива на значение параметра. Если параметр отсутствует, программа возвратит неустановленное значение:

```
function script_param ($name)
{
global $HTTP_GET_VARS, $HTTP_POST_VARS;
unset ($val);
   if (isset ($HTTP_GET_VARS[$name]))
     $val = $HTTP_GET_VARS[$name];
   else if (isset ($HTTP_POST_VARS[$name]))
     $val = $HTTP_POST_VARS[$name];
   # возвращает @$val, а не $val, для предотвращения сообщения
"undefined value"
   # в случае неустановленного значения $val и активизированного режима
выдачи сообщений
   return (@$val);
}
```

Обратите внимание на то, что функция script_param() объявляет явным образом массивы глобальными с помощью ключевого слова global. Глобальные переменные PHP являются доступными без ключевого слова global только в зоне действия глобальных переменных, таких, каким является главное тело сценария. В зоне действия локальных переменных, каковой является функция, ключевое слово global сообщает языку PHP, что вы хотите осуществить доступ к глобальной, а не локальной переменной, которая может иметь то же имя. Эта

функция также содержит оператор @ в операторе return() для подавления сообщений об ошибке. (Если параметр не задан, функция script_param() возвращает неустановленное значение; если сценарий будет изменен для того, чтобы включать сообщение об ошибках, возвращение неустановленного значения вызывает вывод предупреждения.)

В PHP 4.1 были добавлены два новых массива параметров: $_GET и $_PUT. Они аналогичны $HTTP_GET_VARS и $HTTP_POST_VARS, но являются "суперглобальными" массивами. Это означает, что они доступны в любой зоне и для их объявления не надо прибегать к помощи ключевого слова global. Для того чтобы модифицировать функцию script_param() для использования новых суперглобальных массивов, если они имеются в наличии, перепишите новую функцию следующим образом:

```
function script_param ($name)
{
global $HTTP_GET_VARS, $HTTP_POST_VARS;
  unset ($val);
  if (isset ($_GET[$name]))
    $val = $_GET[$name];
  else if (isset ($_POST[$name]))
    $val = $_POST[$name];
  else if (isset ($HTTP_GET_VARS[$name]))
    $val = $HTTP_GET_VARS[$name];
  else if (isset ($HTTP_POST_VARS[$name]))
    $val = $HTTP_POST_VARS[$name];
  if (isset ($val) && get_magic_quotes_gpc ())
    $val = remove_backslashes ($val);
  # возвращает @$val, а не $val, для предотвращения сообщения
"undefined value"
  # в случае неустановленного значения $val и активизированного режима
выдачи сообщений
  return (@$val);
}
```

Эту измененную версию сценария script_param() можно найти в библиотеке sampdb.php дистрибутива sampdb. Он позволяет достаточно просто осуществлять доступ к имени значения входных параметров, не заботясь при этом о том, в каком массиве они могут храниться. Кроме того, вы обнаружите, что эта пересмотренная версия также содержит другое изменение в дополнение к проверке значений массивов $_GET и $_PUT; после выборки значения параметра она передает значение функции remove_backslashes(). Ее задачей является адаптация к конфигурациям, в которых активизирована настройка magic_quotes_gpc, с помощью такой строки файла инициализации PHP:

```
magic_quotes_gpc = On;
```

Если эта настройка активизирована, PHP добавляет обратные косые черты к значению параметра для того, чтобы заключить в кавычки такие специальные символы, как кавычки или обратные косые черты. Дополнительные обратные косые черты затрудняют проверку значений параметров, таким образом, функция remove_backslashes() их выделяет. Это реализуется следующим образом.

Этот алгоритм является рекурсивным, так как в PHP 4 существует возможность создания параметров, имеющих формат вложенных массивов:

```
function remove_backslashes ($val)
{
   if (!is_array ($val))
     $val = stripslashes ($val);
   else
   {
     reset ($val);
     while (list ($k, $v) = each ($val))
         $val[$k] = remove_backslashes ($v);
   }
   return ($val);
}
```

Входные параметры Internet и настройка `register_globals`

Вероятно, вы знакомы с настройкой языка PHP, `register_globals`, которая позволяет входным Web-параметрам быть зарегистрированными непосредственно в переменных сценария. Например, поле формы или параметр URL x в вашем сценарии будут сохранены в переменной $x. К сожалению, включение этой возможности означает, что клиенты получают возможность устанавливать переменные в ваших сценариях. Это определенный риск для системы безопасности, и разработчикам PHP сейчас рекомендуется отключать настройку `register_globals`. Программа `script_param()` вынужденно использует только массивы, предназначенные непосредственно для входных параметров, которые являются более безопасными и работают независимо от установок `register_globals`.

Теперь, когда у нас имеется поддержка удобной выборки входных Web-параметров, мы можем воспользоваться ею при написании сценария `score_entry.php`. Этот сценарий должен иметь возможность передачи информации от одного вызова к другому. Для этого воспользуемся параметром `action`, который может быть получен при выполнении сценария следующим образом:

```
$action = script_param ("action");
```

Если параметр не установлен, это означает, что сценарий вызывается впервые; в противном случае он должен проверить значение $action, чтобы определить, что необходимо сделать. Вот основной "скелет" сценария `script_entry.php`:

```
<?php
# score_entry.php - Сценарий ввода результатов проекта
# "Учет успеваемости"
include "sampdb.php";
# объявление констант операций
define ("SHOW_INITIAL_PAGE", 0);
define ("SOLICIT_EVENT", 1);
define ("ADD_EVENT", 2);
define ("DISPLAY_SCORES", 3);
define ("ENTER_SCORES", 4);

/* ... здесь место для функций ... */

$title = "Проект Учет успеваемости - ввод оценок";
html_begin ($title, $title);
sampdb_connect()
   or die ("Невозможно подключиться к серверу");
# определение выполняемого действия (по умолчанию, если
# ничего не задано, то отображение стартовой страницы)
$action = script_param ("action");
if (!isset ($action))
   $action = SHOW_INITIAL_PAGE;
switch ($action)
```

```
{
case SHOW_INITIAL_PAGE: # отобразить стартовую страницу
  display_events ();
  break;
case SOLICIT_EVENT: # запрос информации о новом экзамене
  solicit_event_info ();
  break;
case ADD_EVENT: # добавить новый экзамен в базу данных
  add_new_event ();
  display_events ();
  break;
case DISPLAY_SCORES: # отобразить результаты выбранного экзамена
  display_scores ();
  break;
case ENTER_SCORES: # ввести новые или отредактированные результаты
  enter_scores ();
  display_events ();
  break;
default:
  die ("Неизвестный код действия ($action)");
}
html_end ();
?>
```

Переменная `$action` может принимать несколько значений, которые затем обрабатываются оператором `switch()`. Оператор `switch()` языка PHP во всем аналогичен своему "собрату" из языка C. В сценарии `score_entry.php` он применяется для определения операции, которую необходимо предпринять в зависимости от передаваемого параметра. Для того чтобы не прибегать к программированию с применением литеральных констант в сценарии, для объявления констант воспользуемся конструкцией `define()`.

Посмотрим, какие функции необходимы для выполнения этих действий. Первая из них, `display_events()`, выбирает и отображает строки из таблицы `event`. В строках перечисляется идентификатор, дата и тип экзамена (тест это или викторина), при этом идентификатор экзамена является ссылкой, на которой нужно щелкнуть, чтобы выбрать результаты этого экзамена:

```
function display_events ()
{
  print ("Выберите экзамен, щелкнув на его номере, \n");
  print ("или выберите New Event для создания записи о
            новом экзамене: <br /><br />\n ");
  $query = "SELECT event_id, date, type FROM event ORDER BY event_id";
  $result_id = mysql_query ($query)
    or die ("Невозможно выполнить запрос");
  print ("<table border=\"1\">\n");
  # Распечатать строку заголовков столбцов
  print ("<tr>\n");
  display_cell ("th", "Event ID");
  display_cell ("th", "Date");
  display_cell ("th", "Type");
  print ("</tr>\n");
  # Отобразить перечень существующих экзаменов.
  # связать каждый идентификатор со ссылкой, которая отобразит
  # оценки по экзамену; используется mysql_fetch_array(), что
  # позволяет обращаться к столбцам по имени
```

```
  while ($row = mysql_fetch_array ($result_id))
  {
    print ("<tr>\n");
    $url = sprintf ("%s?action=%s&event_id=%s",
                            script_name (),
                            urlencode (DISPLAY_SCORES),
                            urlencode ($row["event_id"]));
    display_cell ("td",
                            "<a href=\"$url\">"
                            . htmlspecialchars ($row["event_id"])
                            . "</a>",
                            FALSE);
    display_cell ("td", $row["date"]);
    display_cell ("td", $row["type"]);
    print ("</tr>\n");
  }
  # добавить еще одну ссылку на новый экзамен
  print ("<tr align=\"center\">\n");
  $url = sprintf ("%s?action=%s",
                        script_name (),
                        urlencode (SOLICIT_EVENT));
  display_cell ("td colspan=\"3\"",
                        "<a href=\"$url\">" . "Create New Event" .
"</a>",
                        FALSE);
  print ("</tr>\n");
  print ("</table>\n");
}
```

Все URL для гиперссылок, вызывающих сценарий `score_entry.php`, созданы с помощью функции `script_name()`, которая определяет собственный путь сценария. (Его можно найти в файле `sampdb.php`.) Функция `script_name()` полезна, так как она позволяет нам не прописывать имя сценария в коде; ведь если жестко прописать имя в сценарии, а затем по каким-то причинам его переименовать, это приведет к сбою в работе сценария.

Функция `script_name()` аналогична сценарию `script_param()` в том, что она имеет доступ к глобальным массивам PHP. Однако он использует и другие массивы, поскольку имя сценария является частью информации, которая предоставляется Web-сервером не как часть входных параметров:

```
function script_name ()
{
  global $HTTP_SERVER_VARS, $PHP_SELF;
    if (isset ($_SERVER["PHP_SELF"]))
        return ($_SERVER["PHP_SELF"]);
    if (isset ($HTTP_SERVER_VARS["PHP_SELF"]))
        return ($HTTP_SERVER_VARS["PHP_SELF"]);
    return ($PHP_SELF);
}
```

Функция `display_cell()`, которая используется для генерации табличного отображения:

```
# Отобразите ячейку таблицы HTML. $tag - это имя тэга ("th" или "td"
# для данных заголовка или строки данных), $value - отображаемое зна-
чение, а
# $encode должно иметь значение true или false, показывающее, надо ли
```

```
производить
# HTML-кодировку значения перед его отображением. $encode – не обязательно
# и по умолчанию имеет значение true.
function display_cell ($tag, $value, $encode = TRUE)
{
  if ($value == "") # значение пустое или не установлено?
    $value = " ";
  else if ($encode) # по необходимости выполните HTML-перекодировку
    $value = htmlspecialchars ($value);
  print ("<$tag>$value</$tag>\n");
}
```

Если в таблице, которую отображает функция `display_events()`, выбрана ссылка "New Event", сценарий вызывается повторно с действием SOLICIT_EVENT (ЗАДАТЬ ЭКЗАМЕН). В результате запускается вызов `solicit_event_info()`, отображающий форму ввода информации о новом экзамене:

```
function solicit_event_info ()
{
  printf ("<form method=\"post\" action=\"%s?action=%d\">\n",
  script_name (),
  urlencode (ADD_EVENT));
  print ("Ввести информацию о новом экзамене:< br /><br />\n");
  print ("Дата: <input type=text name=\"date\"");
  print (" value=\"\" size=10> ");
  print ("<input type=\"text\" name=\"date\" value=\"\" size=\"10\"
/>\n");
  print ("<br />\n");
  print ("Тип: ");
  print ("<input type=\"radio\" name=\"type\" value=\"T\"");
  print (" checked=\"checked\" />Test\n");
  print ("<input type=\"radio\" name=\"type\" value=\"Q\" />Quiz\n");
  print ("<br /><br />\n");
  print ("<input type=\"submit\" name=\"button\" value=\"Submit\" />\n");
  print ("</form>\n");
}
```

Форма, генерируемая процедурой `solicit_event_info()`, содержит редактируемое поле для ввода данных, пару радиокнопок для определения типа вводимых данных (это тест или викторина) и кнопку Submit. При подтверждении формы вызывается сценарий `score_entry.php` с действием ADD_EVENT (ДОБАВИТЬ ЭКЗАМЕН). Вызывается функция `add_new_event()` для добавления новой строки в таблице `event`.

```
function add_new_event()
{
  $date = script_param ("date"); # получить дату и тип экзамена
  $type = script_param ("type"); # введенных пользователем
  if (empty ($date))             # проверить ввод даты
    die ("Дата не задана");
  if (!preg_match ('/^\d+\D\d+\D\d+$/', $date))
    die ("Введите дату в формате ISO (CCYY-MM-DD) ");
  if ($type != "T" && $type != "Q")
    die ("Неверный тип экзамена");
  $date = quote_value ($date);
  $type = quote_value ($type);
  $query = sprintf ("INSERT INTO event (date,type)
```

```
VALUES($date,$type)",
   if (!mysql_query ($query))
     die ("Невозможно добавить экзамен!");
}
```

В функции add_new_event() для осуществления доступа к значениям полей используются библиотечная функция script_param(), глобальные переменные, задействованные в форме ввода новых экзаменов (дата и тип доступны как переменные $date и $type). Самая минимальная семантическая проверка заключается в следующем.

- Проверка того, что дата не пустая; в таблицу event вводится новая запись в формате ISO. Функция preg_match() выполняет поиск по шаблону в формате ISO:

  ```
  preg_match ('/^\d+\D\d+\D\d+$/', $date)
  ```

  После этого главная программа отображает список экзаменов, уже включающий новый экзамен. Это позволяет вводить данные по новому экзамену и начать ввод оценок.

  Одинарные кавычки здесь используются для того, чтобы предотвратить интерпретацию символа доллара и обратных косых черт как управляющих символов. Тест можно считать успешным, если дата содержит три последовательности цифр, разделенных нецифровыми символами. Это не стопроцентная гарантия, но эту функциональность очень легко добавить в любой сценарий, и она позволит "отловить" множество ошибок общего характера.

- Тип экзамена должен быть одним из разрешенных значений столбца type таблицы event ('T' или 'Q').

Если параметры соответствуют критериям проверки, функция add_new_event() добавляет новую запись в таблицу event. В коде сборки запроса используется функция quote_value(), которая позволяет убедиться в том, что значения взяты в кавычки правильно и запрос готов. После выполнения оператора функция add_new_event() возвращает управление главной части сценария (оператор switch), который отображает перечень экзаменов снова и готов к выбору нового экзамена и ввода значений для него.

В процессе выбора нового элемента из списка экзаменов, отображенных функцией display_events(), сценарий score_entry.php вызывает функцию display_scores(). Ссылки на экзамены содержат номер экзамена, закодированный как параметр event_id, поэтому display_scores() принимает значение параметра, проверяет его тип (он должен быть целым числом) и использует его в запросе для того, чтобы осуществить поиск любого учащегося и любой текущей оценки, поставленной этому учащемуся:

```
function display_scores ()
{
   # Получить идентификатор экзамена, который должен быть целочисленным
   $event_id = script_param ("event_id");
   if (!preg_match ('/^\d+$/', $event_id))
     die ("Неверный идентификатор экзамена");
   # выбрать оценки для заданного экзамена
   $query = sprintf ("
     SELECT
```

```
        student.student_id, student.name, event.date,
        score.score AS score, event.type
    FROM student, event
        LEFT JOIN score ON student.student_id = score.student_id
        AND event.event_id = score.event_id
    WHERE event.event_id = %s
    ORDER BY student.name
  ", quote_value ($event_id));
  $result_id = mysql_query ($query)
  $result = mysql_query ($query)
    or die ("Невозможно выполнить запрос");
  if (mysql_num_rows ($result_id) < 1)
    die ("Для указанного экзамена данные на найдены");
  printf ("<form method=\"POST\"
action=\"%s?action=%s&event_id=%s\">\n",
                script_name (),
                urlencode (ENTER_SCORES),
                urlencode ($event_id));
  # распечатать оценки в табличном виде
  $row_num = 0;
  while ($row = mysql_fetch_array ($result))
  {
    # распечатать информацию об экзамене
    # перед первой строкой таблицы
    if ($row_num == 0)
    {
        printf ("Идентификатор экзамена: %s, дата экзамена: %s, тип
экзамена: %s\n",
            htmlspecialchars ($event_id),
            htmlspecialchars ($row["date"]),
            htmlspecialchars ($row["type"]));
        print ("<br /><br />\n");
        print ("<table border=\"1\">\n");
        print ("<tr>\n");
        display_cell ("th", "Имя");
        display_cell ("th", "Оценка");
        print "</tr>\n";
    }
    ++$row_num;
    print ("<tr>\n");
    display_cell ("td", $row["name"]);
    $col_val = sprintf ("<input type=\"text\" name=\"score[%s]\ "",
    htmlspecialchars ($row["student_id"]));
    $col_val .= sprintf (" value=\"%s\" size=\"5\" /><br />\n",
    htmlspecialchars ($row["score"]));
    display_cell ("td", $col_val, FALSE);
    print ("</tr>\n");
  }
  print ("</table>\n");
  print ("<br />\n");
  print ("<input type=\"submit\" name=\"button\" value=\"Submit\"
/>\n");
  print "</form>\n";
}
```

Запрос выборки информации об оценках для определенного экзамена, использованный в функции `display_scores()`, не является простым объединением таблиц. Прежде всего потому, что он не выбрал бы ни одной строки по уча-

щемуся, не имеющему оценок за экзамен. В частности, для нового экзамена объединение ничего не выберет, и у нас получится пустая форма ввода данных! Для того чтобы выборка производилась для всех учащихся, воспользуемся объединением типа LEFT JOIN. Это позволит произвести выборку данных по всем учащимся, независимо от наличия у них оценок. (База запроса аналогична той, в соответствии с которой производилась выборка в главе 3, "Синтаксис и использование языка SQL".) Тот запрос выбирал только отсутствующие результаты, здесь мы выбираем только результаты соответствующих экзаменов.

Результаты закодированы в формате именованных полей вида `score[n]`, где *n* является значением идентификатора `student_id`. После этого форма отсылается обратно на Web-сервер, язык PHP преобразует эти поля в элементы массива `$score`. Из этого массива данные формы можно получить следующим образом:

```
$score = script_param ("score");
```

Элементы массива будут отсортированы по идентификатору учащегося. Таким образом, будет очень легко соотнести учащегося с соответствующей оценкой, имеющейся в форме. Содержимое форм обрабатывается функцией `enter_scores()`, имеющей следующий вид:

```
function enter_scores ()
{
   # Получить целочисленный идентификатор экзамена
   $event_id = script_param ("event_id");
   $score = script_param ("score");
   if (!preg_match ('/^\d+$/', $event_id))
     die ("Неверный идентификатор экзамена");
   $invalid_count = 0;
   $blank_count = 0;
   $nonblank_count = 0;
   reset ($score);
   while (list ($student_id, $newscore) = each ($score))
   {
     $newscore = trim ($newscore);
     if (empty ($newscore))
     {
         # если оценка учащегося отсутствует в форме, удалить все его
оценки,
         # которые содержались до этого в таблице
         ++$blank_count;
         $query = sprintf ("
                                 DELETE FROM score
                                 WHERE event_id = %s AND student_id =
%s
                                 ",
                                 quote_value ($event_id),
                                 quote_value ($student_id));
     }
     else if (!preg_match ('/^\d+$/', $newscore)) # должен быть целым
     {
         ++$nonblank_count;
         $query = sprintf ("
                                 REPLACE INTO score
(event_id,student_id,score)
                                 VALUES(%s,%s,%s)
                                 ",
```

```
                                    quote_value ($event_id),
                                    quote_value ($student_id),
                                    quote_value ($newscore));
    }
    else
    {
        ++$invalid_count;
        continue;
    }
    if (!mysql_query ($query))
        die ("ошибка ввода оценки, идентификатор экзамена
    .$event_id, идентификатор учащегося $student_id");
  }
  printf ("Введено оценок: %d<br />\n ", $nonblank_count);
  printf ("Отсутствует оценок: %d<br />\n ", $blank_count);
  printf ("Ошибочных оценок: %d<br />\n ", $invalid_count);
  print ("<br />\n ");
}
```

Значения идентификаторов учащихся и соответствующие им оценки получены при просмотре массива `$score` с помощью функции языка PHP `each()`. Оценки обрабатываются следующим образом.

- Оценка пустая, и вводить нечего. При этом удаляется прежняя оценка, если она существовала. (Возможно, она была раньше ошибочно введена, например, для учащегося, который в действительности на экзамене отсутствовал.) Если у учащегося не проставлена оценка, оператор DELETE не найдет записи, которую необходимо удалить, и это пройдет совершенно без последствий.
- Оценка не пустая. В этом случае производится элементарная проверка значения. Результат считается допустимым, если эта функция возвращает пустое или целое значение. Однако вводимые в форму значения всегда кодируются как строки, поэтому их нельзя проверить с помощью функции `is_int()`. Эти функции вернут значение "FALSE", даже если значение содержит только цифры. В этом случае лучше сработает проверка на соответствие шаблону. Вот проверка, возвращающая значение "TRUE", когда все символы строки от начала до конца строки являются цифрами:

  ```
  preg_match ('/^\d+$/', $str)
  ```

  Если оценка прошла все проверки, ее можно добавить в таблицу `score`. В приведенном здесь примере используется оператор REPLACE, а не INSERT. Если учащийся не имеет оценки за экзамен, оператор REPLACE сработает как оператор INSERT и добавит новую запись. В противном случае оператор REPLACE заменит старую оценку на новую.

Вот функциональная нагрузка сценария `score_entry.php`. Ввод оценок и их исправление теперь можно выполнять с помощью Web-броузера. Одним совершенно очевидным недостатком является пробел в системе безопасности, так как любой желающий может подключиться к Web-узлу и внести свои оценки. Сценарий, который будет позднее создан нами для редактирования записей "Исторической Лиги", будет снабжен простейшей системой авторизации доступа, которую можно применить и в этом сценарии. Более сложную аутентификацию доступа можно обеспечить при помощи SSL-соединения. Но эта тема выходит за рамки данной книги.

Ниже приведен перечень некоторых доработок сценария score_entry.php, которые можно выполнить.

- Вывод информации о плохих оценках.
- Ввод оценок в пределах одной транзакции и откат при вводе плохой оценки. Для этого необходимо убедиться в том, что тип таблицы score поддерживает транзакции (например, тип InnoDB). При этом цикл ввода оценок необходимо предварить оператором BEGIN и завершить его оператором COMMIT или ROLLBACK с соответствующим анализом значения $invalid_count.

Если вы все же решите модифицировать сценарий с использованием транзакций, необходимо воспользоваться неустойчивым соединением. При остановке работы сценария в середине транзакции вы сможете сделать "откат" транзакции, а такое возможно только при неустойчивых соединениях. PHP закроет соединение, а сервер MySQL автоматически осуществит откат любой выполняющейся транзакции, если клиент завершит свою работу с ошибкой. При устойчивом соединении язык PHP оставит соединение открытым, и вполне вероятно, что откат незавершенной транзакции сделан не будет.

Викторина "Президенты США"

Одна из целей создания Web-узла "Исторической Лиги" — создание интерактивной викторины наподобие той, которую публикует лига в разделе для детей своего информационного бюллетеня "Chronicles of US Past". У нас уже есть таблица president, содержащая информацию о президентах США. Ею можно воспользоваться в качестве источника данных для викторины. Для этого напишем сценарий pres_quiz.php.

Основная идея заключается в выборе президента в произвольном порядке, формулировании вопроса, касающегося этого президента, ожидании ответа от пользователя, после чего проверяется правильность ответа. Для простоты ограничимся вопросом о месте рождения президента. Еще одним способом упрощения является представление вопросов в формате многоальтернативного выбора. Это будет проще для тех пользователей, которым требуется только сделать выбор из набора вариантов, а не вводить определенный ответ. Это проще и для нас, поскольку в этом случае не нужен какой-то специальный анализ того, что может ввести пользователь. Нам будет достаточно сравнить выбор, сделанный пользователем, с искомым значением.

Сценарий pres_quiz.php, который осуществляет эту викторину, выполняет две задачи. Первая: для первоначального его вызова он должен создать и отобразить новый вопрос на основании информации из таблицы president. Вторая: если сценарий был вызван после ответа пользователя, ответ необходимо проверить и дать знать пользователю о том, был он верным или нет. Если да, то сценарий должен вывести новый вопрос. Если в ответе содержится ошибка, вопрос следует повторить.

Алгоритм сценария предельно прост. Если пользователь не дает ответа, сценарий отображает первоначальную страницу запроса; в противном случае он проверяет ответ:

```
<?php
# pres_quiz.php - сценарий опроса пользователей о месте рождения президентов США
include "sampdb.php";
# ... здесь необходимо разместить функции обработки вопросов викторины
...
$title = "Викторина "Президенты США";
html_begin ($title, $title);
sampdb_connect ()
  or die ("Извините, невозможно подключиться к базе данных, нет материалов по викторине");
$response = script_param ("response");
if (!isset ($response)) # первый вызов
  present_question ();
else # пользователь дал ответ на вопрос
  check_response ();
html_end ();
?>
```

Для генерирования вопросов воспользуемся возможностью, которая появилась в MySQL 3.23. Это функция ORDER BY RAND(), которая позволяет производить выборку строк из таблицы в произвольном порядке. Например, для того, чтобы выбрать произвольное имя президента и дату его рождения, необходимо сделать запрос:

```
SELECT CONCAT(first_name, ' ', last_name) AS name,
  CONCAT(city, ', ', state) AS place
  FROM president ORDER BY RAND() LIMIT 1
```

Это будет имя президента и его место рождения, которые являются правильными ответами на вопрос "Где родился этот президент?" Кроме того, для того, чтобы запутать пользователя, нам необходимо предусмотреть несколько неправильных ответов. Для этого воспользуемся аналогичным запросом:

```
SELECT DISTINCT CONCAT(city, ', ', state) AS place
FROM president ORDER BY RAND()
```

По результату работы этого запроса будут выбраны первые четыре значения, отличные от правильного ответа. Ключевое слово DISTINCT в этом запросе заключается в попытке избежать возможности выбора одного места рождения больше одного раза. Ключевое слово DISTINCT не потребуется, если места рождения уникальны. В обратном можно убедиться следующим образом:

```
mysql> SELECT city, state, COUNT(*) AS count FROM president
    -> GROUP BY city, state HAVING count > 1;
+-----------+-------+-------+
| city      | state | count |
+-----------+-------+-------+
| Braintree | MA    |     2 |
+-----------+-------+-------+
```

Функция, генерирующая вопрос и предлагающая вероятные ответы, имеет следующий вид:

```
function present_question ()
{
  # запрос по выборке президента и выборке места его рождения
  $query = "SELECT CONCAT(first_name, ' ', last_name) AS name, "
           . " CONCAT(city, ', ', state) AS place"
           . " FROM president ORDER BY RAND() LIMIT 1";
```

```
  $result_id = mysql_query ($query)
    or die ("Невозможно выполнить запрос");
  $row = mysql_fetch_array ($result_id)
    or die ("Невозможно выбрать результат");
  $name = $row["name"];
  $place = $row["place"];
  # Создание набора мест рождения для вывода.
  # Инициализировать массив $choices, хранящий пять
  # мест рождения,
  # одно из которых и является правильным ответом.
  $query = "SELECT DISTINCT CONCAT(city, ', ', state) AS place"
           . " FROM president ORDER BY RAND()";
  $result_id = mysql_query ($query)
    or die ("Невозможно выполнить запрос");
  $choices[] = $place; # Инициализировать массив с
                       # правильным выбором
  while (count ($choices) < 5 && $row = mysql_fetch_array
($result_id))
  {
    if ($row["place"] == $place)
        continue;
        $choices[] = $row["place"]; # добавить другой вариант
  }
  # расположить варианты ответа случайным образом и
  # отобразить форму
  srand ((float) microtime () * 10000000);
  shuffle ($choices);
  display_form ($name, $place, $choices);
}
```

Функция `present_question()` в этом виде не будет работать, если версия используемой вами СУБД MySQL ниже 3.23.2, так как в старых версиях возможность задания функций в выражении `ORDER BY` не предусмотрена. В комментариях, имеющихся в исходном коде сценария `pres_quiz.php`, описаны изменения, которые достаточно внести для того, чтобы обойти это ограничение.

Функция `display_form()`, вызванная функцией `present_question()`, генерирует вопрос викторины, используя форму, отображающую имя президента, набор радиокнопок с перечислением возможных вариантов ответа и кнопку Submit. Эта форма выполняет двойную функцию: она должна предоставлять информацию викторины клиенту и передавать эту информацию обратно на Web-сервер после ответа пользователя. Ответ следует проверять на правильность.

Представление вопроса викторины заключается в отображении имени президента и вероятного выбора мест рождения, которые достаточно односложны. Возможность проверки ответа и возможное повторное отображение вопроса достаточно проблематичны. Для этого требуется, чтобы у нас был доступ к правильному ответу и ко всей информации, необходимой для перегенерации вопроса. Одним из способов осуществления этого является использование скрытых полей. В этом случае форма сама по себе уже содержала всю необходимую для нее информацию. Эти поля становятся частью формы и возвращаются, когда пользователь дает ответ, оставаясь при этом невидимыми пользователю.

Этими скрытыми полями будут `name`, `place` и `choice`, предназначенные для хранения имени президента, места его рождения и набора возможных вариантов ответов соответственно. Варианты кодируются в виде одной строки с помощью

функции implode(), объединяющей значения в одну строку со специальным разделительным символом. (Этот специальный символ нам необходим на будущее, чтобы разбить строку с помощью функции explode(), если понадобится повторно вывести вопрос на экран.) Функция, отображающая форму, имеет следующий вид:

```
function display_form ($name, $place, $choices)
{
  printf ("<form method=\"POST\" action=\"s\"n", script_name ());
  hidden_field ("name" $name);
  hidden_field ("place" $place);
  hidden_field ("choices" implode ("#", $choices));
  printf ("Где родился %s?<br /><br />\n", htmlspecialchars ($name));
  for ($i = 0; $i < 5; $i++)
  {
   radio_button ("response", $choices[$i], $choices[$i], FALSE);
   print ("<br />\n");
  }
  print ("<br />\n");
  submit_button ("submit", "Submit");
  print ("</form>\n"
}
```

Для генерирования полей формы функция display_form() использует несколько вспомогательных функций. Первой является hidden_field(), которая предназначена для генерирования тэга <input> для скрытых полей:

```
function hidden_field ($name, $value)
{
  printf ("<input type=\"hidden\" name=\"%s\"
            value=\"%s\"/>\n",
            "hidden",
            htmlspecialchars ($name),
            htmlspecialchars ($value));
}
```

Так как функция hidden_field() является программой общего назначения и может понадобиться в самых разных сценариях, совершенно естественным будет поместить ее в наш библиотечный файл sampdb.php. Обратите внимание на то, что в ней для кодировки атрибутов имен и значения тэга <input> используется функция htmlspecialchars(), если переменная $name или $value содержит такие специальные символы, как кавычки.

Две другие вспомогательные функции, radio_button() и submit_button(), реализованы следующим образом:

```
function radio_button ($name, $value, $label, $checked)
{
  printf ("<input type=\"%s\" name=\"%s\" value=\"%s\"%s />%s\n",
            "radio",
            htmlspecialchars ($name),
            htmlspecialchars ($value),
            ($checked ? " checked=\"checked\" : ""),
            htmlspecialchars ($label));
}

function submit_button ($name, $value)
{
```

```
  printf ("<input type=\"%s\" name=\"%s\" value=\"%s\" />\n",
          "submit",
          htmlspecialchars ($name),
          htmlspecialchars ($value));
}
```

После того как выбор сделан и пользователь его подтвердил, результат кодируется в поле ответа и отсылается на Web-сервер как значение параметра ответа. Значение ответа можно определить вызовом функции `script_param()`, которая, кроме того, позволяет определить, был сценарий вызван впервые или это уже ответ на ранее отображенную форму. Во время первого вызова параметр не будет установлен. Таким образом, в основном теле сценария можно определить, какие действия ему надо выполнить дальше, проверив наличие или отсутствие параметра:

```
$response = script_param ("response");
if (!isset ($response)) # первоначальный вызов
  present_question ();
else # пользователь дал ответ на вопрос
  check_response ();
```

Нам осталось написать функцию `check_response()`, которая сравнивает ответ пользователя с правильным ответом. Для этого нам необходимы значения, хранящиеся в скрытых полях `name`, `place` и `choices`. Правильный ответ закодирован в поле `place`, а ответ пользователя — в поле `response`. Таким образом, нам остается сравнить переменную `$place` с переменной `$response`. Основываясь на полученном результате сравнения, можно запрограммировать реакцию программы и вывести на экран новый вопрос викторины или повторить прежний, если ответ был неверным:

```
function check_response ()
{
  $name = script_param ("name");
  $place = script_param ("place");
  $choices = script_ param ("choices");
  $response = script_param ("response");
  # Пользователь ответил правильно?
  if ($response == $place)
  {
    print ("Правильно!<br />\n");
    printf ("%s родился в %s.<br />\n",
            htmlspecialchars ($name),
            htmlspecialchars ($place));
    print ("Ответьте на следующий вопрос:<br /><br />\n");
    present_question();
  }
  else
  {
    printf ("\"%s\"Неправильно. Попробуйте еще раз.<br /><br />\n",
            htmlspecialchars ($response));
    $choices = explode ("#", $choices);
    display_form ($name, $place, $choices);
  }
}
```

Вот и все. Теперь добавьте ссылку на сценарий `pres_quiz.php` из стартовой Web-страницы "Исторической Лиги", и посетители смогут проверить свою эрудицию.

Скрытые поля небезопасны

Сценарий `pres_quiz.php` использует в своей работе скрытые поля. Здесь они являются средством передачи информации, необходимой для повторного вызова сценария, которую пользователь видеть не должен. Это очень хорошо для сценария, рассчитанного только для развлечения. Но скрытые поля не должны использоваться для хранения информации, к которой пользователи никогда не должны иметь прямого доступа, так как они при этом никак не защищены. Для того чтобы посмотреть, почему так нельзя делать, установите сценарий `pres_quiz.php` в каталоге `ushl` дерева документов вашего Web-сервера и запросите его с помощью броузера. После этого с помощью команды View Source броузера просмотрите HTML-код страницы викторины, и вы все увидите — содержимое скрытого поля `place`, содержащее правильный ответ на текущий вопрос викторины. Это значит, что викторину "обмануть" очень просто. Для данного конкретного приложения это не самая большая проблема, но на его примере видно, что скрытые поля по крайней мере небезопасны. Поэтому никогда не используйте их для хранения действительно конфиденциальной информации.

Интерактивное редактирование данных о членах "Исторической Лиги"

Наш последний сценарий, `edit_member.php`, позволит членам "Исторической Лиги" самостоятельно редактировать данные о себе непосредственно на Web-странице. Теперь для этого не обязательно связываться с офисом "Исторической Лиги". Такая возможность позволит сделать каталог членов "Исторической Лиги" более актуальным, что уменьшит нагрузку на секретаря лиги.

Но сразу же нужно предупредить, что необходимо будет предпринять меры для того, чтобы никто иной, кроме самого члена "Исторической Лиги", не смог модифицировать свою запись. Это значит, что необходима определенная система безопасности. Продемонстрируем простой метод аутентификации с использованием возможностей СУБД MySQL. Запомним пароль пользователя и при попытке скорректировать запись будем требовать его ввода. Этот сценарий будет работать следующим образом.

- При первоначальном вызове сценарий `edit_script.php` выводит форму, содержащую поля, предназначенные для ввода идентификатора пользователя и его пароля.
- После этого сценарий проверяет таблицу `password`, используя в качестве ключа идентификатор пользователя, чтобы сравнить введенный пароль с паролем, хранящимся в таблице паролей. Если пароль совпадает, сценарий ищет запись с данными этого члена "Исторической Лиги" и готовит ее для редактирования.
- После того как форма будет подтверждена, запись модифицируется с помощью данных, содержащихся в форме.

Для любой из этих задач необходимо создавать какие-то пароли. Самый простой способ сделать это — сгенерировать их случайным образом. Ниже приведены операторы, которые создают таблицу `member_pass`, затем создают пароль для каждого члена "Исторической Лиги", генерируя контрольную сумму методом MD5 как случайное число. Вероятно, право выбора пароля будет возложено на самого пользователя, но для примера возьмем следующий SQL-оператор и запустим его с помощью `mysql`:

```
mysql> CREATE TABLE member_pass (
    -> member_id INT UNSIGNED NOT NULL PRIMARY KEY,
    -> password CHAR(8));
mysql> INSERT INTO member_pass (member_id, password)
    -> SELECT member_id, LEFT(MD5(RAND()), 8) AS password FROM member;
```

Функция MD5() до MySQL 3.23.2 отсутствовала. Поэтому другой способ создания 8-символьных значений, который будет работать с любой версией, будет следующим:

```
mysql> INSERT INTO member_pass (member_id, password)
    -> SELECT member_id, FLOOR(RAND()*99999999) AS password FROM
member;
```

Полученные значения будут менее разнообразными, чем те, которые можно получить с помощью функции MD5(), так как они будут состоять только из цифр.

Добавим также специальную запись члена 0 с паролем, который будет служить паролем администратора (суперпользователя). Этот пользователь будет иметь доступ к любым данным:

```
mysql> INSERT INTO member_pass (member_id, password) VALUES(0,
'bigshot');
```



Примечание

До того как создать таблицу member_pass, можно удалить сценарий samp_browse.pl из каталога сценариев вашего Web-сервера. (Этот сценарий, написанный нами в главе 7, "Программный интерфейс Perl API", позволяет любому пользователю просматривать содержимое любой таблицы в базе данных sampdb, включая и таблицу member_pass. Таким образом, можно будет просмотреть пароль любого члена "Исторической Лиги" или пароль администратора.)

После того как таблица member_pass будет создана, можно приступать к созданию сценария edit_member.php. Вот "костяк" сценария edit_member.php:

```
<?php
# edit_member.php - редактирование данных о членах
# "Исторической Лиги" через Web

include ("sampdb.inc");
# edit_member.php - редактирование данных о членах
# "Исторической Лиги" через Web
include ("sampdb.inc");

# определение констант
define ("SHOW_INITIAL_PAGE", 0);
define ("DISPLAY_ENTRY", 1);
define ("UPDATE_ENTRY", 2);

# ... здесь поместить функции обработки ввода ...
$title = "Форма редактирования данных о членах "Исторической Лиги
США";
html_begin ($title, $title);
sampdb_connect ()
   or die ("Невозможно подключиться к серверу");
# определить, какое действие выполняется (по умолчанию, если ничего не
определено,
# отобразить стартовую страницу)
$action = script_param ("action");
```

```
if (!isset ($action))
  $action = SHOW_INITIAL_PAGE;
switch ($action)
{
case SHOW_INITIAL_PAGE: # вывести стартовую страницу
  display_login_page ();
  break;
case DISPLAY_ENTRY: # отобразить редактируемую запись
  display_entry ();
  break;
case UPDATE_ENTRY: # сохранить модифицированную запись в базе данных
  update_entry ();
  break;
default:
  die ("Неизвестный код действия ($action)");
}
html_end ();
?>
```

Стартовая страница представляется функцией `display_login_page()`, которая генерирует форму запроса идентификатора и пароля члена "Исторической Лиги":

```
function display_login_page ()
{
  printf ("<form method=\"POST\" action=\"%s?action=%s\">\n",
           script_name (),
           urlencode (DISPLAY_ENTRY));
  print ("Введите идентификатор и пароль, \n");
  print ("затем щелкните на кнопке Submit. \n<br /><br />\n");
  print ("<table>\n");
  print ("<tr>");
  print ("<td>Идентификатор члена</td><td>");
  text_field ("member_id", "", 10);
  print ("</td></tr>");
  print ("<tr>");
  print ("<td>Пароль</td><td>");
  password_field ("password", "", 10);
  print ("</td></tr>");
  print ("</table>\n");
  submit_button ("button", "Submit");
  print "</form>\n";
}
```

Для выравнивания названия полей и сами поля, представленные в форме, представляются в виде HTML-таблицы. Здесь это показано на примере двух полей, но это очень полезный прием, особенно тогда, когда создаются формы с наименованиями полей разной длины, что улучшает читабельность и восприятие такой таблицы.

В функции `display_login_form()` используются еще две вспомогательные функции, которые можно найти в библиотечном файле `sampdb.php`. С помощью функции `text_field()` представляется редактируемый текст поля ввода:

```
function text_field ($name, $value, $size)
{
  printf ("<input type=\"%s\" name=\"%s\" value=\"%s\" size=\"%s\"
/>\n",
           "text",
           htmlspecialchars ($name),
```

```
            htmlspecialchars ($value),
            htmlspecialchars ($size));
}
```

Функция password_field() аналогична, за исключением того, что атрибут type равен password (поэтому его показывать не будем).

Когда пользователь вводит идентификатор и пароль в заполняемую форму, параметр action равен DISPLAY_ENTRY, а оператор switch при следующем вызове сценария edit_member.php для проверки пароля и отображения данных о члене организации при вводе правильного пароля вызывает функцию display_entry():

```
function display_entry ()
{
  # Принять параметры сценария; убрать пробелы из идентификатора,
  # не трогая их в пароле, так как необходимо полное соответствие па-
роля.
  $member_id = trim (script_param ("member_id"));
  $password = script_param ("password");
  if (empty ($member_id))
    die ("Идентификатор члена не указан");
  if (!preg_match ('/^\d+$/', $member_id)) # должно быть целым числом
    die ("указан ошибочный индекс (должно быть число)");
  if (empty ($password))
    die ("Пароль не указан");
  if (check_pass ($member_id, $password)) # обычный член
    $admin = FALSE;
  else if (check_pass (0, $password)) # администратор
    $admin = TRUE;
  else
    die ("Неверный пароль");
  $query = sprintf ("
                              SELECT
                                  last_name, first_name, suffix, email,
                                  street, city,
                                  state, zip, phone, interests,
                                  member_id, expiration
                              FROM member WHERE member_id = %s
                              ORDER BY last_name
                    ", quote_value ($member_id));
  $result_id = mysql_query ($query);
  if (!$result_id)
    die ("Невозможно выполнить запрос ");
  if (mysql_num_rows ($result_id) == 0)
    die ("Пользователь с идентификатором = $member_id не найден");
  if (mysql_num_rows ($result_id) > 1)
    die ("Обнаружено несколько пользователей с идентификатором
$member_id");
  printf ("<form method=\"POST\" action=\"%s?action=%s\">\n",
          script_name (),
          urlencode (UPDATE_ENTRY));
  # Добавить идентификатор и пароль как скрытые значения, чтобы при
  следующем вызове
  # сценария можно было сказать, какой записи соответствует запись
  # и что пользователь не вводил пароль повторно.
  hidden_field ("member_id", $member_id);
  hidden_field ("password", $password);
```

```
  # считать результаты запроса и отформатировать их для редактирования
  $row = mysql_fetch_array ($result_id);
  print ("<table>\n");
  # Отобразить идентификатор как статический текст
  display_column ("Идентификатор", $row, "member_id", FALSE);
  # $admin имеет значение true, если пользователь ввел пароль
  администратора,
  # Администратор может редактировать срок действия,
  # а обычные пользователи - не могут.
  display_column ("Срок членства", $row, "expiration", $admin);
  # Отобразить другие значения как редактируемый текст
  display_column ("Фамилия", $row, "last_name");
  display_column ("Имя", $row, "first_name");
  display_column ("Суффикс", $row, "suffix");
  display_column ("Э/почта", $row, "email");
  display_column ("Улица", $row, "street");
  display_column ("Город", $row, "city");
  display_column ("Штат", $row, "state");
  display_column ("Индекс", $row, "zip");
  display_column ("Телефон", $row, "phone");
  display_column ("Интересы", $row, "interests");
  print ("</table>\n");
  submit_button ("button", "Submit");
  print "</form>\n";
}
```

Первое, что должна выполнить функция `display_entry()`, — это проверить пароль. Если для данного идентификатора введенный пароль соответствует значению пароля из таблицы `member_pass` или совпадает с паролем администратора (т.е. паролем для члена 0), сценарий `edit_member.php` отображает запись для редактирования. Функция проверки паролей `check_pass()` генерирует пароль для выборки записи из таблицы `member_pass`:

```
function check_pass ($id, $pass)
{
  $query = sprintf ("SELECT password FROM member_pass WHERE member_id
= %s",
                    quote_value ($id));
  $result_id = mysql_query ($query);
  if (!$result_id)
    die ("Ошибка чтения таблицы password");
  if ($row = mysql_fetch_array ($result_id))
    return ($row["password"] == $pass); # TRUE, если пароль совпадает
  return (FALSE); # запись не обнаружена
}
```

Предположим, что пароль совпадает, функция `display_entry()` просматривает запись из таблицы `member`, соответствующую заданному идентификатору, а затем продолжает генерировать редактируемую форму со значениями, инициализированными из записи. Большинство полей представлено в виде редактируемых текстовых полей, которые может изменить пользователь, но есть два исключения. Во-первых, значение `member_id` отображается в виде статического текста. Это ключевое значение, поэтому оно не подлежит изменениям. Во-вторых, члены лиги не должны самостоятельно изменять срок членства. (В этом случае может появиться соблазн изменить эту дату и не платить ежегодных взносов.) С другой стороны, если административный пароль был выдан во время регистрации, сценарий выводит срок членства в

редактируемом поле. Если секретарь лиги знает этот пароль, он может изменить срок членства для тех, кто возобновил свое членство.

Отображение имен полей и их значений осуществляется с помощью функции `display_column()`. Ее аргументами являются название отображаемого поля, массив, содержащий модифицируемую запись, имя столбца, содержащего значение поля, и логическое значение, показывающее, в какой форме отображать значение, в редактируемой или статической. Последнее значение не обязательно, его значение по умолчанию — TRUE:

```
function display_column ($label, $row, $col_name, $editable = TRUE)
{
  print ("<tr>\n");
  printf ("<td>%s</td>\n", htmlspecialchars ($label));
  print ("<td>");
  if ($editable) # отобразить как редактируемое поле
    text_field ("row[$col_name]", $row[$col_name], 80);
  else # отобразить как статический текст
    print (htmlspecialchars ($row[$col_name]));
  print ("</td>\n");
  print ("</tr>\n");
}
```

Для редактируемых значений функция `display_column()` генерирует текстовые поля, используя имена в формате `row[col_name]`. Таким образом, когда пользователь завершает ввод формы, язык PHP поместит все значения полей в массивы с элементами, обозначенными именами полей. Это существенно упрощает выборку содержимого формы и связывание значений полей с соответствующими им элементами таблицы `member` при изменении записи базы данных. Например, выбрав массив в переменную `$row`, мы получаем доступ к номеру телефона как `$row["phone"]`.

Функция `display_entry()` включает значения `member_id` и `password` в виде скрытых полей таким образом, чтобы они переносились к следующему вызову сценария `edit_script.php` по завершении редактирования записи. Это позволит проверить идентификатор автоматически без повторного запроса. (Обратите внимание на то, что наш простейший метод аутентификации передает пароль в текстовом виде. Вообще-то, это не самое лучшее решение, но, принимая во внимание то, что создание Web-узла "Исторической Лиги" нельзя назвать задачей, требующей мощной степени защиты, этот метод вполне нас удовлетворяет. Вероятно, для решения задач другого рода требуются более серьезные меры защиты (как, например, для выполнения финансовых проводок). Как бы там ни было, функция модификации записи будет выглядеть следующим образом:

```
function update_entry ()
{
  # Принять параметры сценария; отсечь пробелы из идентификатора,
  # не трогая их в пароле, так как необходимо полное соответствие па-
роля; или
  # из row, так как это массив.
  $member_id = trim (script_param ("member_id"));
  $password = script_param ("password");
  $row = script_param ("row");
  $member_id = trim ($member_id);
  if (empty ($member_id))
```

```
  die ("Идентификатор не определен");
if (!preg_match ('/^\d+$/', $member_id)) # должно быть целым
  die ("Задан ошибочный идентификатор (должен быть целым)");
if (!check_pass ($member_id, $password) && !check_pass
(0, $password))
  die ("Ошибочный пароль");
# Сначала нам нужен результирующий набор
# для определения возможности
# хранения пустых значений в столбцах таблицы member
# Вот запрос, позволяющий сделать это без выборки строк.
# Использовать результат запроса для создания
# ассоциативного массива, который ставит в соответствие именам
столбцов значение true/false,
# отражающих, позволяют ли столбцы иметь пустые значения.
$result_id = mysql_query ("SELECT * FROM member WHERE 1 = 0");
if (!$result_id)
  die ("Невозможно сделать запрос к таблице member ");
$nullable = array ();
for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
{
  $fld = mysql_fetch_field ($result_id, $i);
  $nullable[$fld->name] = !$fld->not_null; # TRUE, если может иметь
  пустые значения
}
mysql_free_result ($result_id);
# Просмотрим каждое поле в форме, чтобы создать с
# помощью содержащихся в них значений оператор UPDATE.
$query = "UPDATE member ";
$delim = "SET";
reset ($row);
while (list ($col_name, $val) = each ($row))
{
  $query .= "$delim $col_name=";
  $delim = ",";
  # если значения нет, присвоить значение NULL, если
  # столбец может содержать значения NULL. Это позволит
  # предотвратить попытку присвоить пустое значение
  # столбцу expiration, когда значение должно
  # быть NULL, например:
  $val = trim ($val);
  if (empty ($val))
  {
      if ($nullable[$col_name])
          $query .= "NULL"; # ввести NULL
      else
          $query .= "\'\'"; # ввести пустую строку
  }
  else
      $query .= quote_value ($val);
}
$query .= sprintf (" WHERE member_id = %s", quote_value ($member_id));
if (mysql_query ($query))
  print ("Запись модифицирована успешно. \n");
else
  print ("Запись не модифицирована. \n");
}
```

Сначала проверим пароль, чтобы удостовериться, что никто не пытается обмануть нас с помощью поддельной формы при модификации записи. Модификацию нужно

производить с особой тщательностью и осторожностью, поскольку, когда запись не заполнена, это может означать, что там может быть как значение "NULL", так и пустая строка. Столбец `expiration` может служить примером такого столбца. Значение "NULL" в поле, хранящем дату истечения срока членства, применяется для хранения данных о "пожизненном членстве". Вставка пустой строки приведет к присвоению ошибочного значения "0000-00-00". Таким образом, необходимо знать, какие столбцы могут принимать значения NULL, и вставлять NULL (а не пустую строку), когда такой столбец остается незаполненным.

Для решения этой проблемы рассмотрим метаданные столбца и проверим, как он был объявлен: как принимающий значения "NULL" или как "NOT NULL". Эту информацию можно просмотреть с помощью функции `mysql_fetch_field()`. Это можно сделать с помощью тривиального запроса SELECT, не возвращающего строк:

```
SELECT * FROM member WHERE 1 = 0
```

Запрос возвращает пустой результирующий набор, но этого вполне достаточно для получения метаданных, которые будут использованы для определения возможности хранения значений "NULL" в столбцах таблицы.

Вот и все. Установите сценарий, раздайте членам "Исторической Лиги" их пароли, и они смогут обновлять свои учетные записи в интерактивном режиме.

III

Администрирование MySQL

В этой части...

Глава 9. Введение в администрирование MySQL

Глава 10. Каталог данных MySQL

Глава 11. Общее администрирование MySQL

Глава 12. Безопасность

Глава 13. Резервное копирование и восстановление баз данных

9

Введение в администрирование MySQL

В этой главе...

Обзор задач администрирования
Общее администрирование
Безопасность
Восстановление и поддержка баз данных

С развитием систем баз данных процедуры инсталляции и использования MySQL становятся все проще. Судя по всему, именно простота работы с MySQL стала основной причиной ее широкой популярности среди пользователей. Особенно это относится к тем из них, которые не являются программистами. Безусловно, знания компьютерного профессионала могут оказаться весьма полезными, однако для успешного использования MySQL быть опытным программистом совсем не обязательно.

Однако и полностью без управления MySQL работать также не может. Администратор должен хотя бы иногда проверять согласованность и эффективность ее работы и знать, что делать при возникновении проблем. Вся эта информация будет представлена далее.

Часть III посвящена самым различным аспектам администрирования MySQL. В этой же главе представлено описание всех основных вопросов, которыми необходимо владеть для успешного выполнения задач по администрированию инсталляции MySQL. Она же включает краткий обзор всех обязанностей администратора. Инструкции по их выполнению представлены в следующих главах.

Представленный в этой главе длинный список обязанностей может не на шутку напугать начинающего или неопытного администратора. Каждая указанная в

списке задача действительно очень важна, однако не стоит стараться освоить их все сразу. Гораздо лучше использовать главы этой части в качестве справочного руководства, которое всегда под рукой и в которое всегда можно заглянуть.

Уже имеющие опыт работы администраторы могут заметить, что администрирование MySQL подобно администрированию других систем управления базами данных. В любом случае опыт администрирования других систем просто неоценим. В то же время администрирование MySQL имеет свои уникальные особенности, которые и описываются в этой части.

Обзор задач администрирования

СУБД MySQL состоит из нескольких основных компонентов. Знание их сути и предназначения поможет лучше понять природу управляемой системы и принципы работы ее различных средств. Настоятельно рекомендуется потратить немного времени, чтобы хорошенько разобраться в представленном далее материале. Это значительно упростит вашу дальнейшую работу. В частности, необходимо вникнуть в следующие аспекты работы MySQL.

- **Сервер MySQL.** Сервер `mysqld` выполняет все операции с базами данных и таблицами. Для запуска сервера, мониторинга его работы и перезапуска в случае сбоя применяется программа `mysqld_safe`. (До версии MySQL 4 `mysqld_safe` называлась `safe_mysqld`.) Программа `mysqld_multi` значительно упрощает одновременное управление несколькими серверами.
- **Клиентские программы и программы MySQL.** Для взаимодействия с сервером и выполнения ряда административных задач применяются различные программы MySQL, наиболее важными среди которых являются следующие.
  - `mysql`. Интерактивная программа, позволяющая отправлять SQL-запросы на сервер и просматривать результаты их выполнения.
  - `mysqladmin`. Административная программа, позволяющая выполнять такие операции, как завершение работы сервера, создание и удаление баз данных. Эту же программу можно применять для проверки состояния сервера, если что-то в его работе не так.
  - `mysqlcheck`, `isamchk` и `myisamchk`. Программы, предназначенные для анализа и оптимизации таблиц, а также для восстановления их в случае повреждения.
  - `mysqldump`. Средство резервирования баз данных или их копирования на другой сервер.
- **Язык SQL — язык сервера.** Некоторые задачи администрирования можно выполнить только с помощью программы `mysqladmin`, работающей с командной строкой. Иногда гораздо эффективней справиться с задачей может администратор, который может "общаться" с сервером на его "родном" языке. Предположим, например, что необходимо проверить, почему привилегии пользователя работают вовсе не так, как ожидалось. Напрямую "поговорить" с сервером, к сожалению, нельзя. Зато можно воспользоваться программой-клиентом `mysql` и послать SQL-запрос для анализа таблиц разрешений. Если

же используемая версия MySQL появилась раньше введения оператора GRANT, необходимо сначала использовать команду `mysql` для настройки привилегий каждого пользователя.

- Пользователям, незнакомым с операторами SQL, следует разобраться по крайней мере с основными принципами их работы. Незнание SQL — достаточно серьезное препятствие, и время, затраченное на изучение этого языка, окупится очень быстро. Конечно, чтобы стать настоящим знатоком SQL, придется напряженно учиться и достаточно долго практиковаться. Познакомиться же с основами этого языка можно весьма быстро. Более детально о языке SQL и использовании клиентской программы, работающей с командной строкой, рассказывается в главе 1, "Знакомство с СУБД MySQL и SQL".
- **Каталог данных MySQL.** Каталог данных используется сервером для хранения баз данных и файлов состояния. Важно понимать структуру и содержимое каталога данных, чтобы знать, как сервер представляет свои базы данных и таблицы в файловой системе, где хранятся различные файлы (например, регистрационные) и что в них содержится. Необходимо также уметь управлять распределением дискового пространства, чтобы избежать переполнения раздела с каталогом данных.

Общее администрирование

Общее администрирование включает в себя в основном работу с `mysqld`, сервером MySQL, а также определение прав доступа пользователей. К наиболее важным задачам общего администрирования относятся следующие.

- **Запуск и останов сервера.** Администратор должен уметь запускать и останавливать сервер вручную из командной строки и настраивать автоматический запуск и завершение работы. Важно также знать, как восстанавливать работу сервера в случае сбоя или некорректного запуска сервера.
- **Поддержка учетных записей пользователей.** Администратор должен отчетливо понимать разницу между пользователями MySQL и пользователями UNIX и Windows и уметь настраивать учетные записи пользователей MySQL. Такая настройка зачастую заключается в определении, какой пользователь и с какого компьютера может подключаться к серверу. Новым пользователям необходимо, кроме всего прочего, сообщить свои параметры подключения, чтобы успешно настроить обмен данными с сервером. Настройка учетных записей — это задача администратора, а не пользователей!
- **Поддержка регистрационных файлов.** Администратор должен знать, какие типы регистрационных файлов необходимо проверять и поддерживать, а также каким образом и когда выполнять эти операции. Последовательный просмотр и замена регистрационных журналов позволит избежать переполнения ими используемых дисков.
- **Резервирование и копирование баз данных.** Резервирование баз данных — исключительно важная операция, позволяющая в случае необходимости

восстановить работу системы после сбоя. Желательно, конечно, чтобы имелась возможность восстановить базы данных до того состояния, в котором они находились в момент сбоя. В этом случае потеря данных будет минимальной. Заметьте, однако, что резервирование баз данных отличается от резервирования информации всей системы, выполняемой, например, с помощью традиционной UNIX-программы dump. В процессе активной работы сервера файлы таблиц баз данных, как правило, подвергаются изменениям. Восстановление файлов, зарезервированных в какой-то определенный момент времени, не позволит в полной мере восстановить базу данных, т.е. потеря определенной части данных неизбежна. Более полезными для восстановления базы данных являются файлы, сгенерированные программой `mysqldump`. С ее помощью можно выполнять резервирование без предварительного завершения работы сервера. Также может потребоваться перенести базы данных в случае переполнения дисков.

Иногда возникает необходимость в запуске СУБД на более быстродействующем главном компьютере или создании копии базы данных. В этом случае все содержимое ее каталога можно скопировать на другой компьютер. Администратор должен уметь выполнять и эту процедуру. Файлы баз данных могут зависеть от конфигурации конкретной системы, поэтому простое копирование файлов может не дать достаточно удовлетворительных результатов.

- **Репликация баз данных**. Создание резервной копии базы данных позволяет сделать снимок ее состояния в определенный момент времени. Существует и другая возможность. Это возможность репликации, которая заключается в такой настройке взаимодействия двух серверов, которая позволяет постоянно копировать изменения, произошедшие в базе данных, в соответствующую базу данных, установленную на другом сервере.
- **Настройка сервера**. Пользователи всегда хотят, чтобы сервер работал максимально быстро. Самый быстрый и неэкономный метод повышения производительности сервера — приобретение и установка большего объема памяти и более быстрых дисков. Разбираться в тонкостях работы сервера при применении подобных методов необязательно. Администратору необходимо знать, какие параметры применяются для настройки работы сервера и как их значения повлияют на ситуацию. Одни клиенты пользуются запросами в основном для выборки необходимой информации из баз данных, другие — для добавления и обновления информации. Поэтому решение об изменении значений тех или иных параметров зачастую определяется типом запросов, отправляемых к узлу. Проблемы настройки также включают локализацию серверов (например, для того, чтобы установить правильность используемой кодировки и часового пояса).
- **Установка нескольких серверов**. В некоторых случаях возникает необходимость в запуске сразу нескольких серверов. Она может определяться желанием протестировать работу новой версии MySQL перед полным переходом на нее или обеспечить более высокий уровень защиты для отдельных групп и пользователей. (Последний вариант, в частности, адресован к провайдерам услуг Internet.) В таких ситуациях администратор должен знать,

как устанавливать и настраивать несколько одновременно работающих инсталляций.

- **Обновление MySQL.** Новые версии MySQL появляются достаточно часто. Администратор должен знать, как обновить старую версию для устранения найденных ошибок программного кода и добавления новых возможностей. Следует также хорошо понимать, что иногда имеет смысл воздержаться от обновления, а также уметь выбирать между официальными и тестовыми версиями этого программного продукта.

Безопасность

В процессе инсталляции MySQL важно также обеспечить надежную защиту хранимых данных. Администратор MySQL полностью отвечает за предоставление доступа к каталогам данных и серверу и должен разбираться в следующих вопросах.

- **Защита файловой системы.** В ОС UNIX может поддерживаться несколько учетных записей пользователей, не обладающих полномочиями администратора MySQL. Необходимо проверить, чтобы эти пользователи не имели доступа к каталогам данных. Это позволит защититься от несанкционированного копирования или удаления таблиц баз данных либо чтения регистрационных файлов с критически важной информацией. Администратор должен знать, как настроить учетные записи пользователей UNIX для сервера MySQL, как сделать каталог данных доступным только для владельца и как определить привилегии пользователей сервера.
- **Защита сервера.** Знание принципов работы системы безопасности MySQL поможет правильно настроить пользовательские учетные записи и полномочия доступа. Подключающиеся через сеть к серверу пользователи должны обладать правами доступа только к самой необходимой информации. Полное игнорирование системы безопасности может обернуться предоставлением прав администратора анонимному пользователю.

Восстановление и поддержка баз данных

В душе каждый администратор MySQL надеется избежать повреждения или полного разрушения таблиц баз данных. Однако одних надежд мало. Выполнение определенных процедур позволит минимизировать риск сбоя и быть более подготовленным к негативному развитию событий.

- **Восстановление после сбоя.** Сбой может разрушить результаты даже самых успешных действий, поэтому администратор должен обязательно знать, как отладить и восстановить таблицы баз данных. Восстановление после сбоя — это процесс, который выполняется весьма редко. Однако он очень неприятен и, как правило, сопровождается стрессовым состоянием администратора (которое может еще более усугубляться постоянными звонками и стуком в дверь). Для восстановления применяются средства проверки и восстановления программ `isamchk` и `myisamchk`. Необходимо обязательно научиться восстанавливать информацию из резервных файлов и ис-

пользовать записи регистрационных журналов для восстановления всех изменений, внесенных с момента последнего резервирования.

- **Превентивная поддержка.** Для снижения вероятности повреждения или разрушения баз данных следует постоянно применять программы превентивной поддержки. Необходимо также выполнять и резервирование, хотя, конечно, выполнение мер превентивной поддержки снижает необходимость их использования.

Эта глава посвящена краткому описанию всех основных обязанностей администратора MySQL. В следующих главах эти задачи рассматриваются более детально, а также описываются процедуры их эффективного выполнения. Однако сначала рассматривается каталог данных MySQL. Ведь администратору необходимо четко понимать структуру и содержимое ресурса, на который направлены все операции поддержки. Ниже будут рассмотрены все общие административные задачи, система безопасности MySQL, поддержка работы СУБД и методы устранения всевозможных проблем.

10

Каталог данных MySQL

В этой главе...

Размещение каталога данных
Структура каталога данных
Перемещение содержимого каталога данных

Концептуальные принципы построения большинства систем управления реляционными базами данных одинаковы: все они состоят из набора баз данных, каждая из которых, в свою очередь, включает набор таблиц. Однако каждая система по-своему организует управляемые данные. Не является исключением в этом отношении и MySQL. По умолчанию вся информация, управляемая сервером `mysqld`, содержится в так называемом каталоге данных MySQL (MySQL data directory). В нем хранятся все базы данных и файлы состояния с информацией о функционировании сервера. Пользователь, выполняющий функции администратора MySQL, должен знать структуру этого каталога и уметь использовать его в повседневной работе.

В этой главе даются исчерпывающие ответы на следующие вопросы.

- **Как определить месторасположение каталога данных.** Это необходимо для эффективного управления его содержимым.
- **Как организуется и обеспечивается доступ к базам данных и таблицам сервера.** Эта информация необходима для создания расписания операций превентивной поддержки и восстановления поврежденных таблиц после сбоя.
- **Где размещаются сгенерированные сервером файлы состояния и что они содержат.** Эти файлы содержат информацию о работе сервера, которая может быть очень полезна для устранения всевозможных проблем.

- **Как изменить месторасположение каталога данных по умолчанию или отдельных баз данных.** Знание этих вопросов важно для управления размещением дискового пространства системы. Это может потребоваться для распределения данных по физическим дискам или перемещения данных в файловых системах с целью освобождения пространства на одном из дисков. Данная информация пригодится и при планировании размещения новых баз данных.

Для UNIX-систем в этой главе предполагается существование регистрационной учетной записи, которая используется для выполнения задач администрирования СУБД MySQL и для работы сервера. В этой книге в качестве имен пользователей и групп используются имена `mysqladm` и `mysqlgrp`. Причина использования указанной регистрационной записи для администрирования СУБД MySQL описана в главе 11, "Общее администрирование MySQL".

Размещение каталога данных

По умолчанию местоположение для каталога данных устанавливается при компиляции сервера. Обычно при инсталляции из исходного дистрибутива устанавливается каталог `/usr/local/var`, при инсталляции из двоичного дистрибутива – `/usr/local/mysql/data`, а при инсталляции из RPM-файла – `/var/lib/mysql`. В ОС Windows по умолчанию в качестве каталога данных принимается каталог `C:\mysql\data`.

Размещение каталога данных можно задать и явным образом при запуске сервера. Для этих целей применяется параметр `--datadir=` *`dir_name`*. Он оказывается весьма кстати, если каталог данных необходимо разместить в необычном месте, отличном от того, которое указывается по умолчанию.

Другим способом задания размещения файлов является их перечень в конфигурационном файле, который считывает сервер в момент запуска. В этом случае его не надо задавать в командной строке при каждом запуске сервера. Процедура перемещения каталога данных будет описана ниже.

Администратор MySQL обязательно должен знать, где находится каталог данных. При запуске нескольких серверов следует записать местоположение всех каталогов данных. Если же размещение каталога неизвестно (например, из-за того, что предыдущий администратор плохо вел свои записи), его можно определить несколькими способами.

- Для получения пути к каталогу данных непосредственно с сервера можно воспользоваться командой `mysqladmin variables` или оператором `SHOW VARIABLES`. На компьютере, работающем под управлением ОС UNIX, результат ее работы будет выглядеть примерно так:

```
% mysqladmin variables
+-----------------------+-------------------------+
| Variable_name         | Value                   |
+-----------------------+-------------------------+
...
| datadir               | /usr/local/var/         |
...
```

На компьютере, работающем под управлением ОС Windows, результаты ввода этой же команды будут выглядеть следующим образом.

```
% mysqladmin variables
+-----------------------+-------------------------+
| Variable_name         | Value                   |
+-----------------------+-------------------------+
...
| datadir               | c:\mysql\data\          |
...
```

Из mysql значение переменной можно проверить с помощью следующей команды:

```
mysql> SHOW VARIABLES LIKE 'datadir';
+---------------+-----------------------+
| Variable_name | Value                 |
+---------------+-----------------------+
| datadir       | /usr/local/mysql/var/ |
+---------------+-----------------------+
```

Если на компьютере запущено сразу несколько серверов, каждый из них использует свой порт TCP/IP и разъем. Чтобы получить информацию о каталоге данных от каждого сервера, достаточно подключиться с помощью параметров - -port и - -socket к соответствующему порту и разъему. Задание узла 127.0.0.1 явным образом сообщает утилите mysqladmin о необходимости подключиться к серверу на локальном узле, пользуясь соединением TCP/IP:

```
% mysqladmin --host=127.0.0.1 --port=port_num variables
```

В ОС UNIX определение значения localhost вызывает соединение с использованием разъема UNIX. Кроме того, для отображения пути к файлу разъема можно воспользоваться параметром - -socket:

```
% mysqladmin --host=localhost --socket=/path/to/socket variables
```

В ОС класса Windows NT соединение через именованный канал может задаваться при указании в качестве имени узла '.' и необязательного параметра - -socket для указания имени канала:

```
C:\> mysqladmin --host=. --socket=pipe_name variables
```

Команду mysqladmin можно запускать на любом компьютере, подключенном к серверу. Для подключения к серверу с удаленного компьютера применяется параметр - -host=*host_name*.

```
% mysqladmin --host=host_name variables
```

Кроме того, при необходимости подключаться через порт, отличный от стандартного, можно воспользоваться параметром - -port.

- Используйте команду ps для вывода командной строки исполняемого процесса mysql. Попробуйте воспользоваться одной из указанных ниже команд (в зависимости от версии команды ps, поддерживаемой системой) и поищите переменную - -datadir в выводимых результатах. В ОС типа BSD это можно попробовать сделать с помощью следующей команды:

```
% ps axww | grep mysql
```

А в ОС типа System V воспользуйтесь следующей командой:

```
% ps -ef | grep mysql
```

- Команда `ps` особенно полезна при запуске сразу нескольких серверов на одном компьютере, поскольку позволяет узнать месторасположение всех каталогов данных. Недостаток этого метода заключается в том, что команду `ps` обязательно нужно запускать на главном компьютере. Кроме того, она будет бесполезна, если переменная - `-datadir` не описана явным образом в командной строке `mysql`. (С другой стороны, некоторые сценарии запуска, вызывающие `mysqld`, пытаются определить имя пути к каталогу данных и поместить его в командную строку команды `mysqld`, которая делает эту информацию доступной для команды `ps`.)
- Просмотреть конфигурационный файл, который считывает сервер при запуске. Например, просмотрев файл `/etc/my.cnf` под управлением ОС UNIX или `C:\my.cnf` под управлением ОС Windows, можно найти строку `datadir` в конфигурационной группе `[mysqld]`:

  ```
  [mysqld]
  datadir=/path/to/data/directory
  ```

  Имя пути отражает расположение каталога данных.
- Справка сервера включает отображение месторасположения стандартного каталога данных. Это даст вам информацию о каталоге, который действительно использует сервер в своей работе, если это размещение не будет проигнорировано при старте. Для этого достаточно задать следующую команду:

  ```
  % mysqld -help
  ...
  datadir /usr/local/mysql/var/
  ...
  ```

- Если MySQL инсталлировалась из исходного дистрибутива, месторасположение каталога данных можно получить из информации о конфигурации. Так, например, месторасположение каталога указывается в элементе верхнего уровня `Makefile`. Однако будьте осторожны, поскольку позиция каталога является в `Makefile` значением переменной `localstatedir`, а не `datadir`, как можно ожидать. Кроме того, если дистрибутив размещается на смонтированной сетевой файловой системе NFS и используется для установки MySQL на несколько компьютеров, в информации конфигурации отражаются данные только для компьютера, на котором система устанавливалась последней. Вполне возможно, что им окажется не тот компьютер, для которого необходимы данные.
- Если все предыдущие методы вам не подходят, можно воспользоваться командой `find` для поиска файлов базы данных. Приведенная ниже команда ищет все файлы `.frm` (описания), являющиеся частью инсталляций MySQL:

  ```
  % find / -name "*.frm" -print
  ```

  Файлы `.frm` хранят определения таблиц, обслуживаемых сервером, таким образом, они являются частью любой инсталляции СУБД MySQL. Эти файлы можно найти в каталогах, которые имеют общий родительский корневой каталог, который по логике вещей и должен быть каталогом данных.

Во всех примерах этой главы в качестве каталога данных MySQL определен каталог *DATADIR*. Вполне возможно, что на других компьютерах для этих целей может применяться другой каталог.

Структура каталога данных

Каталог данных MySQL содержит все управляемые сервером базы данных и таблицы. Они организованы в структуру простого дерева, что позволяет воспользоваться преимуществами иерархической структуры файловых систем ОС UNIX и Windows.

- Каждой базе данных соответствует подкаталог, расположенный внутри каталога данных.
- Таблицам базы данных соответствуют файлы, размещенные внутри каталога базы данных.

Исключением из этого правила иерархического хранения баз данных и таблиц в виде каталогов и файлов является то, что все таблицы InnoDB из всех баз данных хранятся в одном общем табличном пространстве. Это табличное пространство создается с использованием одного файла или группы файлов, которые рассматриваются как единая унифицированная структура данных. В этих файлах хранятся таблицы и индексы. Файлы табличного пространства InnoDB по умолчанию хранятся в каталоге данных.

Каталог данных также может содержать и другие файлы.

- Конфигурационный файл `my.cnf`.
- PID-файл. В момент запуска сервер записывает свой идентификатор в этот файл. Это делается для того, чтобы другие программы могли определить значение, если им необходимо посылать ему сигналы. (Это не справедливо для случая с ОС Windows или встроенного сервера.)
- Файлы состояния и регистрации генерируются сервером. Данные файлы располагают важной информацией о работе сервера и имеют смысл для администраторов, особенно для определения причины проблемы, когда что-то работает не так. Например, если какой-то запрос нарушил работу сервера, то вы сможете идентифицировать впоследствии такой запрос, изучив регистрационные файлы.
- Кроме того, в каталоге данных обычно хранятся такие файлы, как ключевые файлы DES, или SSL-сертификаты сервера и ключевые файлы.

Как обеспечивается доступ к данным сервера MySQL

При решении "клиент/сервер" каждый элемент внутри каталога данных находится под управлением MySQL-сервера `mysqld`. Клиентские программы никогда не обращаются к данным напрямую. Точку взаимодействия, с помощью которой осуществляется доступ к базам данным, обеспечивает сервер, работающий в качестве промежуточного звена между клиентскими программами и данными (рис. 10.1).

В процессе запуска сервер открывает регистрационные файлы (если таковые запрашиваются) и предоставляет сетевой интерфейс к каталогу данных, прослушивая сетевые соединения. Для получения доступа к данным клиентская про-

грамма устанавливает соединение с сервером и посылает SQL-запрос на выполнение определенных операций (например, создание таблицы, выборка из таблицы или обновление записей в таблице). Сервер выполняет все операции, а полученные результаты отправляет обратно клиенту. Сервер представляет собой мультипотоковый механизм, поэтому он может обслуживать несколько клиентских соединений одновременно. При работе с приложением, в котором используется встроенный сервер, задействуется несколько иная архитектура, так как в этом случае есть только один "клиент" – приложение, которое содержит сам сервер. В этом случае сервер прослушивает внутреннее соединение, а не работает с сетевыми интерфейсами. Тем не менее встроенная серверная часть приложения управляет доступом к каталогу данных, и по-прежнему требуется координировать активность запросов, поступающих через несколько соединений, если приложение откроет сразу несколько соединений с сервером.

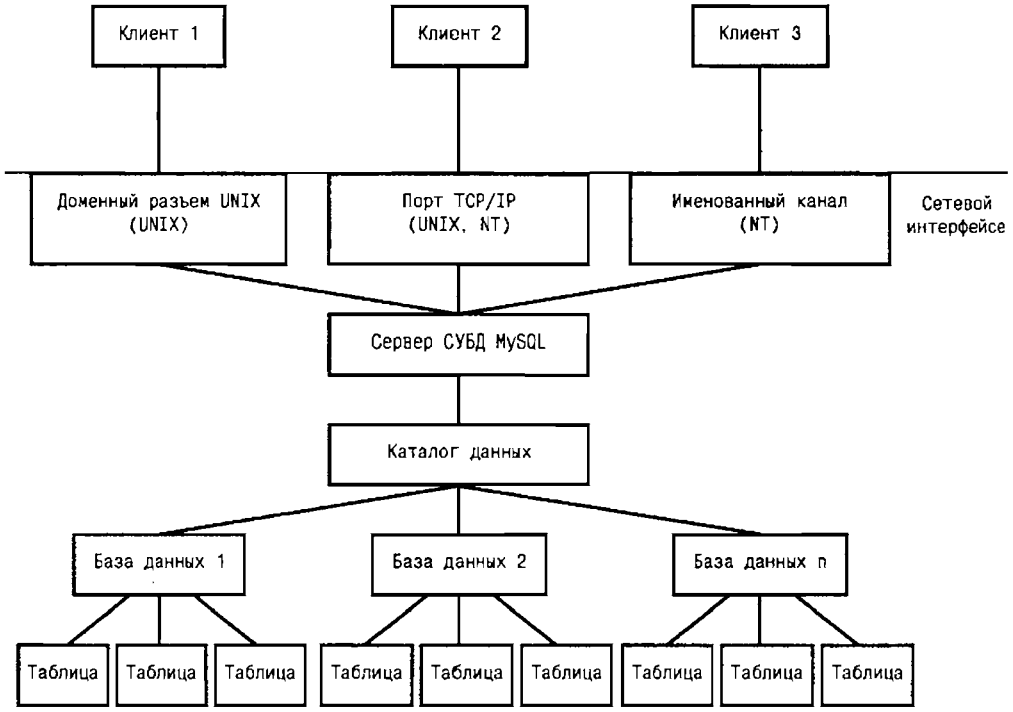


Рис. 10.1. Управление доступом к каталогу данных сервера MySQL

В обычных условиях использование сервера как единственной точки доступа к базе данных обеспечивает защиту от всякого рода недоразумений, связанных с одновременным доступом нескольких процессов к таблицам базы данных. Администраторы, однако, должны знать, что в некоторых случаях серверы не обладают эксклюзивным контролем над каталогом данных.

- **В одном каталоге данных запущено несколько серверов.** Как правило, для управления всеми базами данных используется один сервер, хотя можно запустить сразу несколько серверов. Если это делается для обеспечения дос-

тупа к нескольким отдельным каталогам данных, то проблем с взаимодействием различных серверов не возникает. Можно задать нескольким серверам один каталог данных (хотя удачным такое решение назвать сложно). В таком случае необходимо обязательно убедиться, что системы обеспечивают надежную блокировку файлов друг от друга. Иначе серверы начнут взаимодействовать некорректно. Кроме того, при одновременном запуске сразу нескольких серверов в одном каталоге регистрационные файлы будут содержать беспорядочные данные, а не ценную информацию.

- **Запущены утилиты восстановления таблиц.** Эти утилиты используются для поддержки таблиц, устранения проблем и отладки баз данных, поскольку они обладают возможностью изменения содержимого таблиц и могут получать доступ к данным одновременно с сервером. Это может стать причиной повреждения данных. Администратор должен понимать, как ограничить подобный совместный доступ, чтобы снизить вероятность разрушения таблиц. Более детально об использовании этих утилит рассказывается в главе 13, "Резервное копирование и восстановление баз данных". (Другой альтернативой является использование операторов CHECK TABLE и REPAIR TABLE, ограничивающих взаимодействие через управление сервером при выполнении операций обслуживания таблиц.)

Представление баз данных в файловой системе

Под каждую базу данных MySQL отводится отдельный каталог. Он представлен в виде подкаталога каталога данных и имеет такое же название, как и сама база. Так, например, базе данных mydb будет соответствовать каталог базы данных *DATADIR*/my_db. Такое представление значительно упрощает понимание предназначения и принципов работы некоторых операторов обработки баз данных.

Команда SHOW DATABASES на самом деле выводит список названий подкаталогов каталога данных. Некоторые системы управления базами данных поддерживают специальную таблицу со списком всех баз данных. В MySQL такой таблицы нет. Благодаря простоте структуры список баз является списком подкаталогов каталога данных. Следовательно, и нет необходимости в подобной таблице.

Так, оператор CREATE DATABASE *db_name* создает пустой подкаталог *db_name* в каталоге данных, устанавливая привилегии и режим доступа, которые обеспечивают доступ только для пользователя сервера MySQL (UNIX-пользователя, работающего на сервере). Аналогичные результаты можно получить и вручную, введя следующие команды:

```
% cd DATADIR
% mkdir DATADIR/db_name
% chmod u=rwx,go-rwx db_name
```

Такой подход, заключающийся в создании новой базы данных посредством создания пустого каталога, противоречит принципам, принятым в других СУБД, которые создают большое количество управляющих и системных файлов даже для "пустой" базы данных.

Также легко можно реализовать и команду DROP DATABASE. Команда DROP DATABASE *db_name* удаляет из каталога данных подкаталог *db_name* вместе со всеми

расположенными в нем файлами. Тех же результатов можно достичь в ОС UNIX с помощью следующих команд:

```
% cd DATADIR
% rm -rf DATADIR/db_name
```

или таких команд в ОС Windows:

```
C:\> cd DATADIR
C:\> del /S db_name
```

Разница между оператором DROP DATABASE и командами оболочки заключается в следующем.

- По команде DROP DATABASE сервер удалит лишь файлы, расширение которых отвечает табличным файлам. Если же в каталоге базы данных были созданы также какие-либо другие файлы, они останутся нетронутыми и сам каталог удален не будет. (Единственное неудобство будет заключаться в том, что имя будет продолжать отображаться оператором SHOW DATABASES.)
- Таблица InnoDB и содержимое индекса обслуживаются в табличном пространстве InnoDB, а не как каталог базы данных. Если база данных содержит таблицы типа InnoDB, необходимо пользоваться только оператором DROP DATABASE. Таким образом, дескриптор InnoDB удаляет таблицы из табличного пространства. *Ни в коем случае не удаляйте* каталог базы данных с помощью команды rm или del.

Представление таблиц баз данных

СУБД MySQL поддерживает работу с различными типами таблиц баз данных: ISAM, MyISAM, MERGE, BDB, InnoDB и HEAP. Любая таблица в MySQL представляется на диске как минимум одним файлом, а именно файлом с расширением .frm, который содержит описание структуры таблиц. Для большинства типов таблиц имеются еще и другие файлы, которые содержат данные и индексную информацию. Как будет видно из последующего материала данной главы, это зависит от типа таблицы. (Здесь все внимание фокусируется в первую очередь на характеристиках хранения различных типов данных на диске. Материал, посвященный особенностям различных типов и их поведению, содержится в главе 3, "Синтаксис и использование языка SQL".)

Таблицы ISAM

Первоначальным типом таблиц в СУБД MySQL был тип ISAM. MySQL представляет каждую таблицу в каталоге базы данных в виде трех отдельных файлов: файла формы (описания), файла данных и файла индексов. Все эти файлы имеют имя, совпадающее с именем таблицы, и расширение, которое отражает назначение файла. Например, таблица, имеющая имя mytbl, представляется тремя файлами:

- mytbl.frm — файл описания, в котором хранится формат (структура) таблицы;
- mytbl.ISD — файл данных ISAM, в котором хранится содержимое таблиц;
- mytbl.ISM — содержит индексную информацию всех индексов, которые имеет данная таблица.

Таблицы MyISAM

В версии MySQL 3.23 появились таблицы нового типа. Это был тип MyISAM, который является преемником типа ISAM, в настоящее время считающегося устаревшим. Подобно дескриптору ISAM, дескриптор MyISAM представляет таблицы в виде трех файлов, `.frm`, `.MYD` и `.MYI`, для файлов описания, данных и индексных данных соответственно.

Таблицы MERGE

Таблица типа MERGE представляет собой логическую конструкцию. Это набор таблиц MyISAM с идентичной структурой, которые при запросах рассматриваются как одна большая таблица. В каталоге базы данных таблица MERGE представлена файлом `.frm` и файлом `.MRG`, который является не чем иным, как списком имен таблиц, составляющих таблицы MyISAM, по одному имени в строке.

Таблицы BDB

Дескриптор таблиц типа BDB представляет каждую таблицу в виде двух файлов: `.frm` – файла описания и `.db` – файла, содержащего данные таблицы и индексную информацию.

Таблицы InnoDB

Все вышеперечисленные типы таблиц представлялись с помощью файлов, которые связаны уникальным образом с одной таблицей. Таблицы типа InnoDB поддерживаются несколько иным способом. Единственный файл, имеющий непосредственное отношение к таблице в формате InnoDB, – это файл описания `.frm`, который размещается в каталоге, к которому принадлежит таблица. Данные и индексы для всех таблиц InnoDB хранятся вместе в пределах одного унифицированного табличного пространства. Обычно табличное пространство представляет собой один или более больших файлов в каталоге данных. Эти компоненты табличного пространства формируют логически непрерывное пространство хранения, объем которого равен сумме объемов отдельных файлов.

Таблицы HEAP

Таблицы типа HEAP хранятся непосредственно в оперативной памяти. Так как сервер хранит данные и индексы в памяти, а не на диске, то эта таблица не представлена в файловой системе.

Взаимодействие операторов SQL с табличными файлами

Для хранения описания таблиц все типы таблиц используют файлы `.frm`. Таким образом, вывод оператора `SHOW TABLES` *`db_name`* должен совпадать с распечаткой основных имен файлов `.frm` для каталога базы данных *`db_name`*. Некоторые из систем баз данных ведут реестр, в котором хранятся все таблицы, имеющиеся в базе данных. СУБД MySQL этого не делает, так как это не обязательно. Такой реестр заложен в самой структуре каталога данных.

При выполнении оператора CREATE TABLE *tbl_name*, определяющего структуру таблицы, сервер создает файл *tbl_name*.frm с внутренней кодировкой структуры. Кроме того, создаются также файлы данных и индексов с информацией об отсутствии записей и индексов. Например, он создает файл данных и индексный файл .MYD и .MYI для таблицы типа MyISAM или объединенный файл данных/индексов .db для таблицы типа BDB. В случае таблиц InnoDB дескриптор инициализирует информацию о данных и индексах внутри табличного пространства InnoDB. В ОС UNIX права пользователей и их доступ для всех файлов, созданных для предоставления таблиц, устанавливаются такими, чтобы доступ был разрешен только той учетной записи, под управлением которой работает сервер.

Оператор ALTER TABLE расшифровывает файл *tbl_name*.frm и изменяет файлы данных и индексов с учетом определенных оператором структурных изменений. Такие же операции выполняются и операторами CREATE INDEX и DROP INDEX, поскольку они рассматриваются сервером как эквивалентные оператору ALTER TABLE. Изменение таблицы InnoDB приводит к изменению табличных данных и индексов внутри табличного пространства InnoDB. В процессе выполнения оператора DROP TABLE из каталога базы данных удаляются все три файла, представляющих таблицу. Удаление таблицы InnoDB также вызывает пометку любого пространства, связанного с удаляемой таблицей в табличном пространстве InnoDB, как свободного.

В некоторых типах таблиц пользователь не может вручную создать или изменить таблицу, хотя имеется возможность удалить ее. Для чего достаточно удалить три соответствующих файла. Так, например, эквивалентом оператора DROP TABLE mytbl для текущей базы данных mydb типа ISAM, MyISAM, BDB или MERGE могут быть команды ОС UNIX:

```
% cd DATADIR
% rm -f my_db/my_tbl.*
```

или такие команды ОС Windows:

```
C:\> cd DATADIR
C:\> del mydb\mytbl.*
```

Для таблиц типа InnoDB или HEAP части таблицы не представляются в файловой системе отдельными файлами, таким образом, DROP TABLE не имеет эквивалента в виде команды ОС. Например, файл .frm не представляет собой файла, уникальным образом связанного с таблицей InnoDB. Удаление этого файла приведет к потере табличных данных и индексов внутри табличного пространства InnoDB.

Ограничения операционной системы на имена баз данных и таблиц

В MySQL устанавливается несколько основных правил присвоения имен базам данных и таблицам.

- Имена могут включать буквы и цифры текущего набора символов, а также символы подчеркивания и доллара ("_" и "$").
- Длина имени не может превышать 64 символа.

- Начиная с MySQL 3.23.6 в имени могут использоваться и другие символы, при этом они должны заключаться в обратные одинарные кавычки (например, `odd@name`). Кроме того, в кавычки обязательно должны заключаться зарезервированные слова, если они используются в качестве имен столбцов.

Поскольку именам баз данных и таблиц соответствуют названия каталогов и файлов, операционная система, под управлением которой работает сервер, может накладывать дополнительные ограничения.

- В именах баз данных и таблиц можно использовать только разрешенные для имен файлов символы. Это справедливо для таблиц любого типа, так как любой тип представлен в файловой системе как минимум файлом `.frm`. Так, например, символ "$" разрешается правилами MySQL, однако в некоторых операционных системах его нельзя применять в именах файлов. Такого рода ограничения отсутствуют в ОС UNIX и Windows. Наиболее значительные проблемы могут возникнуть в момент ссылки на имена баз данных при выполнении задач администрирования непосредственно из оболочки. Предположим, например, что база данных имеет имя `$my_db`. В этом случае всякая ссылка на имя базы может интерпретироваться как ссылка на переменную:

```
% ls $mydb
mydb: Undefined variable.
```

  Чтобы избежать возможных недоразумений, необходимо либо избегать использования символа "$", либо использовать кавычки, подтверждая его специальное значение:

```
% ls my_db
% ls '$my_db'
```

  Необходимо использовать одинарные кавычки, поскольку двойные кавычки не отражают специального значения символов имени.

- Имя базы данных или таблицы не может содержать разделитель имени пути, даже если оно взято в кавычки. Например, в ОС UNIX и Windows составляющие имени пути разделяются символами "/" и "\" соответственно, и ни один из этих символов нельзя использовать. Причина того, что оба символа запрещены независимо от платформы, заключается в том, чтобы упростить процедуру перемещения баз данных с одной платформы на другую. (Например, допустив использование символа "/" в имени таблицы в ОС Windows, вы уже не сможете перенести ее в среду ОС UNIX.)
- Несмотря на то что MySQL разрешает задавать для баз данных и таблиц имена длиной до 64 символов, на самом деле их длина ограничивается максимально возможной длиной имени файла. В большинстве случаев эта проблема как таковая отсутствует, хотя в некоторых UNIX-системах System V все еще существует старое ограничение в 14 символов. В таком случае имя таблицы должно содержать не больше 10 символов, поскольку четыре остальные позиции отводятся под точку и трехсимвольное расширение.
- На присвоение имен базам данных и таблицам оказывает влияние также чувствительность используемой файловой системы к регистру символов.

Если буквы нижнего и верхнего регистров операционной системой воспринимаются по-разному (как, например, в ОС UNIX), имена `my_tbl` и `MY_TBL` будут принадлежать разным таблицам. Если же регистр не имеет значения (как, например, в Windows или в файловой системе HFS+ ОС Mac OS X), `my_tbl` и `MY_TBL` окажутся разными названиями одной и той же таблицы. Об этом следует помнить при разработке базы данных, которую впоследствии планируется перенести на другую платформу.

Одним из способов решения проблемы чувствительности к регистру является привычка всегда давать имена базам данных и таблицам в одном определенном регистре. Другой способ заключается в присвоении переменной сервера `lower_case_table_names` значения, равного 1, что имеет два последствия:

- сервер преобразует имена таблиц в нижний регистр перед созданием соответствующих файлов на диске;
- при обращении к таблице в различных запросах сервер преобразует имя в нижний регистр перед тем, как сделать попытку обнаружения таблицы на диске.

Результат этих действий заключается в том, что имена не чувствительны к регистру. Однако имеют место два нюанса, которые необходимо иметь в виду при использовании переменной `lower_case_table_names`. Во-первых, это не касается имен баз данных до версии MySQL 4.0.2; до этого это имело отношение только к именам таблиц. Во-вторых, эту переменную необходимо включить перед тем, как приступать к созданию баз данных или таблиц, а не после этого. В противном случае желаемого эффекта вы не достигнете потому, что на диске будут храниться файлы, которые представлены только в нижнем регистре. Для того чтобы решить эту проблему, переименуйте все базы данных или таблицы, чтобы они содержали только символы в нижнем регистре, а потом установите переменную.

Факторы, влияющие на максимальный размер таблиц

Размеры таблиц в MySQL имеют определенные границы, которые определяются совокупностью факторов. Таким образом, совсем не просто точно определить размеры этих границ. Факторы, влияющие на размеры границ, таковы.

- MySQL имеет свои собственные внутренние ограничения на размеры таблиц, и они варьируются в зависимости от типа таблицы.
  - Для таблиц ISAM размер файлов `.ISD` и `.ISM` ограничивается 4 Гбайт.
  - Для таблиц MyISAM файлы типа `.MYD` и `.MYI` также по умолчанию ограничены размером 4 Гбайт. Однако использование параметров `AVG_ROW_LENGTH` и `MAX_ROWS` позволяет создать таблицу размером порядка 8 миллионов терабайт. (Оператор `CREATE TABLE` описан в приложении Г, "Синтаксис SQL".)
  - Максимальный размер таблицы MERGE является функцией совокупности максимальных размеров, составляющих таблиц MyISAM.
  - Размеры таблиц BDB ограничены допустимым размером файла `.db`. Он варьируется в соответствии с размером страницы таблицы (который зада-

ется во время создания сервера), но даже при минимальном размере страницы (512 байт) максимальный размер файла будет достигать 2 Тбайт.

- Для таблиц InnoDB максимальный размер табличного пространства составляет 4 миллиарда страниц; размер страницы по умолчанию равен 16 Кбайт. (СУБД MySQL может быть перекомпилирована для использования страницы InnoDB размером от 8 до 64 Кбайт.) Максимальный размер табличного пространства также привязан к размеру отдельной таблицы InnoDB.

- Операционная система накладывает ограничение на максимальный размер файла. В целом тенденция для операционных систем заключалась в постепенном росте ограничений на размеры файлов, но предел порядка 2 Гбайт остается повсеместно. Этот предел размера относится к файлам, которые используются для представления файлов `.MYD` и `.MYI` для таблиц типа MyISAM. Это также справедливо и в отношении файлов, составляющих табличное пространство таблиц типа InnoDB. Однако размер табличного пространства InnoDB может достаточно легко превысить максимальный размер файла, для этого достаточно создать табличное пространство, состоящее из нескольких файлов, каждый из которых имеет максимальный размер. Другим приемом, позволяющим обойти ограничение на размер файла, является использование разделов в табличном пространстве типа InnoDB, которые можно создавать начиная с MySQL 3.23.41. Компоненты табличного пространства, базирующиеся на разделах, могут иметь такую же длину, как и сами разделы.

- Для типов таблиц, в которых данные и индексы размещаются в раздельных файлах (таких как ISAM и MyISAM), предел размера таблицы будет достигнут, когда любой из отдельных файлов достигает своего собственного предела размера файла. Характеристики индексирования таблиц влияют на то, какой это будет файл. Для таблиц, не имеющих индексов или имеющих их небольшое количество, вероятнее всего ситуация, когда файл данных первым достигнет предельного размера. Для сильно индексированных таблиц индексный файл может первым достичь предельного размера.

- Присутствие столбца `AUTO_INCREMENT` неявно ограничивает количество столбцов, которые может иметь таблица. Например, если этот столбец имеет тип `TINYINT UNSIGNED`, максимальное значение, которое может хранить этот тип данных, равно 255. Таким образом, эта цифра и станет предельным числом строк таблицы, которые она может хранить. Большие целые типы позволяют хранить и большее число строк. Более обобщенно можно сказать, что любой уникальный индекс таблицы ограничивает его количество строк максимальным количеством уникальных значений индекса.

Для того чтобы определить реальный предельный размер таблицы, необходимо принимать в расчет все факторы. Эффективный максимальный размер таблицы определяется минимальным из этих факторов. Предположим, что нам потребовалось создать таблицу типа ISAM. MySQL позволяет, чтобы максимальный размер файла данных и файла индексов был равен 4 Гбайт. Но если используемая вами операционная система накладывает ограничение на максимальный размер файлов в 2 Гбайт, это ограничение становится предельным размером табличных

файлов. С другой стороны, если ваша система имеет поддержку больших файлов, размеры файлов могут превышать 4 Гбайт, тогда определяющим фактором становится внутреннее ограничение MySQL в 4 Гбайт.

Что касается таблиц типа InnoDB, то необходимо постоянно принимать во внимание, что они должны умещаться в табличном пространстве InnoDB. При наличии одной таблицы типа InnoDB она может иметь такой же размер, как и все табличное пространство. Но если, что более вероятно, имеется много таблиц, они все разделяют одно и то же табличное пространство и, таким образом, все ограничиваются в размере не только размером табличного пространства, но и количеством таблиц, которые размещены в данном табличном пространстве. Каждая отдельная таблица типа InnoDB имеет возможность расширения до тех пор, пока табличное пространство не заполнится полностью. С другой стороны, после заполнения табличного пространства ни одна таблица InnoDB не может больше увеличиваться в размере до тех пор, пока к табличному пространству не будет добавлен еще один компонент. (Начиная с MySQL 3.23.50 последний компонент табличного пространства можно сделать автоматически расширяемым, чтобы он мог автоматически увеличиваться в размере до тех пор, пока его размер не достигнет ограничения по размеру файла и имеется свободное дисковое пространство. Подробно о конфигурации табличного пространства рассказывается в главе 11, "Общее администрирование MySQL".)

Влияние структуры каталога данных на производительность системы

Построение структуры каталогов данных на основании иерархической структуры файловой системы делает структуру каталога данных понятной для пользователей. В то же время эта структура оказывает определенное влияние на производительность системы. Особенно это справедливо в отношении операций открытия файлов таблиц баз данных.

Поскольку в структуре каталога данных таблица представляется несколькими файлами, для открытия каждой таблицы требуется не один, а несколько дескрипторов файлов. Сервер весьма эффективно кэширует дескрипторы, однако загруженному серверу для обслуживания множества одновременных клиентских соединений и обработки сложных запросов к нескольким таблицам потребуется большое количество дескрипторов. Дескрипторы файлов — весьма ограниченный ресурс во многих системах, особенно в тех из них, в которых по умолчанию установлен низкий лимит.

Еще один негативный момент, связанный с представлением одной таблицы в виде нескольких файлов, заключается в том, что с увеличением таблиц увеличивается и время их открытия. Операции открытия таблиц целиком и полностью базируются на системных операциях открытия файлов, а следовательно, зависят от эффективности работы механизмов поиска файлов операционной системы. Как правило, эта проблема несущественна, хотя о ней все же стоит помнить при создании базы данных с большим количеством таблиц.

Так, например, каталог базы данных типа MyISAM, включающей 10000 таблиц, содержит 30000 файлов. При открытии большого количества таблиц замедление

выполнения операций открытия становится достаточно заметным. (Особенно это относится к файловым системам операционных систем Linux ext2 и Solaris.) Если же эта проблема приобретает действительно угрожающие масштабы, возможно, имеет смысл рассмотреть возможность использования файловой системы такого типа, которая бы работала высокоэффективно с большим количеством файлов. Например, файловая система ReiserFS имеет хорошую производительность даже при большом количестве маленьких файлов. Если это невозможно, следует пересмотреть структуру своих таблиц в соответствии со спецификой работы приложений и соответствующим образом их реорганизовать. Тщательно обдумайте, действительно ли необходимо такое большое количество таблиц. Иногда приложения генерируют их безо всякой на то необходимости. Много таблиц с аналогичными структурами могут генерироваться приложениями, которые создают отдельную таблицу для каждого пользователя. Чтобы объединить такие таблицы в одну, достаточно добавить столбец, который идентифицирует пользователя-владельца строки. Если в результате таких действий число таблиц значительно уменьшится, производительность приложения возрастет.

Всегда при проектировании базы данных следует анализировать, подходит ли для данного приложения та или иная стратегия. Однако существуют также причины, не позволяющие объединять таблицы описанным выше способом. Основными среди них являются следующие.

- **Ограничение дискового пространства.** Объединение таблиц приводит к уменьшению их числа (уменьшая, в свою очередь, время открытия), но сопровождается добавлением дополнительных столбцов (занимая дополнительное дисковое пространство). В результате имеет место достаточно распространенный компромисс между временем обработки и свободным пространством, и администратор должен решить, какой фактор для него более важен. Если на первом плане стоит скорость, возможно, придется пожертвовать дополнительным дисковым пространством. Если же объемы дисков сильно ограничены, придется использовать большее количество таблиц и немного дольше ждать при открытии.
- **Вопросы безопасности.** Эта причина также может в значительной степени ограничить возможности и желание объединить таблицы. Основная цель использования отдельных таблиц для каждого пользователя — открытие доступа к данным таблицы только пользователям, обладающим соответствующими привилегиями. На самом деле права пользователей определяются полномочиями на уровне таблицы. После объединения данные всех пользователей окажутся в одной таблице.

Есть еще один способ создания множества таблиц без необходимости создания множества отдельных файлов — это использование таблиц типа InnoDB. Дескриптор таблиц InnoDB уникальным образом ассоциирует таблицы только с файлом типа `.frm` и хранит данные и индексы для всех таблиц в одном табличном пространстве InnoDB. Это уменьшает количество необходимых для отображения файлов на диске и постоянно уменьшает количество дескрипторов файлов, необходимых для открытия таблиц. Для таблиц типа InnoDB требуются только по одному дескриптору на один составляющий табличное пространство файл (это по-

стоянная величина на все время продолжительности жизни серверного процесса) и по дескриптору на любую таблицу, который он открывает при чтении файла `.frm` таблицы.

Файлы состояния и регистрационные файлы MySQL

Помимо подкаталогов баз данных, каталог данных MySQL содержит множество файлов состояния и регистрационных файлов. Краткий обзор этих файлов представлен в табл. 10.1, а детальное их описание следует далее. По умолчанию основная часть имени этих файлов содержит имя главного компьютера. В таблице это имя заменено словом *HOSTNAME*.

Таблица 10.1. Файлы состояния и регистрационные файлы MySQL

Тип файла	Имя по умолчанию	Содержимое файла
Идентификатор процесса	*HOSTNAME*.pid	Идентификатор процесса сервера
Журнал ошибок	*HOSTNAME*.err	События запуска и завершения работы; записи об ошибках
Общий журнал запросов	*HOSTNAME*.log	События подключения/отключения и информация о запросах
Журнал медленных запросов	*HOSTNAME*-slow.log	Текст запросов, требующих длительной обработки
Журнал обновлений	*HOSTNAME*.nnn	Текст всех запросов, изменяющих содержимое или структуру таблицы
Двоичный журнал обновлений	*HOSTNAME*-bin.log	Представление запросов, обновляющих данные в двоичном формате
Индексный файл двоичного журнала	*HOSTNAME*-bin.index	Перечень существующих журналов двоичных обновлений

PID-файлы

Сервер записывает идентификатор своего процесса (Process ID – PID) в PID-файл при запуске и удаляет этот файл по завершении процесса. Именно с помощью PID-файла сервер позволяет находить себя работающим процессам. Так, например, если во время завершения работы системы запустить сценарий `mysql.server` для завершения работы и сервера MySQL, данный сценарий обратится к PID-файлу. Это обращение позволит определить, какому процессу отправить команду на завершение работы.

Регистрационные файлы СУБД MySQL

СУБД MySQL может вести большое число различных регистрационных файлов. Большая часть регистрируемой информации не обязательна; таким образом, с помощью параметров при запуске сервера можно активизировать только те регистрационные журналы, которые вам необходимы, задав при этом их имена, если вас не удовлетворяют те, которые принимаются по умолчанию. В этом разделе кратко описываются регистрационные журналы. Подробнее о регистрационных журналах и параметрах, позволяющих контролировать поведение при регистрации сервера, речь пойдет в главе 11"Общее администрирование MySQL".

Общий журнал запросов предоставляет информацию о функционировании сервера: кто и с какого компьютера подключился и какие запросы присылает. Журнал обновлений также содержит информацию о запросах, однако только о тех из них, которые связаны с изменением содержимого баз данных. Содержимое же журнала обновлений представляется в виде операторов SQL. Впоследствии их можно будет выполнить, представив в виде ввода для клиента `mysql`. Двоичный журнал обновления аналогичен журналу обновления, но представляется в более эффективном двоичном формате. Сопутствующий ему индексный файл двоичного журнала содержит перечисление двоичных файлов обновления, которые в данный момент обрабатывает сервер.

Журналы обновлений оказываются особенно полезными при сбое, когда необходимо отследить все изменения, внесенные с момента последнего резервирования базы данных. Их использование позволяет восстановить состояние баз данных, в котором они находились перед самым сбоем. Эта процедура детально описана в главе 13, "Резервное копирование и восстановление баз данных". Двоичные журналы регистрации также используются для сервера репликации, так как они служат для протоколирования обновлений, которые должны быть переданы с главного сервера на подчиненный.

Ниже приведен пример информации, которая записывается в основной регистрационный журнал в результате короткого сеанса соединения клиента, который создает таблицу в базе данных `test`, вставляет в таблицу строку, а затем удаляет таблицу.

```
020727 15:00:17 1 Connect sampadm@localhost on test
2 Query CREATE TABLE mytbl (val INT)
2 Query INSERT INTO mytbl VALUES(1)
2 Query DROP TABLE mytbl
2 Quit
```

Общие регистрационные журналы хранят информацию о дате и времени, идентификаторе соединения сервера и специфической информации о событии. Если значения даты и времени в строке отсутствуют, это значит, что они не изменились и равны значениям, указанным в предыдущей строке. (Другими словами, в регистрационных журналах сервера регистрируются только те значения даты и времени, которые отличаются от предыдущих показаний.) Аналогичный сеанс записывается в регистрационном журнале обновления. Операторы содержат завершающие точки с запятой, что позволяет их рассматривать в качестве ввода в программу `mysql` для повторения при восстановлении данных.

```
use test;
CREATE TABLE my_tbl (val INT);
INSERT INTO my_tbl VALUES(1);
DROP TABLE my_tbl;
```

Для получения расширенной формы журнала обновлений используется параметр `- -log-long-format`. В расширенном журнале также предоставляется информация об отправителе и времени поступления запроса. Эти данные занимают, конечно, больше места на диске, однако позволяют узнать, кто и что пытался сделать. Сопоставлять информацию о событиях в общем журнале и журнале обновлений для этого не нужно. Для только что показанного сеанса расширенная регистрация обновления дает следующую информацию:

```
# Time: 020727 15:00:17
# User@Host: sampadm[sampadm] @ localhost []
use test;
CREATE TABLE mytbl (val INT);
# User@Host: sampadm[sampadm] @ localhost []
INSERT INTO mytbl VALUES(1);
# User@Host: sampadm[sampadm] @ localhost []
DROP TABLE mytbl;
```

Дополнительная информация записывается с использованием строк, которые начинаются символом '#'. Таким образом, при посылке журнала на `mysql` для выполнения сервером они интерпретируются как комментарии.

Журнал регистрации ошибок содержит диагностическую информацию, которая создается сервером при возникновении ошибочных ситуаций. Этот журнал очень полезен в случае неуспешного запуска сервера или неожиданного выхода, так как при этом он обязательно будет содержать причину произошедшего сбоя. Регистрационные журналы могут разрастаться очень сильно, поэтому очень важно, чтобы их увеличение контролировалось и они не достигли размера вашей файловой системы. Периодически регистрационные журналы необходимо очищать, чтобы предотвратить переполнение файловой системы. Детальная информация по обслуживанию регистрационных журналов содержится в главе 11, "Общее администрирование MySQL".

Совсем не лишним будет обезопасить регистрационные журналы от чтения их случайными пользователями, так как они могут содержать тексты запросов, содержащих такую конфиденциальную информацию, как пароли. Например, следующая запись журнала содержит пароль пользователя `root`; совершенно естественно, что эта информация совсем не принадлежит к разряду такой, к которой может быть разрешен доступ всем:

```
020727 15:47:24 4 Query UPDATE user SET
Password=PASSWORD('secret')
WHERE user='root'
```

Регистрационные журналы по умолчанию записываются в каталог данных, поэтому вопрос безопасности регистрационных журналов прежде всего заключается в обеспечении безопасности каталога, содержащего их. Подробно эта процедура будет описана в главе 12, "Безопасность".

Перемещение содержимого каталога данных

В предыдущем разделе описывалась структура каталога данных, создаваемая сервером по умолчанию. Этот каталог содержит все рабочие базы данных и файлы состояния. Но иногда возникает необходимость в определении специального места для хранения содержимого каталога данных. В этом разделе рассказывается, зачем может потребоваться перемещать отдельные части каталога данных (и даже весь каталог целиком), какие его компоненты можно перемещать и как такое перемещение осуществить.

- Каталог данных можно разместить на диске большего размера, чем используется по умолчанию.
- Если каталог данных располагается на часто используемом диске, перемещение его на другой диск позволит выровнять нагрузку между физическими дис-

ками. В этом случае можно разместить файлы баз данных и журналов на отдельном диске или распределить их по нескольким дискам сразу. Это справедливо и в отношении табличного пространства InnoDB, которое концептуально является большим блоком памяти, но вы можете разместить его на различных устройствах, чтобы улучшить его рабочие характеристики.

- Размещение баз данных и регистрационных журналов на различных дисках позволяет минимизировать ущерб, который может быть вызван сбоем одного из дисков.
- Каждый из одновременно запущенных серверов может работать со своим каталогом данных. Такой подход является одним из способов обойти ограничения на файловые дескрипторы, особенно если эти ограничения нельзя устранить посредством настройки ядра системы.
- Некоторые операционные системы хранят PID-файлы сервера в отдельном каталоге, например `/var/run`. Возможно, для большей согласованности работы системы администратор пожелает разместить в этой папке и PID-файлы MySQL. Аналогично, если ваша система для хранения регистрационных журналов использует каталог `/var/log`, журналы СУБД MySQL можно поместить туда же. (Однако многие системы позволяют производить запись в эти каталоги только пользователю `root`. Это означает, что сервер должен работать с учетной записью `root`, что с точки зрения безопасности не очень хорошо.)

В оставшейся части этого раздела будет описано, какие части каталога данных можно переносить и как сохранить работоспособность после сделанных изменений.

Методы перемещения

Существуют два способа перемещения компонентов каталога данных.

- Определение параметров загрузки сервера с помощью командной строки или в группе `[mysqld]` конфигурационного файла. Например, если необходимо определить местоположение каталога данных, сервер можно запустить с параметром `--datadir=`*`dir_name`* при запуске из командной строки или задать в конфигурационном файле следующие строки:

```
[mysqld]
datadir=dir_name
```

  Как показано в этом примере, обычно группе конфигурационного файла присваивается имя `[mysqld]`. Однако в зависимости от конкретных условий больше подойдут другие имена. Например, группа `[embedded]` имеет отношение к встроенному серверу. Если работает сразу несколько серверов с использованием сценария `mysqld_multi`, имена групп будут иметь форму `[mysqld`*`n`*`]`, где *n* является целым числом, которое связано с определенным экземпляром сервера. В главе 11, "Общее администрирование MySQL", приводится информация о том, какая группа параметров имеет отношение к определенному методу запуска сервера, и содержатся инструкции для запуска нескольких серверов.

- Перемещение элементов и создание в исходном каталоге символической ссылки, указывающей на новое местоположение.

Ни один из приведенных выше методов не является универсальным для переноса информации. В табл. 10.2 отмечается, какие компоненты каталога данных можно перемещать и какой метод следует для этого использовать. Если применяется первый метод, можно задать параметры в глобальном конфигурационном файле (`/etc/my.cnf` – в UNIX, `C:\my.cnf` или `C:\my.ini` – на компьютерах, работающих под управлением ОС Windows). В последних версиях ОС Windows этот файл может располагаться в системной папке `C:\windows` или `C:\NT`.

Конфигурационный файл `my.cnf` также можно размещать в стандартном каталоге данных (каталог, созданный внутри сервера). Это удобно при использовании конфигурационного файла для поддержки специфических задач, выполняемых сервером, но так как сервер ищет их только в откомпилированном каталоге данных, файл не будет найден, если вы этот каталог переместите. (Обойти эту проблему можно следующим образом: перенести каталог данных и сделать их первоначальное местоположение ссылкой на новое местоположение.)

Таблица 10.2. Методы перемещения

Перемещаемый компонент	Метод перемещения
Целый каталог данных	Параметр запуска или символическая ссылка
Каталоги отдельных баз данных	Символическая ссылка
Отдельные таблицы баз данных	Символическая ссылка
Табличное пространство InnoDB	Параметр запуска
PID-файл	Параметр запуска
Файл общего журнала	Параметр запуска
Файл журнала обновлений	Параметр запуска

Прогнозирование эффекта от перемещения

Прежде чем перемещать какие-либо компоненты, настоятельно рекомендуется убедиться, что эта операция приведет к желаемому эффекту. Для получения информации о пространстве диска некоторые пользователи предпочитают использовать команды `du`, `df` и `ls -l`, хотя этот выбор, в первую очередь, определяется правильным пониманием структуры используемой файловой системы.

В приведенном ниже примере существует трудноразличимая ловушка, в которую можно попасться при перемещении каталога данных. Предположим, что каталог данных `/usr/local/var` планируется переместить в каталог `/var/mysql`, поскольку согласно выводу команды `df` файловая система `/var` содержит больше свободного пространства:

```
% df /usr /var
Filesystem       1K-blocks    Used     Avail Capacity   Mounted on
/dev/wd0s3e      396895       292126   73018    80%      /usr
/dev/wd0s3f      1189359      1111924  162287   15%      /var
```

Сколько же пространства освободится в файловой системе /usr в результате перемещения каталога данных? Чтобы вычислить этот объем, воспользуемся командой du -s и посмотрим, сколько этот каталог занимает:

```
% cd /usr/local/var
% du -s
133426
```

Как видно, этот каталог занимает чуть больше 130 Мбайт, которые можно освободить в /usr. Однако можно ли этот прием реализовать на самом деле? Запустите команду df в каталоге данных:

```
% df /usr/local/var
Filesystem        1K-blocks    Used    Avail    Capacity     Mounted on
/dev/wd0s3f       1189359      1111924 162287       15%      /var
```

Что же получается? При запросе объема свободного пространства в файловой системе, содержащей каталог /usr/local/var, команда df отображает свободный объем в /var. Почему так? Ответ на этот вопрос дает команда ls -l:

```
% ls -l /usr/local
...
lrwxrwxr -x 1 mysqladm mysqlgrp 10 Dec 11 23:46 data -> /var/mysql
...
```

Из результатов выполнения этой команды видно, что /usr/local/mysql/var является символической ссылкой на /var/mysql. Другими словами, каталог данных *уже* перемещен в файловую систему /var и включает указывающую на нее символическую ссылку. Следовательно, никакой выгоды перемещение каталога данных из /usr в /var не принесет.

Суть этого примера состоит в том, что несколько предварительных действий по определению эффекта перемещения могут показать нецелесообразность подобного перемещения. Такая предосторожность позволяет вовремя остановиться и не тратить уйму времени на перемещение только для того, чтобы затем понять, что достичь поставленной цели просто невозможно.

Меры предосторожности при перемещении

Перед выполнением операции перемещения необходимо завершить работу сервера, а затем запустить его снова. При перемещении некоторых компонентов (например, каталога базы данных) можно, хотя и не рекомендуется, оставить сервер в рабочем состоянии. В этом случае следует убедиться, что сервер не обращается к перемещаемой базе данных. Не забудьте также выполнить оператор FLUSH TABLES перед перемещением базы данных, чтобы сервер закрыл все открытые файлы таблиц. Игнорирование этих моментов может привести к повреждению таблиц.

Перемещение каталога данных

Для перемещения каталога данных необходимо завершить работу сервера и только после этого перенести каталог на новое место. Затем необходимо удалить данные исходного каталога и заменить его символической ссылкой, указывающей на новое место, либо перезапустить сервер с параметром --datadir, определяющим новое местоположение. Метод символической ссылки является предпочтительным, если каталог данных содержит файл my.cnf и сервер работает с его использованием.

Перемещение баз данных

Базы данных можно перемещать только с помощью метода символической ссылки. В ОС UNIX это можно сделать только следующим образом.

1. Завершить работу сервера.
2. Перенести или скопировать каталог базы данных на новое место.
3. Удалить этот каталог.
4. Заменить его символической ссылкой, указывающей на новое место.
5. Перезапустить сервер.

Так, например, перемещение базы данных `bigdb` на другое место выполняется с помощью следующих команд:

```
% mysqladmin -p -u root -p shutdown
Enter password: ********
% cd DATADIR
% tar cf - bigdb | (cd /var/db; tar xf -)
% mv bigdb bigdb.orig
% ln -s /var/db/bigdb .
% mysqld_safe &
```

Для выполнения всех этих команд необходимо зарегистрироваться в качестве администратора СУБД MySQL. Как видите, исходный каталог данных переименовывается для безопасности в `bigdb.orig`. После проверки правильности работы сервера с перемещенной базой данных его можно удалить:

```
% rm -rf bigdb.orig
```

На компьютере, работающем под управлением ОС Windows, перемещение базы данных осуществляется несколько по-другому.

1. Завершите работу сервера.
2. Перенесите каталог базы данных в необходимое место.
3. Создайте файл `.sym` в каталоге данных СУБД MySQL, который бы указывал на новое местоположение базы данных. Например, при переносе базы данных `sampdb` из `C:\mysql\data\sampdb` в `E:\mysql-book\sampdb` потребуется создать файл с именем `sampdb.sym` в каталоге `C:\mysql\data`, который содержит следующую строку:

   ```
   E:\mysql-book\sampdb\
   ```

   Файл `.sym` действует как символическая ссылка для того, чтобы проинформировать сервер MySQL о том, где можно найти перемещенный каталог базы данных.
4. При запуске сервера убедитесь в том, что поддержка символической ссылки активизирована. Это можно сделать из командной строки с помощью параметра `- -use-symbolic-links` или следующих строк в конфигурационном файле:

   ```
   [mysqld]
   use-symbolic-links
   ```

Для того чтобы после переноса базы данных в ОС Windows сервер работал правильно, необходимо запускать сервер типа -max (mysqld-max или mysqld-max-nt) начиная с MySQL 3.23.16 или выше.

При переносе базы данных в другую файловую систему для перераспределения памяти базы данных необходимо помнить, что содержимое таблиц типа InnoDB хранится в табличном пространстве InnoDB, а не в каталоге базы данных. Для базы данных, состоящей в основном из таблиц типа InnoDB, перенос каталога базы данных будет иметь незначительный эффект на распределение памяти.

Удаление перенесенных таблиц

Удалить базу данных можно с помощью оператора DROP DATABASE. На серверах СУБД MySQL до 3.23 версии удаление перемещенных баз данных является проблемой. Таблицы при этом удаляются корректно, но ошибка возникает, когда сервер делает попытку удалить каталог базы данных, поскольку этот каталог является символической ссылкой, а не реальным каталогом. При возникновении этой проблемы необходимо завершить операцию DROP DATABASE удалением каталога базы данных и указывающей на нее символической ссылки вручную.

Перемещение таблиц баз данных

Перемещение отдельных таблиц поддерживается только при определенных, строго ограниченных обстоятельствах:

- необходимо использовать СУБД MySQL 4.0 или более новую версию;
- операционная система должна работать с вызовами realpath();
- перемещаемая таблица должна иметь тип MyISAM.

При выполнении этих условий можно перенести на новое место файл данных .MYD и индексный файл .MYI, а после этого создать символические ссылки на них в каталоге базы данных с исходными именами файлов данных и индексов. (Файл .frm должен остаться не прежнем месте в каталоге базы данных.)

Нельзя переносить таблицу, если не выполнено хотя бы одно из вышеперечисленных условий. Если все же это было сделано, изменения можно отменить с помощью оператора ALTER TABLE, OPTIMIZE TABLE или REPAIR TABLE. В процессе выполнения каждого такого оператора в каталоге базы данных сначала создастся временная таблица, которая и поддается изменению или оптимизации. Сразу после этого исходная таблица удаляется, а ее имя присваивается временной таблице. В результате этой процедуры символические ссылки будут удалены, а новая измененная таблица окажется записанной в том же каталоге данных, откуда ранее была перемещена исходная таблица. В конечном итоге старые перемещенные ранее файлы таблиц оказываются на новом месте. В большинстве случаев пользователи о них забывают, что способствует неэффективному использованию дискового пространства. Кроме того, в процессе изменения удаляются символические связи, из-за чего впоследствии оказывается довольно трудно вспомнить, куда были перемещены файлы таблиц. Поскольку практически невозможно гарантировать, что обладающие соответствующими полномочиями пользователи не будут пытаться изменить или оптимизировать таблицы (таким образом, сводя на нет все усилия по их перемещению), лучше оставить файлы таблиц в каталоге базы данных.

Перенос табличного пространства InnoDB

Настройка табличного пространства InnoDB первоначально производится посредством перечисления мест размещения составляющих файлов в конфигурационном файле с помощью параметров `innodb_data_home_dir` и `innodb_data_file_path`. (Подробно процедура настройки табличного пространства приводится в главе 11, "Общее администрирование MySQL".) Можно переносить составные части уже созданного табличного пространства, например, для того, чтобы распределить их между различными файловыми системами. Так как вы перечислили места размещения файлов с помощью параметров при запуске сервера, способ перемещения отдельных или всех файлов табличного пространства будет следующим.

1. Завершите работу сервера.
2. Перенесите файл табличного пространства или другие файлы на нужное место.
3. Внесите в конфигурационный файл, в котором определяется конфигурация InnoDB, все новые местоположения файлов.
4. Перезапустите сервер.

Строго говоря, компонент табличного пространства можно переместить с последующим созданием символической ссылки на первоначальном месте. Но в этом нет смысла; в любом случае у вас имеется перечень компонентов в конфигурационном файле, поэтому вы сможете указать в перечне реальное размещение, а не символическую ссылку.

Перемещение файлов состояния

Для переноса PID-файла необходимо завершить работу сервера и запустить его повторно с соответствующим параметром, задающим новое размещение файла. Например, для создания PID-файла `/tmp/mysql.pid`, в командной строке можно воспользоваться параметром `- -pid-file=/tmp/mysql.pid` или включить в конфигурационный файл следующие строки:

```
[mysqld]
pid-file=/tmp/mysql.pid
```

При задании имени файла с абсолютным именем пути сервер создаст PID-файл с этим именем пути. В противном случае файл будет создан в каталоге данных. Например, параметр `- -pid-file=mysqld.pid` приводит к созданию PID-файла `mysqld.pid` в каталоге данных. Для того чтобы перенести регистрационные файлы, воспользуйтесь параметрами запуска сервера, описание которых вы найдете в главе 11, "Общее администрирование MySQL".

11

Общее администрирование MySQL

В этой главе...

Защита новой инсталляции MySQL
Настройка процедур запуска и завершения работы сервера MySQL
Управление пользовательскими учетными записями
Ведение файлов журналов
Дополнительные вопросы настройки сервера
Запуск нескольких серверов
Установка серверов репликации
Обновление MySQL

В данной главе рассматриваются задачи администратора MySQL, точное выполнение которых позволит обеспечить согласованную и эффективную работу сервера MySQL. К задачам подобного рода относится проверка работоспособности сервера, достижение максимально возможной производительности, настройка пользовательских учетных записей для обеспечения клиентского доступа к серверу, поддержка журналов и резервирование баз данных. В некоторых случаях, когда на одном компьютере запускается несколько серверов, администратору для достижения максимальной производительности работы приходится также изменять операционные параметры работы сервера. Стремительное развитие возможностей MySQL вынуждает администратора пристально следить за новинками и вовремя обновлять систему MySQL посредством инсталляции новых версий. Кроме того, существуют и другие задачи администрирования, однако о них речь пойдет в главе 12, "Безопасность", и в главе 13, "Поддержка и восстановление баз данных".

В этой и следующих главах описывается несколько программ, исключительно полезных для выполнения задач администрирования MySQL.

- Программа `mysqld`, являющаяся собственно сервером MySQL.
- Сценарии запуска сервера `mysqld_safe`, `mysql.server` и `mysqld.multi` (до версии 4 сценарий `mysqld_safe` назывался `safe_mysqld`).
- Программа `mysqladmin` позволяет выполнять всевозможные административные функции.
- Программа `mysqldump` `mysqlhotcopy` используется для резервирования и копирования баз данных.
- Программы `myisamchk`, `myisamchk` и `isamchk` применяются для проверки целостности данных таблиц и операций отладки.

Большая часть материала этой главы будет лучше воспринята в том случае, если читатель уже имеет понятие о структуре каталога данных MySQL, где сервер хранит базы данных, о регистрационных журналах и др. Подробнее обо всем этом можно узнать в главе 10, "Каталог данных MySQL". Дополнительную информацию об SQL-операторах и о программировании можно почерпнуть в приложении Г, "Синтаксис SQL", и приложении Д, "Программы MySQL".

Защита новой инсталляции MySQL

Процедура инсталляции MySQL устанавливает каталог данных сервера и добавляет в него две базы данных:

- база данных `mysql`, содержащая таблицы разрешений;
- база данных `test`, которая может быть использована для тестирования.

В результате инсталляции MySQL (например, в соответствии с инструкциями, приведенными в приложении А, "Получение и инсталляция программного обеспечения") привилегии в таблице разрешений базы данных `mysql` устанавливаются таким образом, что подключиться к серверу без пароля сможет любой пользователь. Это небезопасно, поэтому необходимо установить некоторые пароли. При установке второго сервера на платформе, на которой сервер MySQL уже был установлен, вам потребуется установить пароли для нового сервера. Однако в таком случае можно попасть в затруднительное положение, описанное ниже, в разделе "Установка паролей для второго сервера". При инсталляции новой версии MySQL поверх существующей, в которой таблицы разрешений уже установлены, этот раздел можно пропустить. В примерах, приведенных в дальнейшем, используется имя узла сервера MySQL `cobra.snake.net`, которое при применении данных инструкций на практике необходимо поменять на имя своего узла. В этих примерах также предполагается, что сервер уже работает.

Как установлены таблицы при запуске

Сразу после первой инсталляции MySQL привилегии в таблице разрешений базы данных `mysql` устанавливаются следующим образом.

- Учетные записи root. Пользователь root обладает всеми возможными правами (включая и административные) и может выполнять любые операции. (Кстати, совпадение имен суперпользователей MySQL и UNIX — всего лишь совпадение. Они никоим образом друг на друга не влияют.)
- Права анонимного доступа предоставляются всем пользователям, подключающимся с локального компьютера к базе данных test или любой другой базе данных, имя которой начинается со слова test_. Анонимные пользователи могут выполнять любые операции с такими таблицами, но не обладают привилегиями администратора. Однако для ОС Windows существует одно существенное исключение, которым можно будет воспользоваться.

В таблице user перечислены все известные серверу MySQL учетные записи базы данных mysql, там же можно найти и первоначальные учетные записи пользователя root и анонимного пользователя. При этом ни один из них не имеет паролей, так как при этом предполагается, что администратор потом создаст свои собственные пароли. Поэтому одним из ваших первых действий при администрировании инсталляции MySQL должна быть установка паролей, по крайней мере для привилегированных учетных записей. Иначе случайные пользователи могут запросто получить доступ к вашему серверу. После того как были защищены начальные учетные записи, можно перейти к созданию других учетных записей, которые бы позволили членам сообщества пользователей подключаться к серверу с соответствующими привилегиями под именами, определенными для них. (Инструкции о создании новых учетных записей приведены ниже, в разделе "Управление пользовательскими учетными записями".)

Записи в таблице user содержат значение Host, определяющее, откуда может подключаться данный пользователь, и значения User и Password, определяющие имя и пароль, которые должен указать пользователь при подключении с этого узла. Таблица user также содержит столбцы, отражающие привилегии суперпользователя, которые имеет учетная запись.

На компьютерах под управлением UNIX для их инициализации достаточно запустить сценарий mysql_install_db. Этот сценарий устанавливает таблицы разрешений в базе данных mysql. Сценарий mysql_install_db инициализирует таблицу user следующим образом.

Узел	Пользователь	Пароль	Привилегии суперпользователя
localhost	root		Все
cobra.snake.net	root		Все
localhost			Отсутствуют
cobra.snake.net			Отсутствуют

Эти учетные записи позволяют сделать следующие подключения к серверу.

- Учетные записи root позволяют подключиться к серверу с локального компьютера localhost или с компьютера cobra.snake.net. Например, с компьютера cobra.snake.net клиент может подключиться как пользователь root с помощью одной из следующих команд:

```
% mysql -h localhost -u root
% mysql -h cobra.snake.net -u root
```

Как пользователь root, клиент обладает всеми правами и может производить любые операции.

- Учетные записи с пустыми значениями в столбце user соответствуют анонимным пользователям. Они позволяют подключаться к локальному серверу без указания какого-либо имени пользователя:

```
% mysql -h localhost
% mysql -h cobra.snake.net
```

Анонимные пользователи не имеют привилегий суперпользователя, но другая таблица разрешений (таблица db, о которой здесь не упоминалось) указывает, что анонимные пользователи могут использовать базу данных test или любую другую, имя которой начинается с test.

Для сервера MySQL, работающего под управлением Windows, каталог данных и база данных mysql предварительно инициализированы в дистрибутиве MySQL таким образом, что учетные записи установлены несколько иным способом, чем в ОС UNIX. Таблица user для Windows имеет следующую структуру.

Узел	Пользователь	Пароль	Привилегии суперпользователя
localhost	root		Все
%	root		Все
localhost			Все
%			Отсутствуют

В этих записях значение столбца Host, равное %, действует как групповой символ, означая, что пользователь, указанный в столбце User, может подключаться с любого узла. Таким образом, первоначальная таблица user для ОС Windows задает учетные записи следующим образом.

- Учетные записи root позволяют подключиться к серверу с локального компьютера, пользуясь узлами localhost, или с любого удаленного узла. Как пользователь root, клиент обладает всеми правами и может производить любые операции.
- Возможно анонимное подключение без указания имени пользователя. Подключившись как анонимный пользователь с локального узла, клиент получает привилегии суперпользователя и может выполнять любые операции. Под управлением ОС Windows таблица db позволяет анонимным пользователям использовать базу данных test или любую другую, имя которой начинается с test:

```
% mysql -h localhost
% mysql -h cobra.snake.net
```

Анонимные пользователи не имеют привилегий суперпользователя, но другая таблица разрешений (таблица db, о которой здесь не упоминается) указывает, что анонимные пользователи могут использовать базу данных test или любую другую, имя которой начинается с test.

Для ОС Windows существенным усложнением является тот факт, что одна из учетных записей root имеет значение % в столбце Host, а это означает, что любой пользователь из любого узла может подключиться как root без какого-либо пароля. Это делает ваш сервер полностью уязвимым; таким образом, эту учетную запись необходимо закрыть, присвоив ей определенный пароль. Кроме того, тот факт, что анонимный пользователь на узле localhost имеет привилегии суперпользователя, означает, что такая мера, как присвоение паролей только пользователю root, будет явно недостаточной. Также отличным решением является присвоение определенного пароля анонимному пользователю, для того чтобы отключить его привилегии суперпользователя или удалить его полностью. В следующих разделах рассмотрены все три варианта.

Установка паролей для первоначальных учетных записей MySQL

Этот раздел посвящен разнообразным приемам установки паролей для учетных записей root. В зависимости от используемого вами метода, для того чтобы изменения возымели действие, может потребоваться перезагрузить таблицы разрешений на сервере. (Сервер выполняет управление доступом, пользуясь копиями таблиц разрешений, хранящихся в памяти. Для некоторых методов изменения паролей в таблице user сервер может не распознать внесенных изменений, поэтому потребуется перезагружать таблицы явным образом.) В данном разделе также приведены варианты работы с анонимными суперпользователями, которые имеются в таблице user после инсталляции в ОС Windows.

Первый метод заключается в использовании возможностей программы mysqladmin:

```
% mysqladmin -h localhost -u root password "rootpass"
% mysqladmin -h cobra.snake.net -u root password "rootpass"
```

Этот метод применим как в ОС UNIX, так и в Windows. Слово "password", присутствующее в этих командах, отражает операцию, заказанную программе mysqladmin (в данном случае установить пароль), а *rootpass* отражает новое значение, которое будет присвоено паролю. Обе команды необходимы для программы mysqladmin. Первая команда устанавливает пароль для учетной записи root, привязанной к узлу localhost, а вторая — для учетной записи root, привязанной к узлу cobra.snake.net. (В Windows вторая команда должна устанавливать пароль для учетной записи root, привязанной к значению Host, равному %.)

Второй способ установки паролей заключается в использовании операторов SET PASSWORD. В операторе в формате '*user_name*'@'*host_name*' задаются значения User и Host записи таблицы user, которые подлежат изменениям. Для ОС UNIX измените пароли следующим образом:

```
% mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('rootpass');
mysql> SET PASSWORD FOR 'root'@'cobra.snake.net' =
PASSWORD('rootpass');
```

Для ОС Windows второй оператор будет несколько видоизмененным, так как одна из учетных записей root будет иметь другое значение Host:

```
% mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('rootpass');
mysql> SET PASSWORD FOR 'root'@'%' = PASSWORD('rootpass');
```

Третий способ установки паролей заключается в прямой модификации таблицы `user`. Он применим для любой версии СУБД MySQL и может оказаться единственной возможностью при использовании *действительно* старой версии СУБД MySQL, которая не имеет в своем арсенале команд `mysqladmin password` и `SET PASSWORD`. Для того чтобы задать пароль для обеих записей `root` одновременно, необходимо сделать следующее:

```
% mysql -u root
mysql> USE mysql;
mysql> UPDATE user SET Password=PASSWORD('rootpass') WHERE User='root';
mysql> FLUSH PRIVILEGES;
```

При использовании команды `mysqladmin password` или `SET PASSWORD` для изменения паролей сервер "замечает", что были внесены изменения в таблицах разрешения, и автоматически перезагружает их для обновления копий таблиц, хранящихся в памяти. При использовании оператора `UPDATE` для прямого изменения таблицы `user` серверу потребуется перезагрузить сервер явным образом. В СУБД MySQL версии 3.22.9 или выше это можно сделать с помощью оператора `FLUSH PRIVILEGES`, что и показано в предыдущем примере. Для перезагрузки таблицы разрешения также можно использовать программу `mysqladmin`:

```
% mysqladmin -u root reload
% mysqladmin -u root flush-privileges
```

Команда `reload` должна работать для любой версии MySQL; команда `flush-privileges` появилась только в MySQL 3.22.12. Таким образом, когда говорится: "Перезагрузите таблицы разрешения", это значит, что можно, в зависимости от ситуации, воспользоваться любым из трех только что показанных методов. (В примерах, приведенных в оставшейся части этой главы, в основном используется команда `FLUSH PRIVILEGES`.)

После того как был установлен пароль для пользователя `root` (и в случае необходимости перезагружены таблицы разрешения), для подключения к серверу в качестве пользователя `root` требуется задать новый пароль:

```
% mysql -p -u root
Enter password: rootpass
mysql>
```

Другой побочный эффект от установки пароля `root` заключается в том, что теперь, не зная пароля, никто не сможет подключиться как пользователь `root`, в чем и заключается основная цель этого упражнения.

С этого момента указывать пароль при подключении пользователя `root` необходимо не только для `mysql`, но и для таких программ, как `mysqladmin`, `mysqldump` и т.д. Для простоты во многих примерах этой главы не будут показываться параметры `-u` и `-p`; при этом предполагается, что они по мере необходимости будут добавляться при подключении к серверу в качестве сервера `root`.

На этом этапе таблица `user` по-прежнему содержит записи с анонимным пользователем, не имеющим пароля. Если эти записи не нужны, можно подумать о полном их удалении. Для этого подключитесь к серверу как `root` (используя, конечно, новый пароль), затем удалите из таблиц `user` и `db` записи с пустым значением `User` и перезагрузите таблицы разрешения:

```
% mysql -p -u root
Enter password: rootpass
mysql> USE mysql;
mysql> DELETE FROM user WHERE User = '';
mysql> DELETE FROM db WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

Если же не изменить статус анонимного пользователя, необходимо помнить, что в ОС Windows локальный анонимный пользователь имеет те же привилегии, что и пользователь root. Для того чтобы ослабить возможности этой учетной записи до такого же уровня, каким обладает удаленный анонимный пользователь, его суперпользовательские привилегии необходимо аннулировать. Для этого необходимо подключиться к серверу как root и задать следующие команды:

```
mysql> REVOKE ALL ON *.* FROM ''@'localhost';
mysql> REVOKE GRANT OPTION ON *.* FROM ''@'localhost';
```

Проблему этой учетной записи можно решить и другим способом — присвоить ему пароль, например:

```
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('anonpass');
```

Негативные последствия от сохранения анонимных пользователей заключаются в том, что они приводят к загадочному феномену, описанному в разделе "Головоломка с привилегиями" главы 12, "Безопасность". Но этот раздел можно прочитать после того, как вы ознакомитесь с основным материалом данной главы, в котором описана процедура создания новых учетных записей.

Установка паролей для второго сервера

Во всех вышеперечисленных инструкциях предполагалось, что вы хотите установить пароли в системе, где сервер MySQL еще не устанавливался. Если сервер MySQL уже установлен на компьютере и вы попытаетесь установить пароли для нового сервера в другом каталоге той же самой машины, он обнаружит, что все попытки подключиться к новому серверу без пароля приведут к диагностическому сообщению такого вида:

```
% mysql -u root
ERROR 1045: Access denied for user: 'root@localhost' (Using password: YES)
```

Вот это да! Почему сервер сообщает о каком-то принятом пароле, если никто ничего не вводил? Это обычно свидетельствует о том, что в конфигурационном файле перечислены пароли для входа в предварительно установленный сервер. Программа mysql находит этот конфигурационный файл и автоматически использует перечисленные здесь пароли. Для того чтобы обойти это и явно задать работу "без пароля," воспользуйтесь параметром -p и нажмите клавишу <Enter>, после чего mysql запросит пароль:

```
% mysql -p -u root
Enter password: <Enter>
```

Этой стратегией можно воспользоваться при работе с программой mysqladmin и другими программами MySQL. Дополнительное описание работы с несколькими серверами содержится ниже, в разделе "Запуск нескольких серверов".

Настройка процедур запуска и завершения работы сервера MySQL

Одна из основных обязанностей администратора MySQL – обеспечить надежную работу сервера на протяжении длительного периода времени, что даст возможность пользователям получать доступ к нему в любое удобное время. Иногда, однако, возникает необходимость и во временной приостановке работы сервера. (Так, например, при переносе базы данных необходимо убедиться, что сервер в это же время не обновляет ее таблицы.) Эта книга не поможет найти компромисс между необходимостью поддерживать постоянную работу сервера и желанием иногда временно приостановить его работу, поскольку зачастую решение этого компромисса зависит от конкретной ситуации. По крайней мере читатель сможет узнать, как запустить и завершить работу сервера. Многие аспекты этих процедур для ОС UNIX и Windows различаются, о чем будет сказано отдельно.

Запуск сервера MySQL под управлением ОС UNIX

Под управлением ОС UNIX сервер MySQL может запускаться вручную или автоматически. Кроме того, есть возможность настроить автоматическую работу сервера во время загрузки системы как части процедуры загрузки. (В действительности это и есть нормальная процедура запуска сервера после того, как все его компоненты были инсталлированы.) Но до того, как приступить к обсуждению процедуры запуска сервера, рассмотрим, под какой учетной записью он должен запускаться. В такой многопользовательской операционной системе, как ОС UNIX, существует возможность выбора учетной записи, которая будет использоваться при запуске сервера. Например, при ее запуске вручную сервер будет работать под управлением операционной системы UNIX, под которой он был зарегистрирован. Другими словами, если пользователь зарегистрировался как пользователь `paul` и запустил сервер, он будет работать как пользователь `paul`. Если же при этом сначала с помощью команды `su` сделать переключение на пользователя `root` и затем запустить сервер, то он будет работать как пользователь `root`. Администратору следует помнить о двух целях, преследуемых при настройке процедуры запуска сервера MySQL.

- **Желательно, чтобы сервер не обладал привилегиями пользователя `root`.** В большинстве случаев рекомендуется ограничивать возможности всех процессов, не требующих прав доступа `root`. Не нужны эти права и серверу `mysqld`. Когда говорят, что сервер работает "как определенный пользователь", это значит, что процесс сервера связан с идентификатором учетной записи этого пользователя и что он имеет привилегии чтения и записи файлов в файловую систему, которые имеет данный пользователь в системе. Это создает определенные трудности, особенно для процессов, которые работают как пользователи `root`, так как пользователю `root` разрешено делать все, а это очень опасно. (Некоторые из возможных проблем описываются в главе 12, "Безопасность".) Одним из способов избежать этого является отключение особых привилегий. Процессы, запускающиеся как `root`, имеют возможность изменить свои пользовательские идентифика-

ры на идентификатор другой учетной записи, поменяв, таким образом, привилегии пользователя root на привилегии обычного непривилегированного пользователя. Это делает процесс менее опасным. В целом необходимо ограничивать возможности любого процесса, если ему не нужны реальные возможности, присущие пользователю root, которые, в частности, для сервера mysqld не требуются. Серверу требуется только доступ к содержимому каталога данных MySQL. Это означает, что если сервер запускается как root, то во время процедуры запуска он должен поменять пользователя, чтобы в дальнейшем он работал как непривилегированный пользователь. (Исключением является ОС Solaris, когда сервер перегружен операциями свопинга и требуется принудительно оставить его в памяти с помощью параметра --memlock. Этот параметр требует работы сервера под управлением пользователя root.)

- **Желательно, чтобы сервер все время работал под именем одного и того же пользователя.** Весьма нелогично запускать сервер в разное время под именами различных пользователей, поскольку в этом случае файлы и каталоги с данными будут создаваться разными владельцами. В определенный момент времени это может привести к невозможности получения доступа к базам данных или таблицам. Постоянный запуск сервера от имени одного пользователя позволяет избежать этой проблемы.

Запуск сервера MySQL непривилегированным пользователем

Для запуска сервера под управлением обычного пользователя, не обладающего широкими правами, выполните следующие действия.

1. Завершите работу сервера, если он работает.

```
% mysqladmin -p -u root shutdown
```

2. Выберите учетную запись, предназначенную для запуска сервера. Сервер mysqld может работать от имени любого пользователя, однако лучше создать для него отдельную учетную запись. Можно также создать специальную группу для работы с MySQL. Предположим, что такой пользователь и группа уже созданы и имеют имена mysqladm и mysqlgrp. Пользователям, определившим другие имена для этих целей, придется подставлять их далее в примерах кода вместо mysqladm и mysqlgrp. Если же MySQL был установлен под управлением какого-либо пользователя, не обладающего специальными правами администратора, сервер наверняка будет запускаться под именем этого пользователя. В этом случае необходимо заменить mysqladm и mysqlgrp соответствующими регистрационным именем и именем группы. Если же система MySQL инсталлировалась под управлением ОС Linux из RPM-файла, в процессе установки автоматически создается учетная запись с именем mysql. Ее в последующих примерах этой книги нужно будет использовать вместо mysqladm.

3. Создайте, если необходимо, учетную запись сервера, используя обычную процедуру создания учетных записей системы. Для этого необходимо сначала зарегистрироваться в качестве пользователя root.

4. Измените права владения каталогом данных, а также всех его подкаталогов и файлов, чтобы новым владельцем этих элементов оказался пользователь mysqladm. Например, чтобы пользователь mysqladm стал владельцем каталога данных /usr/local/mysql/data, необходимо ввести следующие команды (зарегистрировавшись в качестве пользователя root):

```
# chown -R mysqladm.mysqlgrp /usr/local/mysql/data
```

5. Измените полномочия доступа к каталогу данных и ко всем его подкаталогам и файлам, чтобы работать с ними мог только пользователь mysqladm. Запретите доступ к данным всем остальным пользователям — это самая эффективная мера предосторожности. Если каталог данных размещается в каталоге /usr/local/mysql/data, определить права владения на него для пользователя mysqladm можно с помощью следующих команд (зарегистрировавшись в качестве пользователя root):

```
# chmod -R go-rwx /usr/local/mysql/data
```

Последняя пара шагов является частью более широкой процедуры закрытия, подробнее описанной в главе 12, "Безопасность". Обязательно ознакомьтесь с дополнительными инструкциями по присвоению прав владения и установлению режимов, особенно если инсталляция СУБД MySQL имеет нестандартную структуру.

После завершения предыдущей процедуры необходимо гарантировать, чтобы сервер всегда запускался с параметром --user=mysqladm так, чтобы после его запуска под управлением пользователя root он переключался на идентификатор пользователя mysqladm. (Это справедливо в том случае, когда сервер запускается под управлением root, и для установки сервера во время процедуры запуска вашей системы. Системы UNIX выполняют процедуры запуска как пользователь ОС UNIX root; таким образом, любой процесс, запущенный во время этой процедуры, по умолчанию выполняется с привилегиями пользователя root.) Наилучшим способом избежать неоднозначности объявления пользователя является указание его явным образом в конфигурационном файле. Например, в файл /etc/my.cnf можно добавить следующую строку:

```
[mysqld]
user=mysqladm
```

Подробнее о конфигурационных файлах рассказывается ниже, в разделе "Определение параметров запуска". При запуске сервера и регистрации как mysqladm наличие строки user в конфигурационном файле приведет к выдаче диагностического сообщения о том, что такая возможность может использоваться только пользователем root. Это значит, что у сервера нет возможности изменить его пользовательский идентификатор и он будет работать как mysqladm. Собственно, этого мы и добивались, поэтому это предупреждение следует проигнорировать.

Параметр --user был добавлен в mysql в версии MySQL 3.22. В более ранних версиях для этих же целей можно использовать команду su, указывающую системе, работающей под управлением пользователя root, что запускать сервер необходимо с учетной записью другого пользователя. Настоятельно рекомендуется заглянуть в справочное руководство и ознакомиться с синтаксисом команды su, поскольку его вид в различных версиях он может сильно видоизменяться.

Методы запуска сервера

После определения учетной записи, которая будет использоваться для работы сервера, можно приступать к выбору способа запуска. Сервер можно запускать вручную из командной строки или автоматически в процессе загрузки системы. Существуют три основных метода запуска сервера.

- **Непосредственный вызов `mysqld`.** Это, возможно, самый распространенный метод, поэтому мы не будем рассматривать его подробно. Достаточно лишь заметить, что в случае необходимости можно использовать команду `mysqld --help` для определения параметров, используемых с другими методами запуска.
- **Вызов сценария `mysqld_safe`.**

  Сценарий `mysqld_safe` запускает сервер, следит за нормальной работой сервера и в случае сбоя перезагружает его. Сценарий `mysqld_safe` обычно используется в BSD-версиях ОС UNIX, в других версиях используется сценарий `mysql.server`. (До MySQL 4 сценарий `mysqld_safe` назывался `safe_mysqld`, что необходимо принимать во внимание при учете инструкций, относящихся к сценарию `mysqld_safe`.) Сценарий `mysqld_safe` перенаправляет сообщения об ошибках и другую диагностическую информацию, поступающую от сервера, в диагностический файл с сообщениями об ошибках, расположенный в каталоге данных.

  Сценарий `mysqld_safe` передает право владения журнала регистрации ошибок пользователю, который задан параметром `--user`. Использование различных значений `--user` в разные моменты времени может привести к определенным проблемам. Первым симптомом возникновения этих проблем является попытка сценария `mysqld_safe` записать в журнал регистрации ошибок ошибку, завершающуюся "permission denied". При этом при попытке просмотреть журнал регистрации ошибок на предмет причины возникновения ошибки в нем не будет содержаться сколько-нибудь полезной информации на этот счет. При возникновении этой ошибки необходимо удалить существующий журнал регистрации ошибок и повторно запустить сценарий `mysqld_safe`.
- **Вызов сценария `mysql.server`.** Этот сценарий запускает сервер посредством запуска сценария `safe_mysqld`. Сценарий `mysql.server` предназначен для использования в качестве своеобразной оболочки на компьютерах с системой запуска/завершения работы System V. Данная система включает несколько каталогов со сценариями, вызываемыми при входе или выходе с определенного уровня работы. С помощью соответствующих аргументов `start` и `stop` можно определить, что делать дальше: запустить сервер или остановить его работу.
- **Для координации работы нескольких серверов используйте сценарий `mysqld_multi`.**

  По сравнению с другими методами этот является самым сложным (см. раздел "Запуск нескольких серверов").

Сценарии `mysqld_safe` и `mysqld_multi` находятся в подкаталоге `bin` каталога инсталляции MySQL. Их же можно найти в каталоге `scripts` исходного ди-

стрибутива MySQL. Сценарий `mysql.server` можно отыскать в подкаталоге `share/mysql` каталога инсталляции или в каталоге `supportfiles` исходного дистрибутива MySQL. Для использования эти сценарии необходимо скопировать в соответствующие каталоги запуска. При установке MySQL из RPM-файла, полученного с Web-узла MySQL, сценарий `mysql.server` устанавливается под именем `mysql`, и его можно найти в каталоге `/etc/rc.d/init.d`. При использовании RPM-файла MySQL RPM, полученного от RedHat, аналогичный сценарий запуска устанавливается под именем `mysqld`.

Приготовления, необходимые для выполнения сценария запуска во время загрузки системы, зависят от типа вашей системы. Ознакомьтесь со следующими примерами и используйте или адаптируйте команды для того, чтобы они могли полностью соответствовать процедурам запуска вашей системы. В ОС BSD-UNIX довольно часто используется несколько специальных файлов, которые располагаются в каталоге `/etc` и инициируют службы во время запуска. Как правило, имена таких файлов начинаются с "`rc`". Файл `rc.local` (или имеющий подобное название), например, предназначен специально для запуска локальных служб. Для запуска сервера в подобных системах необходимо добавить в файл `rc.local` следующие строки:

```
if [ -x /usr/local/bin/safe_mysqld ]; then
   /usr/local/bin/safe_mysqld &
fi
```

По необходимости введите изменения в путь к каталогу `bin`.

В системах System V для инсталляции сценария `mysql.server` достаточно разместить его в подкаталоге `/etc` каталога запуска. Это наверняка уже сделано, если Linux или MySQL инсталлировались из RPM-файла. Если нет, инсталлируйте сценарий в основной каталог сценариев запуска, убедитесь в том, что он является исполняемым, и установите связи с ним в соответствующих каталогах уровней запуска.

Примечание

Здесь мы продолжаем считать, что `mysql.server` устанавливается в стартовый каталог под именем `mysql`, но он по-прежнему будет описываться как `mysql.server`, чтобы было понятно, о чем идет речь.

Структура каталогов с файлами запуска в различных системах может варьироваться, поэтому рекомендуется внимательно просмотреть, как эти файлы организованы в используемом компьютере. Например, в системе Solaris общий уровень многопользовательского запуска равен 2. Для запуска применяются каталог `/etc/init.d` и каталог рабочего уровня `etc/rc2.d`. Соответственно инсталляция сценария выполняется с помощью следующих команд:

```
# cp mysql.server /etc/init.d
# cd /etc/init.d
# chmod +x mysql.server
# cd /etc/rc2.d
# ln -s ../init.d/mysql.server S99mysql
```

Во время запуска процедура загрузки автоматически запустит сценарий `S99mysql` с аргументом `start`. ОС Linux имеет подобный набор каталогов, организованных под `/etc/rc.d` (например, `/etc/rc.d/init.d` и `/etc/rc.d/rc3.d`). В Linux-системах имеется возможность использования команды `chkconfig` (часто приме-

няемой под управлением Linux), ее также можно применить для инсталляции сценария `mysql.server`. В этом случае от ввода приведенных выше команд вручную можно отказаться. Ниже показано, как можно под именем `mysql` установить в стартовые каталоги сценарий `mysql.server`.

1. Скопируйте сценарий `mysql.server` в каталог `init.d` и сделайте его исполняемым:

```
# cp mysql.server /etc/rc.d/init.d/mysql
# chmod +x /etc/rc.d/init.d/mysql
```

2. Зарегистрируйте сценарий и активизируйте его:

```
# chkconfig --add mysql
# chkconfig mysql on
```

3. Чтобы проверить правильность активизации сценария, запустите сценарий `chkconfig` с параметром `--list`:

```
# chkconfig --list mysql
mysql 0:off 1:off 2:on 3:on 4:on 5:on 6:off
```

   Отсюда видно, что сценарий будет выполняться во время запуска для рабочих уровней 3–5.

   Не имея сценария под рукой `chkconfig`, можно воспользоваться процедурой, аналогичной той, что используется для ОС Solaris, хотя имена путей могут немного отличаться. Для того чтобы активизировать рабочий уровень 3, используйте следующие команды:

```
# cp mysql.server /etc/rc.d/init.d/mysql
# cd /etc/rc.d/init.d
# chmod +x mysql
# cd /etc/rc.d/rc3.d
# ln -s ../init.d/mysql S99mysql
```

Под управлением ОС Mac OS X процедура запуска немного отличается. Каталоги `/Library/StartupItems` и `/System/Library/StartupItems` содержат подкаталоги для служб, которые инициализируются во время запуска системы. Установить MySQL можно, пользуясь существующими службами в качестве модели или, что еще проще, посетив Web-узел `http://www.entropy.ch/software/macosx/mysql/`, где можно найти полный стартовый пакет. После установки этого пакета можно модифицировать его главный сценарий по своему усмотрению.

Работа сервера MySQL под управлением ОС Windows

Дистрибутивы MySQL для Windows включают несколько серверов, имеющих различные параметры. Обзор различных серверов дается в приложении A, "Получение и инсталляция программного обеспечения". Здесь для примера будут использоваться `mysqld`, который может работать на любой версии ОС Windows, где работает MySQL, и сервер `mysqld-nt` – для примеров, которые больше подходят для версий ОС Windows NT, 2000 и XP.

Сервер может быть запущен вручную из командной строки под любой версией ОС Windows. Кроме того, в NT-системах имеется возможность запускать сервер как службу. Можно задать автоматический запуск службы MySQL при запуске

Windows и управлять ею из командной строки или использовать диспетчер служб Windows. Серверы, созданные специально для NT, можно настроить таким образом, чтобы клиенты подключались через именованные каналы.

Запуск сервера вручную

Для запуска сервера вручную необходимо задать команду

```
C:\> mysqld
```

Параметр `--console` предназначен для перенаправления вывода диагностических сообщений с журнала регистрации ошибок (файл `mysql.err`, находящийся в каталоге данных) в окно консоли:

```
C:\> mysqld --console
```

Возможность осуществления соединений через именованные каналы обеспечивается программой `mysqld-nt` на NT-системах. Поддержка именованных каналов для `mysqld-nt` включена по умолчанию до MySQL 3.23.49. Сейчас положение несколько изменилось, и начиная с MySQL 3.23.50 и выше, для того, чтобы воспользоваться преимуществами, которые предоставляют именованные каналы, во время запуска сервера необходимо задавать параметр `--enablenamed-pipe`. (Но это не очень хорошо! Причина, по которой поименованные каналы по умолчанию отключены, заключается в том, что на практике оказалось, что они могут вызывать проблемы во время выключения на машинах различных типов. Поэтому обязательно нужно убедиться в том, что ваш сервер хорошо выключается.) Для того чтобы завершить работу сервера, необходимо использовать `mysqladmin`:

```
C:\> mysqladmin -p -u root shutdown
```

Запуск сервера как службы

На NT-версиях ОС Windows сервер MySQL может устанавливаться как служба с помощью следующей команды:

```
C:\> mysqld-nt --install
```

Это позволит запускать сервер автоматически при запуске Windows. Запускать сервер как службу можно и в неавтоматическом режиме как службу, запускаемую вручную:

```
C:\> mysqld-nt --install-manual
```

В этих примерах используется `mysqld-nt`, но в качестве службы устанавливается любой сервер. Вместо этого сервера можно установить сервер `mysqld`, однако такая возможность, как использование именованных каналов, будет отсутствовать.

Как общее правило, при запуске сервера как службы параметры задаются не в командной строке, а в конфигурационном файле. (Подробно об этом — ниже, в разделе "Определение параметров запуска".) Исключением из этого правила является запуск сразу нескольких серверов Windows. Подробнее об этом рассказывается в разделе "Запуск нескольких серверов".

После установки сервера как службы им можно управлять, указав его имя `MySql`. Это можно сделать из командной строки или из диспетчера служб ОС Windows, обладающих графическим интерфейсом. Диспетчер служб представлен элементом Services или Administrative Tools панели управления Windows (в зависимости от версии Windows).

Для запуска или остановки службы из командной строки используйте следующие команды (имя службы можно указывать в любом регистре, так как оно не чувствительно к регистру):

```
C:\> net start MySql
C:\> net stop MySql
```

Диспетчер служб выводит окно со списком известных ему служб и дополнительную информацию о состоянии этой службы и порядке ее запуска (автоматически или вручную). Для запуска или остановки сервера MySQL выберите его из списка служб и щелкните на соответствующей кнопке или выберите необходимый элемент меню.

Кроме того, сервер можно отключить из командной строки с помощью команды `mysqladmin shutdown`.

Для того чтобы удалить сервер из списка служб, отключите его, если он работает, и введите следующую команду:

```
C:\> mysqld-nt --remove
```

> **Примечание**
>
> Несмотря на то что работой служб можно управлять как с помощью диспетчера служб, так и с помощью команд DOS, необходимо избегать применения обоих подходов одновременно. *Перед тем как задавать команды из командной строки, обязательно убедитесь в том,* что диспетчер служб закрыт.

Определение параметров запуска

Есть два способа определения дополнительных параметров запуска, которые применяются при загрузке сервера.

- Можно задать параметры как в длинной, так и в короткой форме непосредственно в строке вызова сервера. Например, можно задать параметры `--user=mysqladm` или `-u mysqladm`.
- Параметры можно определить в конфигурационном файле. При задании параметра таким способом может использоваться только длинная форма и без ведущих тире:

  ```
  user=mysqladm
  ```

Профессионалы рекомендуют по возможности использовать для этих целей глобальные конфигурационные файлы. Их можно использовать при любом методе запуска, и, если здесь заданы параметры, они будут работать при каждом запуске сервера. Указание перечня параметров в командной строке допустимо только при запуске сервера вручную или со сценарием `mysqld_safe`. Это неприменимо к сценарию `mysql.server`, который поддерживает в командной строке только параметры `start` и `stop`. Кроме того, с некоторыми исключениями нельзя использовать параметры `--install` и `--install-manual` запуска в командной строке при установке сервера в ОС Windows как службы. (Исключения описаны в разделе "Запуск нескольких серверов".)

В системе UNIX этими файлами обычно являются `/etc/my.cnf` и `my.cnf`. В системе ОС Windows в качестве таковых являются `my.ini` в системном каталоге Windows, `C:\my.cnf` и `my.cnf` — в каталоге данных. Если такого файла не су-

ществует, его необходимо создать. Чаще всего параметры запуска сервера располагаются в параметрической группе [mysqld].

Например, для того, чтобы показать, что сервер должен запускаться как mysqladm, и воспользоваться основным каталогом /usr/local/mysql, необходимо задать в конфигурационном файле следующую группу:

```
[mysqld]
user=mysqladm
basedir=/usr/local/mysql
```

Что эквивалентно запуску сервера с помощью командной строки с параметрами следующего вида:

```
% mysqld --user=mysqladm --basedir=/usr/local/mysql
```

Полный перечень групп, которые используются сервером и программами запуска сервера, приведен в следующей таблице.

Программа	Параметрические группы, которые используются программой
mysqld	[mysqld], [server]
mysqld_safe	[mysqld], [server], [mysqld_safe], [safe_mysqld]
safe_mysqld	[mysqld], [server], [safe_mysqld]
mysql.server	[mysqld], [mysql_server]
libmysqld	[embedded], [server]

Строка libmysqld имеет отношение к встроенному серверу, который включается непосредственно в программы для создания приложений, для которых не требуется отдельный MySQL-сервер. (В главе 6, "Программный интерфейс MySQL C API", описано создание приложений, использующих встроенный сервер.)

Группа [server] предназначается для задания параметров, которые относятся к любому серверу, будь то автономный сервер или встроенный. Группы [mysqld] и [embedded] предназначаются для параметров, которые относятся только к автономным или только к встроенным серверам. Подобным образом, группы [mysqld_safe] и [mysql_server] позволяют задавать параметры в зависимости от имени используемого сценария.

До MySQL 4 сценарий mysqld_safe назывался safe_mysqld. Это значит, что все указания, имеющие отношение к параметрической группе [mysqld_safe], которые используются сценарием mysqld_safe, будут справедливы и для группы [safe_mysqld] при работе со сценарием safe_mysqld.

Более подробная информация о формате и синтаксисе конфигурационных файлов содержится в приложении Д, "Программы MySQL". При запуске сервера с помощью сценария запуска есть и третий способ определения параметров — это внесение изменений таким образом, чтобы сценарий передавал эти параметры прямо на сервер. Но такой метод можно использовать только в крайнем случае. В нем кроется один существенный недостаток, заключающийся в том, что о внесенных изменениях потребуется вспоминать всякий раз, когда инсталлируется новая версия MySQL, которая заменит обновленный сценарий новой версией.

Завершение работы сервера

Для самостоятельного завершения работы сервера применяется команда mysqladmin:

```
% mysqladmin -p -u root shutdown
```

Эта команда применима как для ОС UNIX, так и для ОС Windows. Если сервер работает как служба, его также можно остановить вручную из командной строки:

```
C:\> net stop MySql
```

Можно также воспользоваться графическим интерфейсом диспетчера служб для выбора и остановки сервера.

Автоматическое завершение работы сервера также не требует выполнения каких-либо специальных действий. В UNIX BSD работа служб обычно завершается после отправки сигнала TERM их процессам. Службы либо соответствующим образом отвечают на него, либо завершают свою работу. Сервер mysqld, например, на получение такого сигнала реагирует закрытием.

В системах System V, запуск сервера в которых производится с помощью сценария mysql.server, процедура завершения работы реализуется посредством вызова этого же сценария, но с аргументом stop. Можно также вызвать этот же сценарий для отключения сервера вручную. Например, если сценарий mysql.server был установлен как /etc/rc.d/init.d/mysql, остановить его можно следующим образом (для этого необходимо работать как пользователь root):

```
# /etc/rc.d/init.d/mysql stop
```

Если сервер MySQL работает как служба на Windows NT-системе, диспетчер служб автоматически "прикажет" серверу остановиться во время выключения системы. Для других версий ОС Windows (или если нет необходимости запускать сервер в качестве службы) перед отключением ОС Windows необходимо выключать его вручную командой mysqladmin shutdown из командной строки.

Когда нельзя подключиться к серверу

В некоторых случаях возникает необходимость в самостоятельной перезагрузке сервера из-за невозможности подключения к нему. Честно говоря, получается какой-то парадокс, поскольку обычно для завершения работы сервера необходимо сначала подключиться к нему командой mysqladmin shutdown, а затем дать команду на закрытие. Как же может возникнуть подобная ситуация?

Во-первых, администратор может не знать пароля MySQL-пользователя root. Иногда это случается, когда при изменении пароля случайно введен лишний управляющий символ или пароль просто забыт.

Во-вторых, подключение к компьютеру localhost обычно осуществляется через файл доменного разъема ОС UNIX, которым, как правило, является /tmp/mysql.sock. Удаление этого файла делает невозможным подключение локальных клиентов. Такая ситуация, в свою очередь, может возникнуть после запуска процесса cron, который удаляет временные файлы из каталога /tmp.

Если подключиться нельзя из-за отсутствия файла разъема, проблему можно легко решить посредством перезагрузки сервера. В процессе запуска он воссоз-

даст этот файл. Проблема заключается в том, что использовать этот разъем для установления соединения с сервером нельзя. Для этого необходимо установить соединение TCP/IP. При этом достаточно подключиться к локальному серверу, указав не имя `localhost`, а непосредственно указав IP-адрес `127.0.0.1` с помощью следующей команды:

```
% mysqladmin -p -u root -h 127.0.0.1 shutdown
```

Если файл разъема удален в результате работы задания программы `cron`, проблема может возникнуть снова. Чтобы избежать этого, настройте программу `cron` на использование другого файла разъема. Это можно осуществить с помощью глобального конфигурационного файла. Так, например, если `/usr/local/var` — каталог данных, для перемещения в него файла разъема достаточно добавить в файл `/etc/my.cnf` следующие строки:

```
[mysqld]
socket=/usr/local/var/mysql.sock

[client]
socket=/usr/local/var/mysql.sock
```

Как видите, имя файла и путь к нему одинаковы как для сервера, так и для клиентских программ, в результате чего все они используют один файл разъема. Можно определить новый путь только для сервера, благодаря чему клиентские программы будут искать этот файл по старому адресу. Перезапустите сервер после внесения изменений, чтобы создать файл разъема на новом месте. К сожалению, этот метод работает только для клиентов, считывающих конфигурационный файл, но некоторые из них этого сделать не смогут. После перекомпилирования MySQL из исходного текста можно изменить настройки с тем, чтобы для сервера и клиентов по умолчанию использовались различные пути. Это также автоматически повлияет на сторонние программы, использующие в своей работе клиентские библиотеки.

Иногда администратор не может подключиться к серверу из-за того, что забыл пароль пользователя `root` или в процессе изменения случайно присвоил ему не то значение, которое предполагал. В этом случае необходимо восстановить контроль над сервером, чтобы заново установить пароль. Это можно сделать следующим образом.

1. Завершите работу сервера. Зарегистрировавшись как пользователь `root` на компьютере с сервером, администратор может завершить работу сервера с помощью команды `kill`. Используя команду `ps`, можно отыскать ID-номер процесса сервера. С этой же целью можно просмотреть PID-файл, который обычно располагается в каталоге данных. Сначала необходимо попытаться завершить работу сервера с помощью обычной команды `kill`, а не посылать серверу сигнал TERM:

   ```
   # kill -TERM PID
   ```

   В этом случае все таблицы и журналы будут обработаны и закрыты корректно. Если в работе сервера не все нормально и он не отвечает на сигнал завершения работы, можно воспользоваться командой `kill -9` для принудительного закрытия:

   ```
   # kill -9 PID
   ```

Однако к ее помощи следует прибегать в последнюю очередь, поскольку в этом случае существует риск повредить таблицы.

При работе под управлением ОС Linux команда ps может показать сразу несколько "процессов" mysqld. Это несколько нитей одного процесса, поэтому, для того, чтобы удалить все процессы, достаточно удалить один из них. Запуск сервера с помощью сценария mysqld_safe приводит к тому, что сервер будет запущен повторно сразу же после его удаления. Поэтому здесь, перед тем, как удалить процесс mysqld, необходимо удалить процесс mysqld_safe.

При работе сервера под управлением ОС Windows сервер можно отключить, не зная при этом никакого пароля, с помощью диспетчера служб или команды

```
C:\> net stop MySql
```

Для принудительной остановки сервера на Windows воспользуйтесь диспетчером задач (<Alt+Ctrl+Del>). Но как и команда kill -9 в ОС UNIX, это является крайней мерой.

2. Перезапустите сервер с указанием параметра --skip-grant-tables. Это укажет серверу не использовать таблицы разрешений для проверки соединений и позволит подключиться с полномочиями пользователя root без пароля. После удачного подключения измените пароль пользователя root. Однако при этом сервер будет открыт для любых вторжений. Поэтому сразу же после подключения введите команду FLUSH PRIVILEGES:

```
% mysql
mysql> FLUSH PRIVILEGES;
```

Оператор FLUSH вынуждает сервер перезагрузить таблицы разрешений, чтобы продолжать пользоваться ими в дальнейшем в обычном режиме. Оператор FLUSH также повторно разрешает использование оператора GRANT, который отключен, когда сервер не использует таблицы разрешений. После перезагрузки таблиц можно изменить пароль пользователя root (см. раздел "Установка программных переменных").

3. После изменения пароля root может потребоваться отключить сервер снова и перезагрузить его с помощью обычной процедуры запуска.

Независимо от того, как вы это сделаете, с помощью команды kill -9 под ОС UNIX или диспетчера задач под ОС Windows, грубая природа отключения не позволит серверу сохранить все сделанные изменения на диске. Вполне вероятно, что это закончится разрушением таблиц. Для того чтобы решить проблемы, которые возникают из-за отключений такого типа, неплохо подстраховаться и постоянно сохранять активными возможности автоматического восстановления сервера. Подробности — в главе 13, "Резервное копирование и восстановление баз данных".

Управление пользовательскими учетными записями

В обязанности администратора MySQL входит также создание и настройка учетных записей пользователей MySQL. В процессе этой настройки необходимо

определить, какие пользователи будут иметь возможность подключения к серверу, откуда они смогут подключиться и что смогут делать после подключения. Эта информация хранится в таблицах разрешения в базе данных mysql и управляется с помощью двух операторов:

- оператор GRANT создает пользователей MySQL и позволяет настроить их привилегии;
- оператор REVOKE удаляет привилегии уже существующих учетных записей.

Эти два оператора, появившиеся в MySQL 3.22.11, являются своего рода внешним интерфейсом для таблиц разрешений базы данных mysql и обеспечивают альтернативу непосредственному редактированию содержимого этих таблиц, что имело место до версии MySQL 3.22.11, когда для того, чтобы манипулировать содержимым таблицы разрешения, требовалось непосредственно оперировать такими операторами SQL, как INSERT и UPDATE. Операторы GRANT и REVOKE играют роль своеобразного "переднего края" для работы с таблицами разрешения. С ними проще работать концептуально, так как, работая с ними, администратор описывает разрешения, которые он хочет дать, а сервер автоматически трансформирует эти запросы в соответствующие изменения в таблицах разрешения. Применять операторы GRANT и REVOKE проще, чем непосредственно редактировать таблицы разрешений. Однако для более полного понимания представленного здесь материала рекомендуется также прочитать главу 12, "Безопасность", в которой таблицы разрешений описываются подробнее. Роль этих таблиц действительно велика, и администратор должен понимать, каким образом их обрабатывают операторы GRANT и REVOKE. Она также содержит раздел, посвященный установке учетных записей без использования оператором GRANT, т.е. то, как устанавливать привилегии, если версия сервера ниже 3.22.11.

Администраторы могут также поближе познакомиться со сценариями mysqlaccess и mysql_setpermission, которые являются частью дистрибутива MySQL. Они представляют собой Perl-сценарии и обеспечивают альтернативу оператору GRANT, поскольку применяются для установки пользовательских учетных записей. Для использования сценария mysql_setpermission требуется инсталляция поддержки DBI.

Операторы GRANT и REVOKE работают с четырьмя следующими таблицами:

Таблица разрешений	Содержимое
user	Подключающиеся к серверу пользователи и все их глобальные параметры и привилегии
db	Привилегии уровня баз данных
tables_priv	Привилегии уровня таблиц
columns_priv	Привилегии уровня столбцов

Существует еще одна, пятая таблица разрешений (host), однако операторы GRANT и REVOKE не в состоянии ее обрабатывать. Как с ней работать, рассказывается в главе 12, "Безопасность".

Если оператор GRANT запускается для определенного пользователя, в таблице user для него создается новая запись. Если оператор определяет для пользователя

какие-либо глобальные привилегии (привилегии администратора или привилегии, применяемые сразу ко всем базам данных), они также записываются в таблицу user. Права обработки базы данных, таблицы или столбца записываются соответственно в столбцы db, tables_priv и column_priv.

Далее в этом разделе рассказывается, как создавать учетные записи пользователей MySQL, присваивать или удалять их привилегии, а также полностью удалять пользователей из таблиц разрешений.

Создание новых пользователей и предоставление привилегий

Оператор GRANT имеет следующий синтаксис:

```
GRANT privileges (columns)
  ON what
  TO user IDENTIFIED BY 'password'
  WITH GRANT OPTION
  REQUIRE encryption requirements
  WITH grant or resource management options;
```

Для успешного его выполнения обязательно нужно правильно определить следующую информацию.

- *privileges* (привилегии). Привилегии присваиваются определенному пользователю. Например, привилегия SELECT позволяет пользователю работать с операторами SELECT, а привилегия SHUTDOWN позволяет пользователю отключать сервер.
- *columns* (столбцы). Столбцы, к которым применяются определенные привилегии. Этот параметр необязателен и используется только при установке привилегий для столбцов. Имена столбцов в списке отделяются друг от друга запятыми.
- *what* (что). Уровень применения привилегий. Привилегии могут быть глобальными (применяемыми ко всем базам данных и их таблицам), уровня баз данных (применяемыми ко всем таблицам определенной базы данных) или уровня таблицы. Используя оператор *columns*, можно определить также привилегии уровня столбца.
- *account* (учетная запись). Учетная запись, которой присваиваются привилегии. Значение *account* состоит из имени пользователя и имени узла в формате '*user_name*'@'*host_name*', так как в СУБД MySQL задаются не только те, кто подключается, но и место, откуда они могут его сделать. Таким образом, можно задать отдельные учетные записи для двух пользователей, имеющих одно и то же имя, но подключающихся из разных мест. СУБД MySQL позволяет делать различие между ними и присваивать привилегии каждому из них независимо друг от друга. Значения *user_name* и *host_name* записываются в столбцах User и Host строки таблицы user для account и в любых других записях таблицы разрешения, которые создает оператор GRANT.

  Имя пользователя в MySQL — это простое имя, которое используется для идентификации пользователя при подключении к серверу. У этого имени

нет никакой связи с регистрационным именем в ОС UNIX или ОС Windows. По умолчанию при указании имени пользователя явным образом клиентские программы используют его как имя пользователя СУБД MySQL, однако это стандартное решение. То же самое можно сказать и о пользователе `root`, который и в MySQL является суперпользователем, который может делать все. В таблицах разрешения можно поменять это имя на `nobody`, чтобы затем подключаться как `nobody` и выполнять операции, для которых требуются права суперпользователя.

- *`password`* (пароль). Присвоенный пользователю пароль, который не является обязательным. Если для нового пользователя опустить выражение `IDENTIFIED BY`, пароль ему присвоен не будет (что не совсем разумно с точки зрения безопасности). Если же этот оператор задается для уже существующего пользователя, введенный пароль заменит используемый до настоящего момента. Старый пароль останется неизменным, если новый не будет определен. Строка пароля, задаваемая с помощью выражения `IDENTIFIED BY`, должна представлять собой буквенную строку, которую при записи зашифрует оператор `GRANT`. Поэтому следует избегать использования функции `PASSWORD()`, применяемой с оператором `SET PASSWORD`.

Предложения `REQUIRE` и `WITH` являются необязательными. Предложение `REQUIRE` появилось в MySQL 4.0.0 и используется для установки учетных записей, которые должны производить подключение с помощью протокола SSL. Оператор `WITH GRANT OPTION` является необязательным. С его помощью можно предоставить пользователю все привилегии, определенные оператором `GRANT` для других пользователей. Этот оператор можно применять для делегирования возможностей определенных категорий другим пользователям. Начиная с MySQL 4.0.2 предложение `WITH` также используется для задания параметров управления ресурсами, позволяющими ограничить количество соединений или запросов, которые может обрабатывать обладатель данной учетной записи за час. Эти параметры помогут предотвратить "захват" сервера данной учетной записью.

В именах пользователей, баз данных, таблиц и паролях, записываемых в таблицу разрешений, строчные буквы отличаются от прописных. Регистр в именах компьютеров и столбцов таблиц не учитывается.

Чтобы определить, какую форму оператора `GRANT` применить, достаточно ответить на следующие вопросы.

- Кто и откуда может подключаться к серверу?
- Какой уровень привилегий должен иметь пользователь и на доступ к чему эти привилегии предоставляются?
- Требуются ли безопасные подключения?
- Необходимо ли пользователю предоставлять административные привилегии?
- Будет ли ограничиваться потребление ресурсов для данной учетной записи?

Попробуем ответить на эти вопросы и рассмотрим примеры оператора `GRANT` для создания учетных записей пользователей MySQL.

Кто и откуда может подключаться к серверу

Администратор может разрешить пользователю подключаться как с определенного компьютера, так и с разных компьютеров. В первом случае, если точно известно, что пользователи будут подключаться только со своих компьютеров, можно предоставить им такой доступ с помощью следующих операторов:

```
GRANT ALL ON sampdb.* TO boris@localhost IDENTIFIED BY 'ruby'
GRANT ALL ON sampdb.* TO fred@ares.mars.net IDENTIFIED BY 'quartz'
```

Если часть username или hostname значения *account* не содержит таких специальных символов, как '-' или '%', его необязательно брать в кавычки (например, такое имя, как boris@localhost, будет допустимым и без кавычек). Однако безопаснее будет использовать кавычки, и в примерах, приведенных в этой книге, это, как правило, делается. Но обратите внимание, что части username и hostname берутся в кавычки по отдельности 'boris'@'localhost', а не целиком как 'boris@localhost'.

В случае необходимости можно также разрешить пользователю подключаться с одного компьютера. Другой крайностью является разрешение пользователю, который много путешествует, подключаться с любого компьютера, расположенного в любой точке света. Для того чтобы разрешить пользователю max делать подключение отовсюду, необходимо воспользоваться оператором GRANT следующего вида:

```
GRANT ALL ON sampdb.* TO 'max'@'%' IDENTIFIED BY 'diamond'
```

Символ "%" заменяет все возможные значения адресов и выполняет ту же функцию, что и при сравнении с помощью оператора LIKE. В предыдущем примере его можно условно заменить фразой "любой компьютер". Установка символа "%" аналогична простому опусканию части, задающей компьютер. Это самый простой и в то же время самый небезопасный способ создать пользователя.

По необходимости можно также разрешить пользователю подключаться с ограниченного числа компьютеров. Так, чтобы пользователь mary мог подключаться с компьютеров домена snake.net, достаточно воспользоваться спецификатором %.snake.net:

```
GRANT ALL ON sampdb.* TO 'mary'@'%.snake.net' IDENTIFIED BY 'topaz'
```

Другой групповой символ LIKE ('_') может быть использован в значениях узлов, которые совпадают с любым одиночным символом.

Для определения компьютера можно применять не только имена, но и IP-адреса. Эти адреса можно задавать как явно, так и с помощью вспомогательных символов. Кроме того, в версии MySQL 3.23 появилась возможность определять IP-адреса, задавая маску сети, устанавливая число разрядов в сетевом номере:

```
GRANT ALL ON sampdb.* TO 'joe'@'192.168.128.3' IDENTIFIED BY 'water'
GRANT ALL ON sampdb.* TO ardis@192.168.128.% IDENTIFIED BY 'snow'
GRANT ALL ON sampdb.* TO rex@192.168.128.0/17 IDENTIFIED BY 'ice'
```

Первый оператор определяет только один компьютер, с которого может подключиться пользователь joe. Второй определяет набор IP-адресов для подсети класса C 192.168.128. В третьем операторе часть 192.168.128.0/17 определяет 17-разрядный сетевой номер и соответствует любому компьютеру с адресом 192.168.128 в первых 17 разрядах IP-адреса.

Применение имени компьютера `localhost` в операторе GRANT позволит подключиться пользователю к серверу из локального узла, задав значение `host`, равное `localhost` или `127.0.0.1` (IP-адрес "обратной петли" локального узла). Учетная запись `localhost` соответствует подключению с указанием имени узла, равного точке ".", если данный сервер поддерживает именованные каналы. Под UNIX подключение к `localhost` производится благодаря файлу разъема UNIX. Под управлением ОС Windows NT соединения с "." производятся через именованный канал, если таковые используются. Все остальные соединения производятся через TCP/IP, включая соединения с `127.0.0.1` (адрес интерфейса *"обратной петли"*).

Отсутствие части `host` в спецификации учетной записи аналогично использованию '`%`'. Другими словами, эквивалентными значениями *account* в данном примере будут записи `'max'` и `'max'@'%'` в операторе GRANT. Это значит, что, написав спецификацию `'boris@localhost'` вместо `'boris'@'localhost'`, с точки зрения MySQL вы не совершаете никакой ошибки. Все, что произойдет, – это то, что MySQL проинтерпретирует значение `'boris@localhost'` как содержащее только пользовательскую часть и добавит к ней стандартную узловую часть, равную '`%`'. В результате этого получим имя `'boris@localhost'@'%'`. Для того чтобы избежать таких коллизий, заключайте в кавычки пользовательскую и узловую части отдельно.

Как определить имя локального компьютера в таблице разрешений

Довольно часто пользователи не могут подключиться к серверу с основного компьютера (на котором инсталлирован сервер) из-за того, что вместо имени `localhost` указывают имя сервера. Эта проблема возникает из-за использования разных способов определения имен, записанных в таблицах разрешений и выдаваемых программам. Если процедура сервера выдает неполное имя, например `cobra`, а в таблице разрешений записано полное имя, `cobra.snake.net` (или наоборот), подключение становится невозможным.

Чтобы определить, имеет ли место такая проблема на используемом компьютере, попытайтесь подключиться к локальному серверу с использованием параметра `-h`, устанавливающего имя компьютера:

```
% mysql -h cobra.snake.net
```

Затем загляните в общий учетный файл сервера. Какое имя компьютера в нем записано, полное или неполное? Неважно, какая форма применяется. Важно использовать для определения имени компьютера в операторе GRANT именно это имя.

Какой уровень привилегий должен иметь пользователь и на доступ к чему эти привилегии предоставляются

Администратор может предоставлять пользователям привилегии разных уровней (они приведены в табл. 11.1 и детально описаны в главе 12, "Безопасность"). В этой главе привилегии описываются как с точки зрения их назначения, так и с точки зрения отношения к таблицам разрешения.

Таблица 11.1. Типы привилегий MySQL

Спецификация привилегии	Операция, разрешенная данной привилегией
`CREATE TEMPORARY TABLES`	Создать временные таблицы
`EXECUTE`	Выполнять хранимые процедуры (зарезервировано для использования в будущем)

Окончание табл. 11.1

Спецификация привилегии	Операция, разрешенная данной привилегией
FILE	Читать и записывать файлы на серверном узле
GRANT OPTION	Выделить привилегии данной учетной записи другим учетным записям
LOCK TABLES	Заблокировать таблицы с помощью оператора LOCK TABLES явно
PROCESS	Просмотреть информацию о процессах, выполняемых сервером
RELOAD	Перезагрузить таблицы разрешения или обновить протоколы или кэши
REPLICATION CLIENT	Запрос размещения главного и подчиненного серверов
REPLICATION SLAVE	Действовать как подчиненный сервер репликации
SHOW DATABASES	Задать операторы SHOW DATABASES
SHUTDOWN	Отключить сервер
SUPER	Завершить процессы и произвести другие административные действия
ALTER	Изменить таблицы и индексы
CREATE	Создать базы данных и таблицы
DELETE	Удалить существующие строки из таблиц
DROP	Удалить базы данных и таблицы
INDEX	Удалить индексы
INSERT	Добавить новые строки в таблицы
REFERENCES	Не используется (зарезервировано для использования в будущем)
SELECT	Выбрать существующие строки в таблицах
UPDATE	Обновить существующие строки в таблицах
ALL	Все действия (за исключением GRANT); синонимом является ALL PRIVILEGES
USAGE	Специальная привилегия "нет привилегий"

Спецификаторы привилегий из первой группы, представленной в этой таблице, являются привилегиями администратора. Обычно привилегии этой группы присваиваются очень осторожно, так как обладатели таких привилегий имеют возможность существенно влиять на работу сервера. (Например, привилегия SHUTDOWN для ежедневного использования не нужна.) Привилегии из второй группы применимы к базам данных, таблицам и столбцам и позволяют с помощью сервера управлять доступом к данным. Спецификаторы из третьей группы являются специальными. ALL означает "все привилегии" (за исключением того, что он не включает привилегию GRANT OPTION). USAGE обозначает "отсутствие привилегий", т.е. "создать учетную запись, но не присваивать ей никаких привилегий". USAGE также можно использовать при внесении изменений в аспекты учетной записи, которые не связаны с привилегиями, не изменяя текущие установки привилегий.

Некоторые привилегии появились в MySQL 4.0.2, поэтому их нельзя присваивать в более ранних версиях. Это такие привилегии, как CREATE TEMPORARY TABLES, EXECUTE, LOCK TABLES, REPLICATION CLIENT, REPLICATION SLAVE,

SHOW DATABASES и SUPER. Введение этих привилегий вносит изменение в управление некоторыми операциями. Например, возможность остановить любой процесс на сервере до версии MySQL 4.0.2 предоставлялась привилегией PROCESS, а в последующих версиях – с помощью привилегии SUPER. Обычно это не составляет большой проблемы, так как таблицы разрешения обновляются при обновлении сервера с версий от 4.0.2 и до 4.0.2 или выше; дистрибутив MySQL содержит сценарий `mysql_fix_privilege_tables`, который создает столбцы для хранения значений новых привилегий, а привилегии SUPER для учетных записей инициализируются с использованием привилегий PROCESS. Таким образом, пользователи, которые имели право на уничтожение процессов до версии 4.0.2, смогут делать это и в последующих версиях. (Правила переноса привилегий при обновлении до версии 4.0.2 или выше описаны в главе 12, "Безопасность".)

Следует помнить, что нельзя присваивать привилегии CREATE TEMPORARY TABLES или LOCK TABLES для определенных баз данных в MySQL 4.0.2 и 4.0.3, поэтому эти версии по возможности лучше не применять. Эта проблема была решена в MySQL версии 4.0.4.

Все привилегии можно присваивать на различных уровнях начиная с глобального уровня и заканчивая уровнем столбцов. Для управления привилегиями предназначается спецификатор предложения ON, что видно из следующей таблицы.

Спецификация привилегии	К какому уровню имеет отношение
`ON *.*`	Глобальные привилегии; все базы данных, все таблицы
`ON *`	Глобальные привилегии, если не была выбрана стандартная база данных, в противном случае – привилегии на уровне базы данных для текущей базы данных
`ON db_name.*`	Привилегии уровня базы данных; все таблицы в заданной базе данных
`ON db_name.tbl_name`	Привилегии уровня таблицы; все столбцы указанной таблицы
`ON tbl_name`	Привилегии уровня таблицы; все столбцы указанной таблицы стандартной базы данных

Наиболее мощными являются глобальные привилегии, применяемые ко всем базам данных. Так, например, чтобы предоставить пользователю `ethel` права суперпользователя, который имеет право делать все, в том числе и предоставлять привилегии другим пользователям, необходимо запустить следующий оператор:

```
GRANT ALL ON *.* TO 'ethel'@'localhost' IDENTIFIED BY 'coffee'
  WITH GRANT OPTION
```

Спецификатор `*.*` в предложении ON можно условно заменить фразой "все базы данных и все таблицы". Для повышения безопасности пользователю `ethel` разрешено подключаться только с одного локального компьютера. Ограничение компьютеров, с которых может подключаться к серверу пользователь, обладающий широкими правами, – весьма мудрое решение. В этом случае ограничивается и число компьютеров, с которых могут быть предприняты попытки взлома пароля.

Некоторые привилегии по своей природе являются административными и могут присваиваться с использованием только спецификатора глобальных привилегий `ON *.*`. Это такие привилегии, как FILE, PROCESS, RELOAD, SHUTDOWN и

другие, приведенные в табл. 11.1. Например, привилегия RELOAD позволяет использовать оператор FLUSH, таким образом, следующий оператор создает пользователя flush, который имеет право только на работу с операторами FLUSH:

```
GRANT RELOAD ON *.* TO 'flush'@'localhost' IDENTIFIED BY 'flushpass';
```

Этот тип учетной записи MySQL может понадобиться для создания административных сценариев, которые могут потребоваться для выполнения таких операций, как, скажем, обновление протоколов при замене файла протокола. (Подробнее об этом см. в разделе "Обновление файлов регистрации".)

Привилегии уровня баз данных относятся ко всем таблицам в указанной базе данных. Такие привилегии присваиваются предложением ON *db_name*.*:

```
GRANT ALL ON sampdb.* TO 'bill'@'racer.snake.net' IDENTIFIED BY 'rock';
GRANT SELECT ON menagerie.* TO 'reader'@'%' IDENTIFIED BY 'dirt';
```

Первый из указанных операторов предоставляет пользователю bill все права для работы со всеми таблицами базы данных sampdb. Второй оператор создает пользователя reader с ограниченными правами (только чтение), который может получать доступ к любой таблице базы данных menagerie, однако только для чтения. Другими словами, этот пользователь имеет возможность запускать только оператор SELECT.

При задании оператора GRANT можно задавать сразу несколько привилегий одновременно. Так, например, если необходимо предоставить пользователю возможность считывать и вносить изменения в содержимое существующих таблиц, но запретить создавать новые таблицы или удалять уже существующие, это можно сделать следующим образом:

```
GRANT SELECT, INSERT, DELETE, UPDATE ON sampdb.* TO 'jennie'@'%'
IDENTIFIED BY 'boron';
```

Для еще более детального управления доступом можно предоставлять привилегии отдельным таблицам или даже отдельным их столбцам. Привилегии по столбцам оказываются особенно полезными, если определенную часть таблицы необходимо скрыть от пользователя либо предоставить возможность изменения только заданных столбцов. Предположим, например, что в офис "Исторической Лиги" на должность секретаря нанимается сотрудник. Администратор решает предоставить новому сотруднику только права на чтение таблицы member, содержащей информацию о действующих членах общества, и привилегию UPDATE столбцу expiration (срок окончания членства) этой таблицы. При таком доступе новый секретарь вполне сможет изменять даты истечения членства организаций-участников, если они продолжают свое членство. Для создания такого пользователя MySQL можно использовать следующие операторы:

```
GRANT SELECT ON sampdb.member
  TO 'assistant'@'localhost' IDENTIFIED BY 'officehelp';
GRANT UPDATE (expiration) ON sampdb.member
  TO 'assistant'@'localhost';
```

Первый оператор предоставляет права на чтение таблицы member и определяет пароль. Второй оператор добавляет привилегию UPDATE, но только для столбца expiration. Поскольку пароль устанавливается первым оператором, во втором его определять повторно вовсе необязательно.

Если необходимо присвоить привилегии для работы с несколькими столбцами, их имена следует разделять запятыми. Так, например, чтобы добавить привилегии UPDATE для полей адресов таблицы member для пользователя assistant, используется приведенный ниже оператор. При его выполнении новые привилегии будут добавлены к уже существующим привилегиям пользователя:

```
GRANT UPDATE (street,city,state,zip) ON sampdb.member
   TO 'assistant'@'localhost';
```

Записи из таблиц разрешения при операциях переименования обычно не "наследуются". Например, любая привилегия, привязанная непосредственно к заданной таблице, после ее переименования к данной таблице относиться больше не будет. И этот принцип справедлив как для уровня баз данных, так и для таблиц и столбцов.

Чем полезна привилегия USAGE ("без привилегий")

Специальный спецификатор привилегий USAGE означает "нет привилегий". С первого взгляда такая возможность может показаться не очень полезной, но это не совсем так. Она позволяет изменять характеристики, отличные от тех, которые относятся к привилегиям, не затрагивая существующих характеристик. Для их использования предоставьте привилегию USAGE на глобальном уровне, укажите имя учетной записи и обеспечьте новые непривилегированные характеристики учетной записи. Например, если необходимо изменить пароль учетной записи с учетом того, что пользователь будет подключаться с использованием протокола SSL, или ввести ограничение на соединение для данной учетной записи, не затрагивая при этом привилегий, присвоенных данной учетной записи ранее, можно задать операторы следующего вида:

```
GRANT USAGE ON *.* TO account IDENTIFIED BY 'new_password';
GRANT USAGE ON *.* TO account REQUIRE SSL;
GRANT USAGE ON *.* TO account WITH MAX_CONNECTIONS_PER_HOUR 10;
```

Насколько необходимы безопасные соединения

Начиная с MySQL 4 защищенные соединения могут осуществляться с помощью протокола SSL (Secure Sockets Layer), который позволяет зашифровать обмен данными между клиентом и сервером и таким образом скрыть пересылаемые данные. Кроме того, для предоставления идентификационной информации по соединениям SSL может использоваться протокол X509. Безопасные соединения обеспечивают дополнительные средства защиты, но при этом затрачиваются дополнительные ресурсы центрального процессора, необходимые для выполнения операций шифрования и дешифрования. В данный момент протокол SSL поддерживается только ОС UNIX.

Для определения параметров безопасного соединения используется предложение REQUIRE. Для того чтобы объявить, что пользователь смог подключиться с использованием SSL, не определяя при этом тип безопасного соединения, используется предложение REQUIRE SSL:

```
GRANT ALL ON sampdb.* TO 'eladio'@'%.snake.net' IDENTIFIED BY 'flint'
   REQUIRE SSL;
```

Для более точного определения можно задать условие задания клиентом действительного сертификата X509:

```
GRANT ALL ON sampdb.* TO 'eladio'@'%.snake.net' IDENTIFIED BY 'flint'
   REQUIRE X509;
```

С целью еще более полной детализации предложение REQUIRE позволяет указать конкретные характеристики сертификата X509 или то, что соединение шифруется с использованием шифра определенного типа. Эти характеристики можно задавать в предложении REQUIRE параметрами ISSUER, SUBJECT или CIPHER. (Параметры ISSUER и SUBJECT задают информацию об организации, выдавшей сертификат, и принимающей стороне.) Например, каталог ssl дистрибутива sampdb включает файл сертификации клиента, client-cert.pem, который можно использовать при тестировании SSL-соединений. Эти значения сертификата могут отображаться следующим образом:

```
% openssl x509 -subject -noout -in client-cert.pem
issuer= /C=US/ST=WI/L=Madison/O=sampdb/OU=CA/CN=sampdb
subject= /C=US/ST=WI/L=Madison/O=sampdb/OU=client/CN=sampdb
```

Следующий оператор, GRANT, создает учетную запись, для которой клиент должен предоставить сертификат, соответствующий обоим этим значениям:

```
GRANT ALL ON sampdb.* TO 'eladio'@'%.snake.net' IDENTIFIED BY 'flint'
  REQUIRE ISSUER '/C=US/ST=WI/L=Madison/O=sampdb/OU=CA/CN=sampdb'
  AND SUBJECT '/C=US/ST=WI/L=Madison/O=sampdb/OU=client/CN=sampdb';
```

Для того чтобы явным образом отобразить, что защищенные соединения не требуются, используйте REQUIRE NONE. Параметр NONE появился в версии MySQL 4.0.4.

При использовании предложения REQUIRE необходимо учитывать следующие нюансы.

- Оператор GRANT, создающий учетную запись с использованием безопасного соединения, только накладывает ограничение на учетную запись. В действительности он не обеспечивает средств безопасного подключения клиента с помощью этой учетной записи. Для этого СУБД MySQL должна настраиваться включением поддержки протокола SSL. Как это можно сделать, описано в главе 12, "Резервное копирование и восстановление баз данных".
- Если сервер и клиентские программы настроены с поддержкой протокола SSL, любой пользователь сможет использовать безопасные соединения. Предложение REQUIRE используется только для того, чтобы показать, что производится безопасное соединение.
- Использование предложения REQUIRE для учетных записей, которые не подключаются к серверу через внешнюю сеть, имеет один недостаток. Такие соединения не могут прослушиваться, поэтому нет смысла делать их безопасными; кроме того, они создают дополнительную нагрузку. Учетная запись такого типа включает таких пользователей, которые подключаются только через файл разъема UNIX, именованный канал, или по IP-адресу 127.0.0.1 (интерфейс "обратной петли" компьютера). Эти соединения обрабатываются полностью внутри узла, их трафик попадает во внешнюю сеть.

Нужны ли пользователю административные привилегии

Администратор может предоставить владельцу базы данных возможность управления доступом, предоставив ему все привилегии базы данных и определив параметр WITH GRANT OPTION. Так, чтобы разрешить пользователю alicia

подключаться с любого компьютера домена big-corp.com и предоставить ему административные привилегии для работы со всеми таблицами базы данных sales, необходимо использовать оператор GRANT следующего вида:

```
GRANT ALL ON sales.*
   TO 'alicia'@'%.big-corp.com' IDENTIFIED BY 'applejuice'
   WITH GRANT OPTION;
```

Фактически предложение WITH GRANT OPTION позволяет администратору делегировать права предоставления доступа другому пользователю. Однако следует проявлять осторожность, поскольку два пользователя с привилегиями GRANT могут предоставлять другим пользователям свои права. Если предоставить одному пользователю только привилегию SELECT, а второму, помимо SELECT, привилегию GRANT, второй пользователь легко может сделать первого "более сильным". Другой способ присвоения привилегии GRANT OPTION заключается в указании ее в начале оператора GRANT:

```
GRANT GRANT OPTION ON sales.* TO 'alicia'@'%.big-corp.com';
```

Однако оператор такого вида не сработает:

```
GRANT ALL,GRANT OPTION ON sales.* TO 'alicia'@'%.big-corp.com';
```

ALL может использоваться только отдельно и не должен указываться вместе с другими спецификаторами привилегий.

Нужно ли накладывать ограничения на потребление ресурсов пользователем

Начиная с MySQL 4.0.2 появилась возможность ограничить количество подключений к серверу, которые пользователь может сделать за один час, и количество запросов или обновлений в час, которые пользователь должен сделать. Эти ограничения заданы в предложении WITH. Следующий оператор задает параметры пользователя spike, имеющего полный доступ к базе данных sampdb, но может подключаться десять раз в час и делать 200 запросов в час (по крайней мере 50 из которых могут быть обновлениями):

```
GRANT ALL ON sampdb.* TO 'spike'@'localhost' IDENTIFIED BY 'pyrite'
   WITH MAX_CONNECTIONS_PER_HOUR 10 MAX_QUERIES_PER_HOUR 200
   MAX_UPDATES_PER_HOUR 50;
```

Указание параметров управления ресурсами в предложении WITH не имеет никакого значения. Значение по умолчанию для каждого параметра равно 0, что значит "ограничений нет". Административный пользователь, имеющий привилегию RELOAD, может переустановить текущие значения счетчиков с помощью оператора FLUSH USER_RESOURCES. Оператор FLUSH PRIVILEGES так же делает это. После переустановки всех счетчиков, достигших предельного значения, учетные записи могут подключаться снова и делать запросы.

Отмена привилегий и удаление пользователей

Для отмены привилегий пользователя применяется оператор REVOKE. Его синтаксис очень похож на синтаксис оператора GRANT, с той лишь разницей, что предложение TO заменено на FROM, а предложения IDENTIFIED BY, REQUIRE и WITH GRANT OPTION отсутствуют вовсе:

```
REVOKE privileges (columns) ON what FROM account
```

Часть *account* этого оператора должна соответствовать части *account* исходного оператора GRANT для пользователя, привилегии которого отменяются. Часть *privileges* необязательно должна соответствовать ранее определенным привилегиям. Пользуясь оператором REVOKE, можно отменить только некоторые из привилегий, предоставленных оператором GRANT.

```
GRANT ALL ON sampdb.* TO 'boris'@'localhost' IDENTIFIED BY 'ruby';
REVOKE DELETE,UPDATE ON sampdb.* FROM 'boris'@'localhost';
```

Привилегия GRANT OPTION не включена в ALL, поэтому отменить ее можно только при явном ее упоминании в части *privileges* оператора REVOKE:

```
REVOKE GRANT OPTION ON sales.* FROM 'alicia'@'%.big-corp.com';
```

Оператор REVOKE применяется для отмены привилегий, но не для удаления пользователей. В таблице user все равно остается запись для пользователя, даже если все привилегии для него отменены. Это означает, что пользователь все еще имеет возможность подключаться к серверу. Для полного удаления пользователя необходимо явным образом удалить его запись из таблицы user. Для этих целей применяется оператор DELETE. Например, для удаления учетной записи, созданной для mary@%.snake.net:

```
% mysql -u root mysql
mysql> USE mysql
mysql> DELETE FROM user
    -> WHERE User = 'mary' and Host = '%.snake.net';
mysql> FLUSH PRIVILEGES;
```

Оператор DELETE удаляет запись пользователя, а оператор FLUSH указывает серверу перезагрузить таблицы разрешений. Таблицы перезагружаются автоматически при использовании операторов GRANT и REVOKE. Однако это не происходит при непосредственном изменении таблиц разрешений. (Проверьте также другие таблицы разрешения на наличие в них любых записей, относящихся к удаляемой учетной записи, и перед обновлением таблиц удалите любое упоминание, имеющее к ней отношение.)

Несколько парадоксально то, что с помощью оператора GRANT можно сделать сразу несколько операций отключения привилегий. Например, не существует синтаксиса оператора REVOKE, учитывающего возможность учетной записи осуществлять соединения с использованием SSL. Вместо этого можно воспользоваться оператором GRANT для предоставления привилегии USAGE на глобальном уровне (для того, чтобы текущие привилегии остались неизменными) и включить предложение REQUIRE NONE, свидетельствующее о том, что поддержка протокола SSL не требуется:

```
GRANT USAGE ON *.* TO account REQUIRE NONE;
```

Аналогично, при установке ограничения ресурсов для пользователя их нельзя удалить с помощью оператора REVOKE. Вместо этого для установки значений ограничения в 0 ("нет ограничений") воспользуйтесь оператором GRANT с указанием предложения USAGE:

```
GRANT USAGE ON *.* TO account
  WITH MAX_CONNECTIONS_PER_HOUR 0 MAX_QUERIES_PER_HOUR 0
  MAX_UPDATES_PER_HOUR 0;
```

Замена или обновление утерянных паролей

Один из способов внесения изменений или обновления пароля пользователя заключается в использовании оператора UPDATE, который задает значения для столбцов User и Host для записи таблицы user пользователя:

```
mysql> UPDATE user SET Password=PASSWORD('silicon')
    -> WHERE User='boris' AND Host='localhost';
mysql> FLUSH PRIVILEGES;
```

Значительно проще использовать оператор SET PASSWORD, так как учетная запись задается в том же формате, что и GRANT, и нет необходимости обновлять привилегии явным образом:

```
mysql> SET PASSWORD FOR 'boris'@'localhost' = PASSWORD('silicon');
```

Оператор SET PASSWORD безопаснее оператора UPDATE; пользуясь оператором UPDATE, проще сделать ошибку и внести изменения в учетную запись другого пользователя.

Другой способ изменения пароля заключается в использовании оператора GRANT USAGE с предложением IDENTIFIED BY, которым задается пароль. При этом не используется функция PASSWORD():

```
mysql> GRANT USAGE ON *.* TO 'boris'@'localhost' IDENTIFIED BY 'silicon';
```

Вероятна ситуация, когда необходимо восстановить утерянный пароль пользователя root. В этом случае невозможно подключиться к серверу как пользователь root, что является своего рода замкнутым кругом, так как изменять пароли может только пользователь root. А не зная пароля, потребуется остановить сервер и перезапустить без проверки таблицы разрешений. Как это сделать, см. выше, в разделе "Когда нельзя подключиться к серверу".

Ведение файлов журналов

В процессе запуска сервер MySQL исследует параметры командной строки и определяет, необходимо ли выполнять регистрацию. Если необходимо, сервер открывает соответствующие файлы журналов. Существует несколько основных типов журналов, которые может сгенерировать сервер по указанию администратора.

- **Журнал общих запросов.** Содержит информацию о подключениях клиентов, запросах и другого рода событиях. Этот файл исключительно полезен для отслеживания деятельности сервера: кто и откуда подключен, что делает. Это наиболее удобный протокол, который будет использоваться для определения того, какие запросы отправляют клиенты на сервер, и может быть полезен для решения возникающих проблем или отладки.
- **Журнал медленных запросов.** Назначение этого журнала заключается в помощи при идентификации элементов системы, которые необходимо изменить для повышения производительности. Сервер обрабатывает переменную long_query_time, которая задает параметры "медленных" запросов. Если для выполнения запроса требуется значительно больше времени, запрос считается медленным и протоколируется как таковой в журнал мед-

ленных запросов. Журнал медленных запросов также используется для протоколирования запросов, для которых индексы не используются.

- **Журнал обновлений.** В этом протоколе записываются запросы, которые модифицируют базу данных. Термин "обновление" (update) в этом контексте относится не только к операторам UPDATE, но и к операторам, которые модифицируют данные. По этой причине он содержит информацию о записях, таких как DELETE, INSERT, REPLACE, CREATE TABLE, DROP TABLE, GRANT и REVOKE. Содержимое журнала обновлений написано в операторах SQL в форме, которая используется в качестве ввода программы `mysql`. Первоначальная цель этого журнала заключалась в создании записи, которая использовалась для восстановления таблиц после их краха. (Базу данных можно восстановить из резервных файлов и последовательно перезапустить все запросы, которые модифицировали ее после резервного копирования, с использованием журналов обновления как вводимой информации в `mysql`. Таким образом, можно вернуть таблицы в состояние, в котором они были в момент аварии.) Но начиная с версии 3.23.14, когда был добавлен двоичный журнал, журнал обновления рассматривался как устаревшая возможность.
- **Двоичный журнал обновлений и индексный файл двоичного журнала обновления.** Содержимое двоичного журнала обновления аналогично содержимому журнала регистрации, но сохранение выполняется в более эффективном формате, кроме того, он предоставляет дополнительную информацию. Оно используется при восстановлении и для передачи обновлений на подчиненные серверы репликации. Двоичные журналы сопровождаются индексным файлом, в котором содержится перечень двоичных файлов регистрации, созданных на сервере.

  По умолчанию любой из этих регистрационных файлов размещается в каталоге данных, несмотря на то, что сервер не создаст ни одного из них, если ему не указать это отдельно. Любой журнал может быть запущен при определении параметра запуска для `mysqld`. Журналы, не являющиеся двоичными, записываются в формате ASCII и могут быть просмотрены непосредственно. Для просмотра содержимого двоичного журнала используется программа `mysqlbinlog`.

  Еще один файл регистрации, журнал регистрации ошибок, обрабатывается несколько иначе. (Например, в ОС UNIX он создается не сервером, а сценарием `mysqld_safe`.) Подробно о нем см. выше, в разделе "Журнал диагностических сообщений". Сервер также обрабатывает несколько журналов специального назначения, связанных с определенными дескрипторами таблиц. Журнал ISAM используется для отладки внесения изменений в записи таблиц типа ISAM и MyISAM, и о нем мы больше упоминать не будем.

  Дескрипторы таблиц типа BDB и InnoDB обрабатывают свои собственные журналы, предназначенные для внутренних целей (такие как выполнение автовосстановления после краха). Из всех этих журналов общий журнал запросов является наиболее полезным для наблюдения за сервером. Таким образом, при первом запуске с использованием MySQL, кроме всех других журналов, рекомендуется использовать общий журнал. После получения не-

которого опыта работы с MySQL может быть целесообразным отключить общий журнал, чтобы уменьшить потребление дискового пространства.

Для активизации процесса регистрации используются параметры, указанные в следующей таблице. Если имя файла регистрации является необязательным (как можно судить по квадратным скобкам) и не было указано, сервер использует стандартное имя и сохраняет такой файл в каталоге данных. Стандартное имя для регистрационных файлов состоит из имени компьютера, на котором установлен сервер, или *HOSTNAME*. При определении имени журнала с указанием относительного пути имя интерпретируется относительно каталога данных. Для размещения журнала в каком-либо другом каталоге необходимо указать полный путь к нему. Сервер создает любой несуществующий еще журнал, но не создаст каталог, в который должен записываться файл. По мере необходимости создайте такой каталог до создания сервера.

Параметр протоколирования	Какой журнал активизируется данным параметром
`--log[=file_name]`	Общий журнал регистрации
`--log-bin[=file_name]`	Двоичный журнал обновлений
`--log-bin-index=file_name`	Индекс двоичного журнала обновлений
`--log-update[=file_name]`	Журнал обновлений
`--log-slow-queries[=file_name]`	Журнал медленных запросов
`--log-isam[=file_name]`	Журнал регистрации ISAM/MyISAM
`--log-long-format`	Влияет на формат журналов регистрации медленных запросов и обновления

Если активизированы дескрипторы таблиц BDB или InnoDB, создаются их собственные журналы (по умолчанию в каталоге данных). Контролировать процесс создания журналов невозможно, но можно задать место их расположения с помощью следующих параметров.

Параметр протоколирования	*Назначение*
`--bdb-logdir=dir_name`	Каталог файла регистрации таблиц типа BDB
`--innodb_log_arch_dir=dir_name`	Каталог архивирования протоколов InnoDB
`--innodb_log_group_home_dir=dir_name`	Каталог регистрационного файла таблиц типа InnoDB

При задании одного из параметров InnoDB необходимо задавать и другой, присваивая им при этом равные значения.

Параметры регистрации могут быть заданы в командной строке при запуске `mysqld` или `mysqld_safe`. Эти параметры можно задать и в группе `[mysqld]` конфигурационного файла. О деталях параметрических групп, применимых к серверу и программам запуска сервера, см. в разделе "Определение параметров запуска".

Обновление журналов

Обновление журналов приводит к тому, что сервер закрывает и повторно открывает файлы регистрации. Этого можно добиться с помощью команды `mysqladmin flush-logs` или

оператора FLUSH LOGS (начиная с MySQL 3.22.9). Отправка сигнала SIGHUP на сервер также содействует обновлению журналов. (Другой способ обновления журналов заключается в использовании команды mysqladmin refresh, которая, кроме того, выполняет и другие функции, и использование этой команды может оказаться излишним, если требуется только обновление журналов.)

Процедура обновления журналов имеет отношение только к общему журналу, журналу обновлений, двоичному журналу обновлений и индексному файлу и журналу медленных обновлений и не имеет никакого отношения к журналу ошибок или журналам определенных дескрипторов файлов. Для двоичных журналов обновление журналов вызывает закрытие сервером текущего журнала и открытие нового со следующим порядковым номером. Это же происходит и с журналом обновления при создании пронумерованных последовательностей файлов журналов обновления.

Обновление журналов имеет смысл при ротации или истечении срока действия журналов, что подробно обсуждалось в разделе "Обновление файлов регистрации".

Общий журнал запросов

Этот журнал содержит запись о том, когда клиенты подключились к серверу, каждый отосланный клиентами запрос на сервер и другие события, которые представляются как запросы (такие как запуск и остановка сервера). При активизации общего журнала с помощью параметра `--log` без имени файла по умолчанию файлу общего журнала присваивается имя *HOSTNAME*`.log`, и он размещается в каталоге данных.

Запросы записываются в этом каталоге в порядке их поступления на сервер. Этот порядок может отличаться от порядка, в котором запросы завершают свое выполнение. В основном это происходит из-за чередования длинных и коротких запросов.

Журнал медленных запросов

Журнал медленных запросов хранит запись о запросах, на выполнение которых требуется много времени и которое задается в секундах в переменной `long_query_time`. В результате медленного запроса на сервере дается приращение счетчику `Slow_queries`. Журнал медленных запросов может пригодиться для оптимизации работы некоторых медленных запросов.

Однако при интерпретации содержимого этого журнала потребуется перенести основную нагрузку на учетную запись. Время выполнения запроса измеряется реальным временем, а не процессорным; таким образом, при перегруженном сервере, вероятнее всего, запрос будет оценен как медленный, даже если при других обстоятельствах он работает вполне удовлетворительно.

При активизации журнала регистрации медленных запросов `--log-slow-queries` без указания имени файла по умолчанию принимается имя файла *HOSTNAME*`-slow.log` из каталога данных. При одновременном задании параметров `--log-long-format` и `--log-slow-queries` сервер также регистрирует запросы, выполняющиеся без использования индексов.

Так как время, которое требуется запросу для выполнения, неизвестно до момента его завершения, запросы регистрируются в журнал медленных запросов после их выполнения, а не при их получении. Программа `mysqldumpslow` используется для просмотра запросов, содержащихся в журнале медленных запросов.

Журнал обновлений

Журнал обновлений используется для записи элементов, модифицирующих записи в таблицах INSERT, DELETE или UPDATE. Операторы SELECT в этот журнал не записываются. Однако оператор UPDATE, наподобие показанного ниже, не будет запротоколирован в журнале обновления, так как он в действительности не обновляет никаких значений:

```
UPDATE t SET i = i;
```

СУБД MySQL сначала выполняет операторы, чтобы определить, что она действительно что-то модифицирует, поэтому запросы записываются в журнал обновлений после завершения их выполнения, а не при их получении.

До СУБД MySQL 3.23.14 (начиная с этой версии появились двоичные журналы обновления) журнал обновления может быть использован в целях восстановления базы данных. Однако журнал обновлений был постепенно заменен двоичным журналом, который служит тем же целям и также поддерживает операции репликации. Протоколирование обновлений активизируется параметром `--log-update`. Сервер MySQL именует файлы обновления в соответствии со следующими правилами.

- При активизации журнала обновления заданием параметра `--log-update` без имени файла сервер генерирует последовательность регистрационных файлов в каталоге данных, имя которых состоит из имени вашего компьютера: *HOSTNAME*`.001`, *HOSTNAME*`.002` и т.д.[1]
- При указании имени регистрационного журнала без расширения сервер в качестве основного будет использовать это имя (а не имя узла) и генерировать цифровую последовательность для этих файлов. Например, при задании параметра `--log-update=update` он генерирует журналы обновления `update.001`, `update.002` и т.д.
- При активизации протоколирования обновлений и указании журнала регистраций, содержащего расширение, сервер всегда будет использовать для протоколирования этот файл, а не генерировать последовательность файлов регистрации.

Для журналов обновления, которые сгенерированы в пронумерованной последовательности, при запуске или обновлении журналов сервер создает следующий файл в последовательности.

Если параметр `--log-long-format` задается одновременно с параметром `--log-update`, сервер MySQL записывает в журнал дополнительную информацию о том, какой пользователь задал какой оператор и когда он это сделал.

---

[1] Есть планы использовать в номерах имен журналов шесть цифр, а не три. Это позволит улучшить сортировку имен журналов. На данный момент сортировка нарушается при переходе с .999 на .1000. Использование шести цифр позволит избежать этого нарушения порядка нумерации.

Двоичный журнал обновления и индексный файл двоичного журнала

Аналогично журналу обновления, двоичный журнал используется для протоколирования запросов, которые вносят изменения в данные, но его содержимое написано в более эффективном двоичном формате, чем формат ASCII. Двоичный файл содержит также и дополнительную информацию, такую как временные метки выполнения запросов. Двоичная природа этого журнала означает, что его нельзя просмотреть непосредственно, но для его просмотра можно воспользоваться программой `mysqlbinlog`.

Двоичный журнал обновлений предназначается для резервного копирования и восстановления базы данных, и вы сможете активизировать его, если вам необходимо использовать сервер в качестве главного сервера, который производит репликацию на подчиненный сервер.

При активизации журнала обновления заданием параметра `--log-bin` без имени файла сервер генерирует последовательность регистрационных файлов в каталоге данных, имена которых состоят из *HOSTNAME*`-bin`: *HOSTNAME*`-bin.001`, *HOSTNAME*`-bin.002` и т.д. В противном случае указанное имя используется в качестве основы (за исключением того, что при наличии расширения последнее отсекается). Следующий по порядку файл генерируется при каждом новом запуске сервера, обновлении журналов или тогда, когда текущий журнал достигает максимальных размеров. Этот размер задается значением переменной сервера `max_binlog_size`. (Заметим, что правила присвоения имен двоичным файлам регистрации аналогичны, правда, не на все 100%, правилам присвоения имен журналов обновления.)

Запросы записываются в двоичный журнал регистрации в порядке их выполнения, т.е. в том порядке, в котором они завершают свою обработку, а не по мере их получения, и это очень важно учитывать для соблюдения правильности выполнения репликации.

Запросы, являющиеся частью транзакции, кэшируются в процессе выполнения транзакции, а все операторы, принадлежащие данной транзакции, протоколируются после завершения выполнения транзакции. При откате транзакции она не записывается в двоичный журнал, поскольку такая транзакция не завершается никакими изменениями в базе данных. (Это аналогично тому, как отдельные запросы не протоколируются в журнал обновления до тех пор, пока они не изменят данные в действительности; для двоичного журнала обновления действуют те же принципы, но при условии распространения его действия в контексте всей транзакции.)

При активизации двоичного протоколирования сервер также создает сопровождающий его индексный файл, в котором дается перечень существующих файлов двоичного журнала. По умолчанию имя индексного файла совпадает с основным именем двоичного журнала, но имеет расширение `.index`. А для того, чтобы определить имя журнала явным образом, можно воспользоваться параметром `--log-bin-index`. Если имя не имеет расширений, расширение `.index` будет добавлено к имени автоматически. Например, при определении `--log-bin-index=binlog` индексный файл будет иметь имя `binlog.index`.

При использовании двоичных журналов для репликации убедитесь в том, чтобы ни один из двоичных файлов регистрации не удалялся до тех пор, пока его со-

держимое не будет реплицировано на один из имеющихся подчиненных серверов. После этого двоичный журнал больше не нужен. В разделе "Обновление файлов регистрации, задействованных при репликации" подробно описана процедура такой проверки.

Файлы регистрации и резервирование систем

Ваши журналы обновления или журналы двоичного обновления не смогут вам ничем помочь, если сбой работы диска приводит к их потере. Убедитесь в том, что осуществляется регулярное резервное копирование. Очень практично будет сохранить эти журналы не на диске, на котором хранятся базы данных. Это потребует их переноса из каталога данных, где сервер хранит их по умолчанию. Подробнее об этом см. в главе 10, "Каталог данных MySQL".

Журнал диагностических сообщений

Журнал диагностических сообщений используется для сохранения диагностической информации и сообщений об ошибках. Этот журнал обрабатывается в ОС UNIX и Windows по-разному. Особенностям их обработки и посвящены следующие разделы.

Журнал диагностических сообщений в ОС UNIX

В ОС UNIX журнал диагностических сообщений, в отличие от других журналов, создается не сервером, а сценарием `mysqld_safe`, который используется для запуска сервера. Сценарий `mysqld_safe` создает журнал ошибок, перенаправляя стандартный вывод сервера на стандартный вывод ошибок (выводные потоки, известные в языке программирования C как `stdout` и `stderr`). Имя стандартного журнала об ошибках — *HOSTNAME*`.err`. Другое имя журнала диагностических сообщений можно задать с помощью параметра `--err-log` сценарию `mysqld_safe` в командной строке или с помощью строки `err-log` в группе `[mysqld_safe]` конфигурационного файла. (До MySQL 4 сценарий `mysqld_safe` назывался `safe_mysqld`.) Сценарий `safe_mysqld` поддерживает параметр `--err-log` начиная с версии 3.23.22. До этого сценарий `safe_mysqld` всегда пишет журнал, пользуясь стандартным именем, и другого способа изменить его имя, кроме редактирования сценария, не существует.

Относительное имя пути к журналу регистрации ошибок интерпретируется относительно каталога, из которого вызывается сценарий `mysqld_safe`. Это контрастирует с поведением других регистрационных файлов, для которых относительные пути интерпретируются относительно каталога данных. Так как совсем не обязательно вызывать сценарий `mysqld_safe` из того же каталога (например, при его вызове вручную в различных ситуациях), рекомендуется задавать абсолютный путь, чтобы гарантировать, что журнал диагностических сообщений находится в одном и том же месте.

Заметим, что если файл регистрации ошибок уже существует, но пользователь, запустивший сервер, не имеет права для записи в этот файл, запуск завершится ошибочным выходом без какой-либо диагностики. Это может произойти в случае, если сервер запускался с различными значениями `--user` в разное время. Рекомендуется постоянно использовать одну и ту же учетную запись, как это уже упоминалось в разделе "Запуск сервера MySQL непривилегированным пользователем".

Журнал регистрации ошибок создается при запуске сервера с использованием сценария `mysql.server`, так как он в свою очередь вызывает сценарий `mysqld_safe`. Однако сценарий `mysql.server` не распознает параметр `--err-log` в командной строке или в группе `[mysql_server]` конфигурационного файла. Таким образом, если необходимо задать в таком случае имя журнала регистрации ошибок, это следует сделать в группе `[mysqld_safe]` конфигурационного файла.

При непосредственном запуске `mysqld` и отсутствии журнала регистрации ошибок сообщения об ошибках отправляются на терминал. Администратор сможет самостоятельно перенаправить вывод для перехвата диагностических сообщений. Например, для записи информации об ошибках в файл `/tmp/mysql.err` под управлением оболочки `csh` или `tcsh` сервер запускается следующим образом:

```
% mysqld >& /tmp/mysql.err &
```

или таким образом для оболочек `sh` и аналогичных оболочек:

```
% mysqld > /tmp/mysql.err 2>&1 &
```

Журнал диагностических сообщений в ОС Windows

Под управлением ОС Windows сервер по умолчанию записывает диагностические сообщения в файл `mysql.err` каталога данных. Другое имя этому файлу задать нельзя. При запуске сервера с параметром `--console` он выводит диагностику в консольное окно, не создавая при этом журнала регистрации ошибок. (Параметр `--console` не оказывает никакого влияния при запуске сервера как службы, так как в этом случае никакой консоли, на которую можно было бы выводить сообщения, нет.)

Обновление файлов регистрации

Одна из опасностей, которую таит в себе активизация всякого протоколирования, заключается в том, что процесс протоколирования может генерировать огромные объемы информации, которые могут заполнить собой все дисковое пространство. Это справедливо для случая сервера, сильно загруженного обработкой большого количества запросов. Для того чтобы сохранить несколько самых важных файлов, предупреждая безграничное разрастание файлов, можно прибегнуть к методам задания срока действия файлов. Ниже приведены некоторые из них.

- **Ротация журналов.** Этот метод относится к файлам, имеющим фиксированное имя, таким как журнал регистрации общих запросов и журнал регистрации медленных запросов.
- **Истечение срока действия файлов.** Этот метод удаляет файлы журналов регистрации, срок действия которых превышен. Этот метод применим к пронумерованным регистрационным файлам, таким как журналы обновления и двоичные журналы обновления.
- **Срок действия, имеющий отношение к репликации.** При использовании для репликации двоичных журналов обновления руководствоваться возрастом файлов для определения срока действия журнала не рекомендуется. В этом случае срок действия журнала можно считать истекшим только после

того, как он был реплицирован на все подчиненные серверы. Поэтому эта форма определения истечения срока действия журнала базируется на анализе использования двоичных журналов.

Ротацию журналов часто используют одновременно с обновлением журналов, чтобы убедиться в том, что вся имеющаяся в памяти информация была успешно записана на диск. Журналы могут быть сброшены с помощью команды `mysqladmin flush-logs` или оператора `FLUSH LOGS`.

Ниже в этом разделе описано применение различных методов истечения срока годности. Для любого из них необходимо рассматривать, как регистрационные файлы подходят к вашим методам резервирования базы данных. (Рекомендуется резервировать все регистрационные файлы, которые могут потребоваться для операций восстановления, поэтому не надо думать об истечении срока действия до тех пор, пока они не будут зарезервированы!) Примеры сценариев, которые описаны здесь, содержатся в каталоге `admin` дистрибутива `sampdb`.

Ротация журналов регистрации с фиксированными именами

Сервер MySQL сохраняет некоторые типы регистрируемой информации в файлах, имеющих фиксированные имена. Это имеет отношение к общему журналу регистрации и журналу регистрации медленных запросов, а также справедливо и для журналов обновления, если они не записываются в пронумерованную последовательность файлов. По истечении срока действия журналов используется ротация журналов. Это позволит обслуживать последние несколько файлов, но ограничит их количество настолько, насколько это позволит избежать переполнения вашего диска.

Ротация файлов регистрационных журналов осуществляется следующим образом. Предположим, что регистрационный файл имеет имя `log`. При первой ротации этот файл будет переименован в `log.1`, а серверу будет сообщено о том, что необходимо начать записывать информацию в новый файл. При второй ротации файл `log.1` переименовывается в `log.2`, `log` — в `log.1`, а серверу сообщается о том, что необходимо начать записывать информацию в новый файл. Таким образом, файл постепенно переименовывается в `log.1`, `log.2` и т.д., а когда этот файл достигает определенного уровня ротации, предыдущий файл заменяется новым. Например, если производится ежедневная ротация и требуется сохранить все журналы за неделю, необходимо хранить журналы от `log.1` до `log.7`. При каждой ротации из-за перезаписи файла `log.6` как файла `log.7` последний будет теряться.

Периодичность ротации журналов и количество старых журналов будет зависеть от степени загруженности сервера (активные серверы генерируют больше регистрационной информации) и от того, как много дискового пространства может быть выделено под старые журналы.

В ОС UNIX можно переименовать текущий регистрационный файл, пока сервер открыт. Обновление журналов приводит к тому, что сервер разрывает текущий файл и открывает новый, создавая тем самым новый регистрационный файл с оригинальным именем. Ниже приведен сценарий оболочки, который можно использовать при выполнении ротации регистрационных файлов с фиксированным именем:

```
#! /bin/sh
# rotate_fixed_logs.sh - ротация регистрационного
# журнала MySQL, имеющего фиксированное имя
# Аргумент № 1: имя регистрационного файла
if [ $# -ne 1 ]; then
  echo "Usage: $0 logname" 1>&2
  exit 1
fi
logfile=$1
mv $logfile.6 $logfile.7
mv $logfile.5 $logfile.6
mv $logfile.4 $logfile.5
mv $logfile.3 $logfile.4
mv $logfile.2 $logfile.3
mv $logfile.1 $logfile.2
mv $logfile $logfile.1
mysqladmin flush-logs
```

Этот сценарий принимает в качестве аргумента имя файла. Вы можете задать полный путь к файлу или заменить текущий каталог каталогом, в котором находится журнал, и ограничиться указанием только имени файла в этом каталоге. Например, для того, чтобы осуществлять ротацию журнала `log`, находящегося в каталоге `usr/mysql/data`, можно воспользоваться командой

```
% rotate_fixed_logs.sh /usr/mysql/data/log
```

или парой команд следующего вида:

```
% cd /usr/mysql/data
% rotate_fixed_logs.sh log
```

Этот сценарий рекомендуется запускать под управлением пользователя `mysqladm`, что гарантирует наличие прав на переименование файлов журналов. Заметим, что команда `mysqladmin` в этом сценарии не включает таких аргументов параметра соединения, как `-u` или `-p`. Если соответствующие параметры соединения для вызова `mysql` сохранены в конфигурационном файле `.my.cnf`, нет необходимости задавать их в сценарии командой `mysqladmin`. Если же конфигурационный файл не используется, команда `mysqladmin` должна "знать", как можно подключиться к серверу с помощью учетной записи MySQL, имеющей удовлетворительные привилегии для обновления журналов. Для этого может быть достаточно создать учетную запись, обладающую ограниченными привилегиями, позволяющими задавать только команды `FLUSH`. Это позволяет включить пароль данной учетной записи напрямую в сценарий, который доступен только для учетной записи `mysqladm`. Для этого данная учетная запись MySQL должна иметь только привилегию `RELOAD`. Например, для того, чтобы создать пользователя `flush` и присвоить пароль `flushpass`, воспользуемся оператором `GRANT`:

```
GRANT RELOAD ON *.* TO 'flush'@'localhost' IDENTIFIED BY 'flushpass';
```

После того как эта учетная запись будет создана, команда `mysqladmin` в сценарии `rotate_fixed_logs.sh` будет иметь следующий вид:

```
mysqladmin -u flush -pflushpass flush-logs
```

Подробнее о том, как можно производить периодическое обновление журналов, см. в разделе "Автоматизация процедуры обновления журналов". В ОС Linux можно прибегнуть к использованию программы `logrotate`, чтобы установить сценарий

mysql-log-rotate, который имеется в дистрибутиве MySQL, и не использовать сценарий rotate_fixed_logs.sh или создать свой собственный сценарий. Сценарий mysql-log-rotate можно найти в каталоге /usr/share/mysql RPM-дистрибутива, или в каталоге support-files инсталляции MySQL для двоичных дистрибутивов, или в каталоге share/mysql исходного дистрибутива MySQL.

В ОС Windows ротация журналов не работает так же, как и в ОС UNIX. При попытке переименовать регистрационный журнал, пока сервер его не открыл, возникает ошибка "file in use". Для того чтобы осуществить ротацию, сначала необходимо выключить сервер, переименовать файлы, а затем перезапустить сервер. Оставим остановку и перезапуск сервера на совести администратора, а собственно переименование регистрационного журнала может быть осуществлено с помощью пакетного файла такого вида:

```
@echo off
REM rotate_fixed_logs.bat - ротация регистрационного
REM журнала MySQL, имеющего фиксированное имя
if not "%1" == "" goto ROTATE
   @echo Usage: rotate_fixed_logs logname
   goto DONE
:ROTATE
set logfile=%1
erase %logfile%.7
rename %logfile%.6 %logfile%.7
rename %logfile%.5 %logfile%.6
rename %logfile%.4 %logfile%.5
rename %logfile%.3 %logfile%.4
rename %logfile%.2 %logfile%.3
rename %logfile%.1 %logfile%.2
rename %logfile% %logfile%.1
:DONE
```

Сценарий rotate_fixed_logs.bat вызывается аналогично сценарию оболочки rotate_fixed_logs.sh, с единственным аргументом, в котором дается имя регистрационного журнала, который будет обрабатываться в процессе ротации:

```
C:\> rotate_fixed_logs C:\mysql\data\log
```

или вот так:

```
C:\> cd \mysql\data
C:\> rotate_fixed_logs log
```

При первых нескольких циклах ротации журнала еще не будет полного набора журналов, поэтому журнал будет "жаловаться", что нет полного набора файлов журнала для ротации. Не обращайте на это внимание — это нормально.

Обновление пронумерованных журналов регистрации

Регистрационные файлы с фиксированным именем могут обновляться с использованием механизма ротации. Для пронумерованных регистрационных файлов, вроде тех, которые создаются для журналов обновления, механизм обновления выглядит немного по-другому. В таком случае обновлять файлы можно на основании анализа их возраста (времени последней модификации), а не механизма ротации в определенном диапазоне имен. Причина этого кроется в том, что нумерованные журналы не обязательно создаются в соответствии с фиксирован-

ным графиком, поэтому нельзя сказать, что все будет нормально, если сохранить последние *n* файлов. Сервер может получить несколько команд обновления журнала за короткий промежуток времени и вполне может оказаться так, что существует множество файлов, которые недостаточно устарели для того, чтобы их обновить.

Для упорядоченных журналов сервера сценарий обновления, базирующийся на времени существования, может выглядеть следующим образом:

```
#! /usr/bin/perl -w
# expire_numbered_logs.pl - просмотреть набор нумерованных регистрационных журналов MySQL
# и удалить те из них, время существования которых превышает неделю.
# Usage: expire_numbered_logs.pl logfile ...
use strict;
die "Usage: $0 logfile ...\n" if @ARGV == 0;
my $max_allowed_age = 7; # максимально допустимый срок в днях
foreach my $file (@ARGV) # проверка каждого аргумента
{
  unlink ($file) if -e $file && -M $file >= max_allowed_age;
}
exit (0);
```

Сценарий `expire_numbered_logs.pl` написан на языке Perl. Он работает как на платформе UNIX, так и Windows, поскольку Perl представляет собой язык написания сценариев, не зависящий от платформы. Для того чтобы его запустить, необходимо вызывать его с перечнем имен регистрационных файлов, которые являются кандидатами на обновление. Например, в UNIX можно сделать так:

```
% expire_numbered_logs.pl /usr/mysql/data/update.[0-9]*
```

или так:

```
% cd /usr/mysql/data
% expire_numbered_logs.pl update.[0-9]*
```

Необходимо помнить, что сценарий `expire_numbered_logs.pl` представляет определенную опасность, если его запускать без определенных аргументов. Например, его определенно нельзя запускать таким образом:

```
% cd /usr/mysql/data
% expire_numbered_logs.pl *
```

По этой команде будут удалены все файлы в каталоге данных, время существования которых превышает неделю.

Обновление файлов регистрации, задействованных при репликации

Сервер генерирует двоичные журналы обновления в пронумерованной последовательности. Как было сказано в предыдущем разделе, одним из методом их обработки является их обновление на основе анализа времени его существования. Однако при использовании для репликации двоичных файлов время существования не может служить достоверным свидетельством того, что журнал можно удалить. Напротив, двоичный журнал подлежит обновлению только после того, как его содержимое было реплицировано на все подчиненные серверы.

К сожалению, главный сервер не может "знать", сколько подчиненных серверов имеется у него в распоряжении и какие файлы были на них переданы. Главный сервер никогда не удалит двоичный журнал, пока он не будет разослан на активные подчиненные серверы, однако нет никакой гарантии, что подчиненный сервер подключен постоянно. Это значит, что администратор должен постоянно следить за тем, какие серверы действуют как подчиненные, и потом подключаться к ним и с помощью оператора SHOW SLAVE STATUS определить, какие двоичные регистрационные файлы главного сервера в данный момент обрабатывает подчиненный сервер. (Имя файла хранится в столбце Master_Log_File.) Любой двоичный журнал, который больше не используется подчиненными серверами, может быть удален. Предположим, что у нас есть следующий сценарий.

- Локальный сервер является главным и имеет два подчиненных сервера, S1 и S2.
- Двоичные регистрационные файлы, существующие в данный момент на главном сервере, имеют имена binlog.038–binlog.042.
- Оператор SHOW SLAVE STATUS на сервере S1 даст следующий результат:

  ```
  mysql> SHOW SLAVE STATUS\G
  ...
  Master_Log_File: binlog.41
  ...
  ```

  А на сервере S2 — следующий результат:

  ```
  mysql> SHOW SLAVE STATUS\G
  ...
  Master_Log_File: binlog.40
  ...
  ```

  В таком случае двоичным журналом с наименьшим номером по-прежнему является журнал binlog.40, востребованный подчиненным сервером, поэтому любой журнал с номером, меньшим, чем этот, можно удалить. Для этого подключитесь к главному серверу и задайте оператор следующего вида:

  ```
  mysql> PURGE MASTER LOGS TO 'binlog.040';
  ```

  Эта команда удалит с сервера все двоичные журналы, номера которых меньше, чем номер, указанный в команде. Таким образом, будут удалены файлы binlog.038 и binlog.039.

Оператор SHOW SLAVE STATUS появился в MySQL 3.23.22, а PURGE MASTER LOGS — в MySQL 3.23.28. Для работы с обоими операторами требуется привилегия SUPER (до MySQL 4.0.2 — привилегия PROCESS).

Автоматизация процедуры обновления журналов

Вызов сценариев обновления можно производить вручную, но если имеется возможность запрограммировать автоматическое выполнение команд, отпадет необходимость заботиться о их запуске самим. Одним из таких средств является утилита cron и настройка файла crontab, в котором можно запрограммировать автоматический график обновления. Если вы не знакомы с утилитой cron, о ней можно узнать в справочнике по UNIX, введя следующие команды:

```
% man cron
% man crontab
```

Для того чтобы подробнее узнать о формате файла crontab, можно также воспользоваться командой:

```
% man 5 crontab
```

Предположим, что планируется осуществить ротацию журнала общих запросов и обновить пронумерованные журналы обновления с помощью сценариев rotate_fixed_logs.sh и expire_numbered_logs.pl, которые находятся в каталоге /u/mysqladm/bin. Войдите в систему как mysqladm, а затем отредактируйте файл crontab, принадлежащий пользователю mysqladm, с помощью команды

```
% crontab -e
```

Эта команда позволяет отредактировать копию текущего файла crontab (который может оказаться пустым, если задания еще не устанавливались). Добавьте в этот файл следующие строки:

```
0 4 * * * /u/mysqladm/bin/rotate_fixed_logs.sh /usr/mysql/data/log
0 4 * * * /u/mysqladm/bin/expire_numbered_logs.pl
/usr/mysql/data/update.[0-9]*
```

Эти записи программируют работу утилиты cron для запуска обоих сценариев каждый день в 4 часа утра. При этом можно установить любое время запуска. Вероятно, что для более загруженного сервера, который генерирует массу протокольной информации, журналы нужно обновлять чаще, чем для сервера, который загружен не так сильно.

Для того чтобы запрограммировать регулярное выполнение обновления журнала (например, генерировать следующий пронумерованный журнал обновления или двоичный журнал обновления), в файл crontab можно добавить команду mysqladmin flush-logs. Для того чтобы утилита cron гарантированно находила программу mysqladmin, необходимо указать полный путь к нему.

Дополнительные вопросы настройки сервера

В этом разделе описывается настройка сервера в соответствии с пожеланиями администратора или для повышения производительности сервера:

- управление "прослушиванием" сервером сетевых интерфейсов на предмет подключения клиентов;
- включение и отключение возможности LOCAL для оператора LOAD DATA;
- вопросы интернационализации и локализации, такие как установка часового пояса сервера и поддерживаемых кодировок;
- включение и отключение дескрипторов специфических типов данных;
- настройка дескриптора таблиц типа InnoDB;
- настройка сервера с помощью внутренних переменных.

Как сервер "прослушивает" соединения

Сервер MySQL "прослушивает" соединения по нескольким сетевым интерфейсам, которыми можно управлять следующим образом.

- На всех платформах сервер "прослушивает" сетевой порт для TCP/IP-соединений, если он не был запущен с параметром --skip-networking.

По умолчанию сервер работает с портом 3306; для того чтобы задать другой номер, нужно воспользоваться параметром `--port`. Если узел сервера имеет больше одного IP-адреса, определить, какой из них будет прослушивать сервер MySQL, можно с помощью параметра `--bind-address`.

- На UNIX сервер "прослушивает" файл разъема домена UNIX на предмет соединений от локальных клиентов, которые подключаются к серверу через специальное имя узла `localhost`. По умолчанию файл разъема – `/tmp/mysql.sock`, а чтобы задать другое имя файла, воспользуйтесь параметром `--socket option`.
- На Windows NT серверы с суффиксом `-nt` в их именах включают поддержку именованных каналов. По умолчанию имя канала – MySql; для того чтобы задать другое название, используйте параметр `--socket`. До версии 3.23.50 поддержка именованных каналов была активизирована по умолчанию. После этого она автоматически отключена, и активизировать ее можно только с помощью параметра `--enable-named-pipe`.

При работе одного сервера он, как правило, использует сетевые установки, принятые по умолчанию. Если запущено больше одного сервера, необходимо убедиться, что каждый настроен на использование уникальных сетевых параметров. Подробнее об этом см. в разделе "Запуск нескольких серверов".

Вся предыдущая информация имеет отношение только к автономным серверам, работающим по схеме "клиент/сервер". Она никак не касается встроенных серверов, которые работают с клиентской программой, подключенной к серверу через внутренний канал и не "прослушивающей" никаких внешних сетевых интерфейсов.

Активизация или отключение возможности LOCAL для оператора LOAD DATA

Начиная с MySQL 3.23.49 возможность LOCAL для оператора LOAD DATA стала управляемой как во время создания, так и во время работы.

- Во время создания сервера возможность LOCAL может быть запущена или отключена с помощью параметра `--enable-local-infile` или `--disable-local-infile`.
- Во время работы сервера эта возможность может быть включена или отключена с помощью параметра `--local-infile` или `--disable-local-infile` на стороне сервера (до MySQL 4.0.2 она отключалась с помощью параметра `--local-infile=0`).

Если возможность LOCAL на сервере отключена, клиенты не могут использовать ее вообще. Если она включена, у клиентской библиотеки возможность LOCAL может быть по-прежнему отключена по умолчанию на клиентской части, но некоторые программы позволяют включать ее по запросу. Например, программа `mysql` поддерживает параметр `--local-infile`, разрешающий работать с возможностью LOCAL.

Проблемы интернационализации и локализации

Понятие "интернационализация" означает возможность использовать программное обеспечение в соответствии с локальным соглашением, принятым в

любой точке Земного шара. Понятие "локализация" относится к выбору определенного набора локальных соглашений из числа поддерживаемых кодировок. Вот перечень настроек СУБД MySQL, имеющих непосредственное отношение к интернационализации и локализации:

- часовой пояс;
- язык диагностических сообщений и сообщений об ошибках;
- текущая кодировка и кодировка по умолчанию.

Выбор часового пояса для сервера

Если сервер не может корректно определить часовой пояс, он будет сообщать время неправильно (например, в GMT – по Гринвичу). Чтобы исправить эту ситуацию в UNIX, пояс необходимо установить явным образом. Но следует заметить, что часовой пояс задается при запуске сервера с помощью стартового сценария `safe_mysqld` или `mysqld_safe`, а не для самого сервера `mysqld`.

Для определения часового пояса используется параметр `--timezone`. Этот параметр лучше всего задавать непосредственно в конфигурационном файле, особенно, если вызвать сценарий `safe_mysqld` или `mysqld_safe` из сценария `mysql.server`, который не поддерживает параметры в командной строке. Например, для определения Центрального часового пояса США в сценарии `mysqld_safe` в конфигурационный файл можно добавить такую строку:

```
[mysqld_safe]
timezone=US/Central
```

Это пример широко используемого синтаксиса (например, он работает для Solaris, Linux или Mac OS X). Вот еще один популярный формат определения часового пояса:

```
[mysqld_safe]
timezone=CST6CDT
```

В принципе, можно использовать наиболее подходящий для вашей системы синтаксис. До MySQL 4 сценарий `mysqld_safe` назывался `safe_mysqld`, который тоже поддерживал параметр `--timezone` вплоть до версии 3.23.28. Синтаксис командной строки аналогичен синтаксису, принятому для `mysqld_safe`, но при использовании конфигурационного файла поместите установку часового пояса в группу `[safe_mysqld]`. До MySQL 3.23.28 сценарий `mysqld_safe` не работает с параметром `--timezone`, поэтому, к сожалению, придется модифицировать сам сценарий `safe_mysqld`. Для этого необходимо вставить пару строк, задающих переменную среды TZ, до строки, которая запускает сервер. Например, можно добавить следующие строки:

```
TZ=U.S./Central
export TZ
```

или такие:

```
TZ=CST6CDT
export TZ
```

Выбор языка сообщения об ошибках

Сервер имеет возможность выводить диагностические сообщения и сообщения об ошибках на нескольких языках. По умолчанию принят `english`, но можно задать диагностику и на других языках. Для того чтобы выяснить, какие языки имеются в вашем распоряжении, просмотрите каталог `share/mysql` в инсталляции MySQL. Каталоги с определенными языками соответствуют имеющимся в наличии языкам. Для того чтобы изменить язык, прибегните к помощи параметра запуска `--language` с аргументом, задающим используемый язык или путь к каталогу этого языка. Например, для использования французского языка, если ваша инсталляция размещена в дереве `/usr/local/mysql`, можно задать `--language=french` или `--language=/usr/local/mysql/share/mysql/french`.

Настройка поддержки кодировки

Сейчас MySQL поддерживает любое количество кодировок. Выбор кодировки непосредственно влияет на то, какие символы допустимы в строках. Кроме того, он также влияет на порядок сортировки, который используется в операциях сравнения строк, и на символы, допустимые в именах таблиц и столбцов. Настоящий раздел посвящен настройке поддержки кодировок СУБД MySQL. Подробнее об использовании различных кодировок можно узнать из главы 12, "Безопасность".

Для того чтобы определить имеющиеся в наличии на вашем сервере кодировки, посмотрите текущие настройки инсталляционного каталога сервера, например каталог `share/mysql/charsets`. Индексный файл, имеющийся здесь, дает полный перечень кодировок, которые можно использовать в данный момент. Узнать имена также можно с помощью

```
mysql> SHOW VARIABLES LIKE 'character_sets';
```

Начиная с MySQL 4.1 появилась также возможность работать с оператором `SHOW CHARACTER SET` для получения списка кодировок и дополнительной информации о каждой кодировке.

Установить кодировку по умолчанию и другие кодировки, имеющиеся в наличии, можно при создании, задав параметры сценарию настройки.

- Стандартной кодировкой по умолчанию является `latin1`. Для установки другой кодировки по умолчанию воспользуйтесь параметром `--with-charset`.
- Для поддержки других кодировок можно воспользоваться параметром `--with-extra-charsets`. Аргументом к этому параметру является перечень названий кодировок, разделенный запятыми. Например, можно включить поддержку кодировок `latin1`, `big5` и `hebrew`:

  ```
  % ./configure --with-extra-charsets=latin1, big5, hebrew
  ```

Параметр `--with-extra-charsets` позволяет использовать две специальные групповые кодировки. Это кодировка `all`, которая включает все имеющиеся кодировки, и кодировка `complex`, содержащая все сложные кодировки. Кодировка является сложной, если она содержит многобитовую кодировку или поддерживает специальные правила сортировки.

При работе сервер использует свою стандартную, принятую по умолчанию кодировку, если не указано другое. Изменить кодировку можно с помощью параметра `--default-character-set` во время запуска сервера.

Несмотря на то что сервер поддерживает несколько кодировок, до появления версии 4.1 он мог поддерживать только одну кодировку. Начиная с MySQL 4.1 параметры, применяемые для управления кодировками при создании и работе, остались в силе, но появилась возможность управлять кодировками на уровне SQL для выбора кодировки на лету, на уровне сервера, базы данных, столбца и даже строковой константы. Другими словами, сервер получил возможность поддержки нескольких кодировок сразу.

Расширение поддержки кодировок повышает вероятность того, что пользователи захотят воспользоваться альтернативными кодировками, поэтому при создании сервера необходимо предусмотреть как можно большее количество кодировок. (Например, многие пользователи уже давно ожидали появление поддержки такой кодировки, как Unicode, поэтому ее включение скорее всего надо предусмотреть во время создания сервера.)

До MySQL 4.1 изменение стандартной кодировки сервера после создания таблиц приводило к тому, что требовалось обновление индексов для исправления порядка сортировки в соответствии с новой кодировкой. Для этого в таблицах типа MyISAM можно перегруппировать индексы с помощью программы `myisamchk`, задав параметры `--recover` и `--quick`, вместе с параметром `--set-character-set`, которым задается наименование используемой кодировки. При этом сервер должен быть отключен. Есть и другой способ, при котором не надо отключать сервер, — воспользоваться оператором `REPAIR TABLE ... QUICK` или командой `mysqlcheck --repair --quick`. Еще один путь решения этой проблемы, который применим не только к таблицам MyISAM, заключается в следующей последовательности операций: вывод дампа таблиц и удаление таблиц с последующим их восстановлением. Начиная с MySQL 4.1 улучшенная поддержка кодировки делает перестройку индекса при смене кодировки необязательной. Однако для того, чтобы воспользоваться новыми возможностями, необходимо перевести сервер на версию 4.1 (см. раздел "Преобразование старых таблиц для поддержки новых кодировок MySQL 4.1").

На стороне клиента можно задать кодировку для клиентской программы с помощью параметра `--default-character-set`. Если необходимая кодировка не предусмотрена инсталляцией MySQL, но в вашем распоряжении имеются все необходимые для сопровождения этой кодировки файлы, то все равно ими можно воспользоваться. Для этого необходимо задать параметр `--character-sets-dir` для клиентской программы.

Преобразование старых таблиц для поддержки новых кодировок MySQL 4.1

При переходе на версию MySQL 4.1 или выше рекомендуется преобразовать все таблицы в формат 4.1. Это позволит в полной мере воспользоваться новыми возможностями поддержки кодировок.

1. Создайте резервную копию баз данных с помощью `mysqldump`:

```
% mysqldump -p -u root --all-databases --opt > dumpfile.sql
```

Параметр `--all-databases` вызывает резервирование всех баз данных, а `-opt` оптимизирует размеры файла с резервной копией, что ускоряет процедуру восстановления. (Программа `mysqldump` подробно обсуждается в главе 13, "Резервное копирование и восстановление баз данных".)

2. Отключите сервер.
3. Обновите свою инсталляцию MySQL и перезапустите сервер, не изменяя при этом стандартную кодировку сервера.
4. Преобразуйте таблицу в формат 4.1, перезапустив ее из резервной копии:

```
% mysql -p -u root < dumpfile.sql
```

Эта процедура позволит серверу установить поддержку новой кодировки в самих таблицах, что имеет два существенных последствия.

- Кодировка может задаваться на уровне столбца. Поэтому при последующем изменении кодировки сервера кодировка самого столбца не затрагивается (т.е. столбцы изолируются от изменений кодировки сервера, следовательно, индексы, созданные на основании столбцов не при изменении кодировки сервера, не будут "испорчены").
- При последовательном изменении кодировок столбца сервер будет автоматически изменять все индексы, содержащие символы, влияющие на сортировочную последовательность новой кодировки.

Можно также поступить следующим образом: сначала обновить сервер, а потом преобразовать таблицы с помощью оператора ALTER TABLE, но этот способ труден и подвержен ошибкам. Предположим, что таблица объявлена следующим образом:

```
CREATE TABLE t
(
   c1 CHAR(10),
   c2 CHAR(10),
   c3 CHAR(10)
);
```

Для того чтобы преобразовать таблицы с явным объявлением кодировки, воспользуемся следующим оператором:

```
ALTER TABLE t
   MODIFY c1 CHAR(10) CHARACTER SET latin1,
   MODIFY c2 CHAR(10) CHARACTER SET latin1,
   MODIFY c3 CHAR(10) CHARACTER SET latin1;
```

Это очень трудоемкий процесс, так как он должен быть выполнен для всех таблиц. Метод дампа и перезагрузки значительно проще.

Выбор типов таблиц

СУБД MySQL поддерживает различные типы таблиц. Некоторые из них могут быть встроены или отсутствовать, а встроенные, в свою очередь, могут быть отключены во время запуска сервера.

- До MySQL 4 дескриптор ISAM был всегда встроенным, в последующих версиях его можно пропустить. Для этого при настройке используется параметр `--without-isam`.

Для встроенного сервера дескриптор ISAM не включается по умолчанию. Для того чтобы его включить, потребуется отредактировать файл `mysql_embed.h` в исходном дистрибутиве и перестроить сервер. Но лучше будет изменить тип таблиц ISAM на тип MyISAM и "забыть" о постоянных ссылках на формат хранения баз данных ISAM. В будущем поддержка этого формата будет отключена.

- Дескриптор типа BDB встраивается с помощью параметра `--with-berkeley-db`. Если он встроен, то его можно отключить во время запуска сервера параметром `--skip-bdb`.
- До MySQL 4 дескриптор InnoDB встраивается с помощью параметра `--with-innodb`. В последующих версиях тип InnoDB является встроенным по умолчанию, и его можно отключить параметром `--without-innodb`. В случае встроенного дескриптора InnoDB может быть отключен во время запуска сервера параметром `--skip-innodb`.
- Дескриптор MyISAM всегда встроен начиная с MySQL 3.23 и может быть не задан во время конфигурации или отключен во время запуска сервера.

Настройка табличного пространства InnoDB

Дескриптор таблиц InnoDB для хранения отдельных таблиц не использует отдельные файлы. Он обрабатывает все таблицы InnoDB в пределах единого табличного пространства, логически являющегося единым блоком хранения, которое дескриптор рассматривает как гигантскую структуру данных. (В определенном смысле, табличное пространство является виртуальной файловой системой.) Единственным уникально связанным с таблицей InnoDB является файл описания `.frm`, который хранится в каталоге баз данных, к которой принадлежит таблица.

Хотя табличное пространство InnoDB логически и является одной областью хранения, содержит один или несколько файлов на диске. Каждый компонент может являться обычным файлом или разделом диска.

В данном разделе описываются параметры настройки, которые используются для установки табличного пространства InnoDB и управления им. Эти параметры могут задаваться в командной строке запуска сервера, но так делают достаточно редко. Вместо этого табличное пространство можно настраивать, задав необходимые параметры в соответствующей группе конфигурационного файла (например, в группе `[mysqld]` или `[server]`). Это позволит использовать постоянные настройки при каждом запуске сервера. Самыми важными являются такие настройки:

- `innodb_data_home_dir` задает корневой каталог всех файлов, составляющих табличное пространство. Без этого параметра по умолчанию принимается стандартное имя каталога данных;
- `innodb_data_file_path` отражает спецификации файлов, составляющих табличное пространство, в каталоге InnoDB.

Значением этого параметра является список из одной и нескольких спецификаций, разделенных точками с запятой. Все спецификации состоят из имени файла, размера и, возможно, других параметров, разделенных двоеточиями. Общий размер компонентов табличного пространства должен быть как минимум 10 Мбайт.

В MySQL 3.23 необходимо задать значение innodb_data_file_path, в противном случае дескриптор InnoDB не запустится правильно. (Одним из последствий будет то, что сервер не запустится. Убедиться в том, что сбой запуска сервера связан с таблицами InnoDB, можно с помощью журнала регистрации ошибок.) В MySQL 4 сервер создает стандартное табличное пространство, состоящее из одного файла — ibdata1. Это табличное пространство составляет 64 Мбайт саморасширяющегося файла в MySQL 4.0.0 и 4.0.1 и 10 МБайт саморасширяющегося файла в последующих версиях.

В качестве простейшего примера предположим, что в каталоге данных требуется создать табличное пространство, состоящее из двух файлов, innodata1 и innodata2, размером 10 Мбайт.

Эти файлы настраиваются следующим образом:

```
innodb_data_file_path = innodata1:10M;innodata2:10M
```

Установка параметра innodb_data_home_dir в этом случае не требуется, так как его значением по умолчанию является каталог данных сервера, что нам требуется в данном случае.

Ниже приведены правила, в соответствии с которыми дескриптор InnoDB оперирует значениями параметров innodb_data_home_dir и innodb_data_file_path для определения путей к файлам табличного пространства.

- Если параметр innodb_data_home_dir имеет пустое значение, все спецификации в параметре innodb_data_file_path считаются абсолютным путем.
- Если параметр innodb_data_home_dir имеет непустое значение, он должен содержать имя каталога, в которым находятся все спецификации innodb_data_file_path. В этом случае все имена файлов интерпретируются относительно значения параметра innodb_data_home_dir.
- Если innodb_data_home_dir не определен, его значением по умолчанию будет значение каталога данных MySQL и имена файлов в innodb_data_file_path интерпретируются относительно каталога данных.

Опираясь на вышеуказанные правила, если каталогом данных является /var/mysql/data, следующие настройки определяют один и тот же набор файлов табличного пространства:

```
innodb_data_home_dir=
innodb_data_file_path=/var/mysql/data/ibdata1:10M;/var/mysql/data/ibda
ta2:10M
innodb_data_home_dir=/var/mysql/data
innodb_data_file_path=ibdata1:10M;ibdata2:10M
innodb_data_file_path=ibdata1:10M;ibdata2:10M
```

Значение innodb_data_file_path состоит из спецификаций файлов, разделенных точкой с запятой, а части спецификаций разделены двоеточием. Самый простой синтаксис спецификации файла состоит из имени файла и его размера, однако имеют место и другие формы синтаксиса:

```
path:size
path:size:autoextend
path:size:autoextend:max:maxsize
```

В первом формате задается файл фиксированного размера. Значение размера должно быть положительным целым числом с суффиксом M или G, обозначаю-

щими Мбайты и Гбайты соответственно. Второй формат определяет саморасширяющийся файл, т.е. после заполнения файла дескриптор InnoDB расширяет его сразу на 8 Мбайт. Третий формат аналогичен, но включает значение, отражающее максимальный размер, до которого саморасширяющийся файл может увеличиваться. Саморасширяющиеся компоненты табличного пространства могут использоваться начиная с MySQL 3.23.50, но только последний компонент табличного пространства может задаваться как саморасширяющийся.

Для первоначальной установки табличного пространства добавьте в конфигурационный файл соответствующие строки (предварительно убедившись в том, что ни один из составляющих файлов табличного пространства еще не существует) и запустите сервер. Дескриптор InnoDB заметит, что файл не существует, и приступит к его созданию и инициализации.

Начиная с MySQL 3.23.41 стало возможным использовать в качестве табличного пространства InnoDB целые разделы, благодаря чему можно создавать очень большие табличные пространства. Компонент, занимающий раздел, ограничен только размерами раздела, в то время как обычные файловые компоненты ограничены в размере максимальным размером файла, на который рассчитана операционная система. Кроме того, файлы, занимающие разделы, гарантированно занимают все непрерывное пространство на диске, в то время как обычные файлы подвергаются постоянной фрагментации. При инициализации табличного пространства дескриптор InnoDB минимизирует фрагментацию обычных файлов, сразу прописывая множество нулей в файлы, чтобы попытаться разместить их сразу, а не с постепенным приращением. Это позволяет уменьшить вероятность фрагментации, но не гарантирует полное ее отсутствие.

Включение раздела в табличное пространство является двухэтапной процедурой. Предположим, что у нас имеется раздел размером 2 Гбайт, имеющий путь `/dev/rdsk8`. В таком случае необходимо задать значение параметра `innodb_data_home_dir`, так как раздел не находится в каталоге данных. Настройте раздел следующим образом.

1. Первоначально настройте раздел со значением размера с суффиксом `newraw`. Это показывает, что файл является разделом, который и будет проинициализирован:

```
innodb_data_home_dir =
innodb_data_file_path = /dev/rdsk8:2Gnewraw
```

   Добавив эти строки в параметрическую группу `[mysqld]`, запустите сервер. Дескриптор InnoDB "заметит" этот суффикс и проинициализирует раздел. (Он также будет считать, что табличное пространство предназначено только для чтения, поскольку вы еще не выполнили второй шаг.) После инициализации раздела отключите сервер.

2. Внесите изменения в информацию о настройке, заменив суффикс `newraw` на `raw`:

```
innodb_data_home_dir =
innodb_data_file_path = /dev/rdsk8:2Graw
```

   После этого запустите сервер снова, и дескриптор InnoDB "увидит", что суффикс new уже отсутствует, поэтому он будет знать, что раздел был про-

инициализирован и теперь табличное пространство можно использовать в режиме "read/write".

Задавая раздел как часть табличного пространства InnoDB, необходимо убедиться в том, что у сервера есть право на операции чтения и записи в нем. Кроме того, обязательно убедитесь в том, что раздел больше не используется ни в каких других целях. В противном случае вы получите конкурирующие процессы, каждый из которых будет считать, что раздел принадлежит только ему, и будут использовать его по своему усмотрению, что в результате может привести к тому, что они будут накладывать свои данные друг на друга. Например, если вы первоначально зададите использование дескриптором swap-раздела, ваша система поведет себя совершенно непредсказуемо!

При настройке табличного пространства InnoDB в системах Windows обратные косые черты (backslases) имитируются одинарной косой чертой (/) или двойными обратными косыми (\\). Кроме того, спецификации отдельных файлов разделяются двоеточиями, даже если двоеточия присутствуют в именах файлов (полный путь в Windows начинается с буквы, обозначающей привод, и двоеточия). При обнаружении двоеточия дескриптор InnoDB разрешает неоднозначность просмотром следующего символа. Если это будет цифра, следующая часть спецификации считается размером табличного пространства. В противном случае он считается частью пути. Например, следующая настройка устанавливает табличное пространство на устройствах C и D размером 50 и 60 Мбайт:

```
innodb_data_home_dir =
innodb_data_file_path = C:/ibdata1:50M;D:/ibdata2:60M
```

При каждом запуске дескриптор InnoDB по необходимости создает файлы данных табличного пространства. Он также создает регистрационные файлы, если таковых не существует. По умолчанию эти журналы создаются в каталоге данных и начинаются с `ib_`. Заметим, что дескриптор InnoDB создает только файлы, не создавая каталоги. Все каталоги, необходимые для работы InnoDB, должны создаваться при запуске сервера. (Кроме того, с помощью параметров, описанных в разделе "Ведение файлов журналов", можно указать, в какие каталоги будут записываться эти журналы.)

Если запуск завершается неудачно при первоначальной установке табличного пространства, так как InnoDB не может создать некоторые необходимые файлы, проверьте журнал ошибок — это поможет определить причину. Удалив все файлы, созданные дескриптором InnoDB (за исключением разделов, которые тоже могут использоваться), исправьте ошибки настройки и перезапустите сервер.

После того как табличное пространство проинициализировано, изменить размер составляющих файлов уже нельзя. Однако эта проблема разрешима. Существует возможность добавить еще один файл в список существующих файлов, если существующее табличное пространство уже переполнено. Самым первым симптомом того, что табличное пространство близко к переполнению, является то, что транзакции, которые по всем признакам должны завершаться успешно, будут откатываться. Наличие свободного пространства можно проверить с помощью следующего оператора, где *tbl_name* — имя любой таблицы типа InnoDB:

```
mysql> SHOW TABLE STATUS LIKE 'tbl_name';
```

Чтобы увеличить размер табличного пространства за счет добавления еще одного компонента, воспользуйтесь следующей процедурой.

1. Остановите сервер.
2. Если последний компонент табличного пространства является самораширяющимся файлом, то, перед тем как добавить новый файл, в его спецификации необходимо указать, что он становится файлом фиксированного размера. Для этого необходимо определить текущий размер файла, затем округлить его до меньшего кратного 1 Мбайт числа (равного 1 048 576 байт, а не 1 000 000) и указать этот размер в спецификации файла. Предположим, что в нашем распоряжении есть файлы, имеющие следующую спецификацию:

   ```
   innodb_data_file_path = ibdata1:100M:autoextend
   ```

   Пусть реальный размер этого файла составляет 121 634 816 байт, т.е. 121 634 816 /1 048 576 = 116 Мбайт. Изменим спецификацию следующим образом:

   ```
   innodb_data_file_path = ibdata1:116M
   ```
3. Добавьте спецификацию нового компонента в конец текущего списка файлов. Если новый компонент является обычным файлом, предварительно убедитесь в том, что такой файл не был создан. Если компонент является разделом, для его добавления воспользуйтесь описанной выше двухэтапной процедурой для задания раздела в качестве части табличного пространства.
4. Перезапустите сервер.

Если необходимо перенастроить табличное пространство другим способом, отличным от добавления нового файла в конец списка, сначала создайте резервную копию, а затем воссоздайте новую настройку.

1. С помощью программы `mysqldump` скопируйте все таблицы типа InnoDB.
2. Остановите сервер и удалите все существующие файлы табличного пространства журнала InnoDB и файлы с расширением `.frm`, имеющие отношение к таблицам InnoDB.
3. Переинициализируйте табличное пространство в соответствии с новыми настройками.
4. Перезагрузите файл с резервной копией на сервер, чтобы воссоздать таблицы InnoDB.

Настройка сервера

На работу сервера MySQL влияет несколько основных параметров (переменных). Если установленные по умолчанию значения этих параметров не полностью соответствуют среде работы сервера, администратор может их изменить. Так, например, пользователи компьютеров с большим объемом памяти могут указать серверу, что необходимо увеличить буфера для операций с диском и индексами. В этом случае в память будет загружаться больше информации, что резко уменьшает число обращений к диску. Если же возможности системы ограничены, желательно указать, чтобы сервер использовал буфера меньших размеров, чтобы ресурсы системы не использовались неэффективно в ущерб другим процессам.

В следующих разделах обсуждаются проблемы установки и проверки переменных сервера и описываются некоторые переменные, которые непосредственно влияют на работу сервера в целом или более специфические для дескриптора таблицы InnoDB. Полный перечень переменных сервера задан в описании оператора `SHOW VARIABLES` (приложении Г, "Синтаксис SQL"). Вы также можете оптимизировать настройки сервера в *Справочном руководстве MySQL*.

Установка и проверка значений переменных сервера

Переменные сервера задаются во время его запуска. Кроме того, начиная с MySQL 4.0.3 многие из этих переменных могут изменяться динамически во время работы сервера.

Допустимый синтаксис установки переменной сервера во время запуска зависит от версии MySQL. Начиная с MySQL 4.0.2 можно рассматривать имя переменной как необязательное и устанавливать его непосредственно. Например, размер кэша таблиц управляется с помощью переменной `table_cache`. Для увеличения размера кэша до 128 можно воспользоваться следующим параметром из командной строки:

```
--table_cache=128
```

Кроме того, можно задать размер кэша через конфигурационный файл, пользуясь следующим синтаксисом:

```
[mysqld]
table_cache=128
```

Другая особенность синтаксиса, "переменная как параметр", заключается в том, что символы подчеркивания могут быть заменены на тире, что придает такой синтаксической конструкции совсем "параметризированный" вид:

```
--table-cache=128
```

Аналогично можно задать переменную в конфигурационном файле с использованием следующего синтаксиса:

```
[mysqld]
table-cache=128
```

Другой способ установки переменной заключается в использовании параметров `--set-variable` или `-O`, которые могут задаваться в командной строке в следующем виде:

```
--set-variable=table_cache=128
-O table_cache=128
```

В конфигурационном файле имеется возможность задавать только форму с "длинным" параметром:

```
[mysqld]
set-variable=table_cache=128
```

Если необходимо задать сразу несколько переменных, используйте для каждой из них отдельный параметр. До MySQL 4.0.2 имена переменных в качестве параметров рассматриваться не могут, и могут использоваться только форматы задания переменных `--set-variable` или `-O`. Начиная с 4.0.2 форматы `--set-variable` и `-O` по-прежнему поддерживаются, но уже являются устаревшими. Ка-

кой бы синтаксис вы ни использовали, все равно это лучше делать в конфигурационном файле. Это освобождает вас от необходимости помнить об установке переменных при каждом запуске сервера.

Переменные сервера до MySQL 4.0.3 могут устанавливаться только во время запуска, а их значения останутся неизменными на все время работы сервера. MySQL 4.0.3 добавляет два изменения в обработку переменных сервера.

- Настройка переменных может производиться динамически во время работы сервера. Это позволяет улучшить управление его работой и предотвратить остановку сервера при определенных обстоятельствах, когда в других условиях остановка будет неизбежна. (Например, можно поэкспериментировать с размерами буфера, чтобы увидеть, как это влияет на производительность сервера без остановки и перезапуска при каждом изменении.) Изменения, которые сделаны таким образом, не обязательно действуют до самого завершения работы сервера, но существует возможность задать значение переменной, которое будет отличаться от текущего стандартного значения. Кроме того, переменная может быть задана в конфигурационном файле, и это значение будет задаваться сервером при каждом запуске в будущем.
- Переменные задаются на двух уровнях — глобальном и сеансовом. Глобальные переменные влияют на работу в целом. Переменные сеансового уровня влияют только на поведение соединения данного клиента. Для переменных, которые существуют на обоих уровнях, глобальные переменные используются для инициализации соответствующих сеансовых переменных. Это происходит только в начале нового соединения; изменение глобальной переменной во время соединения не влияет на текущее значение соединения соответствующей сеансовой переменной.

Чтобы установить глобальную переменную *var_name*, используется оператор SET в одном из следующих видов:

```
SET GLOBAL var_name = value;
SET @@GLOBAL.var_name = value;
```

Для установки сеансовой переменной используется аналогичный формат:

```
SET SESSION var_name = value;
SET @@SESSION.var_name = value;
```

Если индикатор отсутствует, оператор SET модифицирует переменную сеансового уровня:

```
SET var_name = value;
SET @@var_name = value;
```

С помощью одного оператора SET можно установить сразу несколько переменных. При этом они разделяются запятыми:

```
SET SESSION sql_warnings = 0, GLOBAL table_type = InnoDB;
```

Во всех случаях, когда указание SESSION разрешено, его можно заменить синонимом LOCAL (это предусматривает использование @@LOCAL вместо @@SESSION).

Для установки глобальных переменных требуется привилегия SUPER. Установка сохраняется до нового изменения или выхода сервера. Для установки сеансовых переменных специальных привилегий не требуется. Такие установки со-

храняются до следующего изменения или завершения соединения. Для просмотра текущих установок переменных сервера используется оператор SHOW VARIABLES. Он предназначен для отображения всех переменных или только тех из них, имена которых соответствуют заданному шаблону SQL:

```
SHOW VARIABLES;
SHOW VARIABLES LIKE 'pat';
```

Начиная с 4.0.3 разрешены дополнительные форматы, позволяющие конкретно запрашивать значения глобальных и сеансовых переменных:

```
SHOW GLOBAL VARIABLES;
SHOW GLOBAL VARIABLES LIKE 'pat';
SHOW SESSION VARIABLES;
SHOW SESSION VARIABLES LIKE 'pat';
```

Когда отсутствует ключевое слово GLOBAL или SESSION, оператор возвращает значение сеансовой переменной, если она существует на этом уровне, а не возвращает глобальное значение. Из командной строки программа mysqladmin отображает текущие значения глобальных переменных сервера.

Информация об операторе SHOW VARIABLES, приведенная в приложении Г, "Синтаксис SQL", содержит перечень переменных, включая те из них, которые могут устанавливаться динамически, а также на каком уровне это можно сделать.

Переменные сервера общего назначения

Некоторые из переменных, которые чаще всего используются при общей настройке производительности, описаны ниже.

- back_log. Число запросов на подключение клиентов, размещаемых в очереди при обработке сервером запросов текущих клиентов. Администраторам весьма сильно загруженных узлов следует увеличить значение этой переменной.
- delayed_queue_size. Эта переменная определяет число строк оператора INSERT DELAYED, размещаемых в очереди. Если очередь заполнена, последующие операторы INSERT DELAYED блокируются до освобождения очереди. В результате клиенты временно не могут запускать эти операторы. Если подобного рода дополнение делают много клиентов и администратор замечает, что их запросы весьма часто блокируются, имеет смысл увеличить значение этой переменной, чтобы ускорить обслуживание клиентов. (Об операторе INSERT DELAYED см. в главе 4, "Оптимизация запросов".)
- flush_time. Если рабочая система подвержена систематическим сбоям и довольно часто перезагружается, установите значение этой переменной больше нуля. В этом случае сервер будет обновлять кэш таблиц каждые flush_time секунд. Подобная запись изменений таблиц негативно отразится на производительности системы, однако в то же время значительно снизит вероятность потери или повреждения данных таблицы.

  На компьютерах с ОС Windows можно запускать сервер с параметром командной строки --flush, в результате чего изменения таблицы будут сохраняться после каждого обновления.

- `key_buffer_size`. Эта переменная определяет размер буфера, используемого для хранения блоков индексов. Увеличение ее значения приведет к более быстрому созданию и изменению индексов. Чем больше значение этой переменной, тем выше вероятность того, что MySQL найдет ключевые значения в памяти. Следовательно, уменьшается и число обращений к диску, необходимое для обработки индексов.

  В версиях, предшествующих MySQL 3.23, эта переменная называется `key_buffer`. Системы MySQL версий 3.23 и выше распознают оба названия.

- `max_allowed_packet`. Максимальный размер буфера для обмена информацией с одним клиентом. Значение этой переменной рекомендуется увеличить, если клиенты в процессе работы отправляют большие значения BLOB и TEXT.

  В настоящее время для клиентов по умолчанию используется буфер размером 24 Мбайт. В более старых версиях размер буфера меньше, поэтому при использовании их, возможно, имеет смысл его увеличить. Так, например, для определения лимита в 24 Мбайт можно воспользоваться следующей командой:

  ```
  % mysql --set-variable=max_allowed_packet=64M
  ```

- `max_connections`. Максимальное количество одновременных клиентских соединений сервера. Администраторам сильно загруженных серверов следует увеличить значение этой переменной. Так, например, если сервер MySQL используется Web-сервером для обработки запросов DBI- и PHP-сценариев и объем Web-данных очень велик, установка низкого значения этого параметра приведет к отклонению запросов многих пользователей.

- `table_cache`. Эта переменная определяет размер кэша таблиц. Увеличение этого значения позволит `mysqld` держать открытыми больше таблиц, что уменьшает число операций открытия и закрытия файлов.

При увеличении значений переменных `max_connections` или `table_cache` рверу понадобится большее число дескрипторов файлов. С этим могут возникть проблемы, если в операционной системе установлены жесткие ограничения числу дескрипторов файлов на один процесс. Это ограничение необходимо бо увеличить, либо попытаться обойти каким-то другим способом. Способы еличения числа дескрипторов могут быть разными. Можно, например, при ильзовании сценария `mysqld_safe` воспользоваться параметром `--open-les-limit` для запуска сервера либо перенастроить сервер. Для подобной ройки в некоторых системах достаточно отредактировать системный файл исания и перезагрузиться. В других операционных системах для этого же ится редактировать файл описания ядра или перестраивать ядро. Как действоть в каждом конкретном случае, лучше узнать в справочном руководстве по ерационной системе.

Еще один способ обойти ограничение на число дескрипторов файлов – разбить галог данных на несколько каталогов и запустить несколько серверов. Число дерипторов файлов, доступных для нескольких запущенных серверов, значительно еличится. Такой подход, однако, чреват другими осложнениями. Так, пользовате не смогут получить доступ с одного сервера к базам данных разных каталогов, к как потребуется перенести одинаковые привилегии в таблицах разрешений

всех серверов для пользователей, которым необходим доступ сразу к нескольким серверам.

Существуют еще две переменные, значения которых администраторы иногда увеличивают в надежде на повышение производительности работы, – `read_buffer_size` и `sort_buffer_size`. (До MySQL 4.0.3 эти переменные назывались `record_buffer` и `sort_buffer`.) Эти буфера используются в процессе операций объединения и сортировки данных, и их значения определяют размер буфера для одного клиентского соединения. Таким образом, каждый клиент получает свой собственный буфер. Если значения этих переменных установить достаточно большими, производительность системы может снизиться из-за неэффективного распределения ресурсов. Чтобы настроить эти переменные, просмотрите сначала их текущие значения с помощью команды `mysqladmin variables`, а затем незначительно увеличьте (или уменьшите) их. В этом случае можно достичь максимального эффекта без серьезного снижения производительности.

Обязательно убедитесь в том, что вы используете достоверные условия тестирования. Эти буфера заполняются по мере необходимости, а не в момент подключения клиента. (Например, клиенту, который не использует в своей работе объединения, буфер объединения не понадобится.) Условия тестирования должны имитировать работу максимального количества пользователей одновременно и запускать такие сложные запросы, которые позволят определить реальные требования к памяти.

Переменные дескриптора InnoDB

Кроме переменных сервера общего назначения, сервер имеет несколько переменных, имеющих отношение к дескриптору InnoDB, когда активизирована поддержка таблиц типа InnoDB. Ниже описаны некоторые из них, которые используются для воздействия на работу дескриптора InnoDB.

- `innodb_buffer_pool_size`. При достаточном объеме памяти увеличение буфера InnoDB может снизить загруженность диска при доступе к табличным и индексным данным.
- `innodb_log_buffer_size`. Увеличение размера этой переменной позволяет сохранять в буфере транзакции большего размера, без необходимости сброса на диск части данных.
- `innodb_log_file_size`, `innodb_log_files_in_group`. Когда регистрационный журнал переполняется, InnoDB сохраняет буфер на диске. Увеличение размера файла регистрационного журнала уменьшает периодичность переполнения журнала и соответственно периодичность, с которой происходит обновление. (Издержкой этого метода является то, что чем больше размер регистрационного журнала, тем больше время восстановления после сбоя.) Можно изменить значение переменной `innodb_log_file_size` для изменения размера регистрационных журналов или значение переменной `innodb_log_files_in_group` для изменения количества файлов. Важной характеристикой является общий размер регистрационных журналов, который является произведением двух значений. Напомним, что общий размер регистрационных журналов не может превышать 4 Гбайт.

Запуск нескольких серверов

Большинство пользователей запускают на одной машине только один сервер MySQL, однако в некоторых случаях (они описаны ниже) возникает необходимость в запуске сразу нескольких серверов.

- Администратор желает протестировать новую версию сервера, прежде чем полностью передать ее в эксплуатацию. В этом случае запускается, как правило, несколько разных версий одного сервера.
- Операционные системы обычно накладывают ограничения по числу дескрипторов файлов на один процесс. Если увеличить это значение невозможно, обойдите данное ограничение посредством запуска сразу нескольких серверов. (Повышение лимита, например, может потребовать перекомпиляции ядра, но довольно часто администратор MySQL не является одновременно и администратором компьютера, что делает невозможным такую компиляцию.) В этом случае, как правило, запускается сразу несколько серверов одной версии.
- Провайдеры услуг Internet зачастую обеспечивают своих клиентов собственными инсталляциями MySQL, для которых необязательно выделять отдельные серверы. В таком случае достаточно запустить несколько экземпляров одной версии сервера или нескольких версий, если клиенты используют разные версии MySQL.

Существует еще ряд причин запуска сразу нескольких серверов. Например, в случае написания документации по MySQL часто приходится проверять работу версий сервера собственноручно, чтобы увидеть различия в их поведении. Автор тоже попадает в данную категорию, так как именно по этой причине у него установлено сразу несколько серверов (на данный момент больше 30). Однако одновременно у него запущено сразу несколько из них. Другие запускаются лишь изредка для проверки, поэтому необходимо иметь возможность простого запуска и остановки их в случае необходимости.

Общие проблемы одновременной эксплуатации нескольких серверов

Эксплуатация сразу нескольких серверов более сложна, чем запуск только одного сервера, так как появляется проблема предотвращения их взаимного влияния. Некоторые проблемы возникают во время установки СУБД MySQL. При использовании различных версий они могут быть размещены в разных каталогах. Для двоичных предварительно откомпилированных дистрибутивов этого можно добиться, распаковывая их в различных каталогах. Для дистрибутивов в исходных текстах, которые компилируются во время инсталляции, и для настройки различных каталогов установки каждого дистрибутива можно воспользоваться параметром `-prefix`.

Совсем другие проблемы возникают во время запуска серверов. Каждый процесс сервера должен иметь уникальные значения для нескольких параметров. Так, например, каждый сервер должен прослушивать поступающие соединения

на различных TCP/IP-портах. Это также справедливо и для случая работы различных программ или нескольких экземпляров одной программы. Аналогичная проблема возникает при активизации протоколирования. Каждый сервер должен вести протоколирование своего собственного набора журналов, так как запись в один журнал сразу несколькими серверами вызывает определенные проблемы.

Определение параметров сервера при его запуске обычно производится в его конфигурационном файле. Кроме того, параметры сервера можно задавать в процессе построения серверов. Эти настройки становятся его встроенными значениями по умолчанию, которые нет необходимости задавать явным образом при запуске.

При запуске нескольких серверов необходимо четко документировать все сделанные установки параметров. Одним из надежных способов сделать это является использование конфигурационных файлов для определения значений параметров. (Это может пригодиться даже для серверов, имеющих уникальные значения параметров, установленные при компиляции, поскольку конфигурационные файлы служат своеобразной формой явной документации.)

Остановимся на нескольких формах параметров, которые могут стать потенциальным источником конфликта, если они не установлены по серверному принципу. Заметим, что некоторые из этих параметров влияют на другие параметры, поэтому не надо устанавливать их все явным образом для каждого сервера. К примеру, каждый сервер во время работы использует уникальный идентификатор процесса. Но для PID-файла каталог данных является каталогом размещения по умолчанию, таким образом, каждый сервер имеет различные каталоги данных, что неявно приводит к созданию различных стандартных PID-файлов.

- При работе различных серверных версий совершенно естественно, что каждый дистрибутив будет установлен в разных основных каталогах. Кроме того, неплохо, чтобы они использовали разные каталоги данных[2]. Для того чтобы задать параметры явным образом, можно воспользоваться следующими параметрами.

Параметр	Назначение
`--basedir=dir_name`	Путь к корневому каталогу инсталляции MySQL
`--datadir=dir_name`	Путь к каталогу данных MySQL

Во многих случаях каталог данных будет подкаталогом основного каталога, но это правило выполняется не всегда. Например, провайдер может предоставить пользователям общий каталог для инсталляции СУБД MySQL (т.е. аналогичное множество клиентов и исполняемых модулей сервера), запуская при этом одинаковые экземпляры для сервера, каждый из которых работает с определенным каталогом данных. В таком случае основной каталог для всех серверов может совпадать, но отдельные каталоги данных могут размещаться в другом месте, например в корневых каталогах пользователей.

Следующие параметры должны иметь различные значения для каждого сервера, чтобы держать друг друга "на расстоянии".

[2] Иногда допустимо, чтобы различные серверы использовали один и тот же каталог, но я не рекомендую делать это.

Параметр	Назначение
`--port=`*port_num*	Номер порта для соединений TCP/IP
`--socket=`*file_name*	Путь к файлу доменного разъема для UNIX
`--pid-file=`*file_name*	Путь к файлу, в который сервер записывает идентификатор своего процесса

- При активизации протоколирования имена используемых журналов для каждого сервера должны различаться. В противном случае может возникнуть ситуация, когда сразу несколько серверов запротоколируют свои действия в один и тот же файл. С одной стороны, такое положение вещей может дать неожиданные результаты, но с другой стороны, неправильно будет работать такая процедура, как репликация. В случае указания относительных путей регистрационные журналы, указанные в параметрах следующей таблицы, создаются в каталоге данных сервера. Если каждый сервер использует различные каталоги данных, для того, чтобы производить протоколирование в различные наборы файлов, нет необходимости задавать абсолютные пути. (Более подробно о присвоении имен журналам регистрации см. в разделе "Ведение файлов журналов".)

Параметр протоколирования	Журнал, активизированный данным параметром
`--log[=`*file_name*`]`	Общий регистрационный файл
`--log-bin[=`*file_name*`]`	Двоичный файл регистрации обновления
`--log-update[=`*file_name*`]`	Файл регистрации обновления
`--log-bin-index=`*file_name*	Индексный файл двоичного журнала регистрации обновления
`--log-slow-queries[=`*file_name*`]`	Файл регистрации медленных запросов
`--log-isam[=`*file_name*`]`	Файл регистрации для таблиц типа ISAM/MyISAM

- Активизация таблиц типа BDB или InnoDB предполагает, что каталоги, в которые сервер производит запись своих журналов, должны быть уникальными для каждого сервера. По умолчанию сервер производит запись этих журналов в каталог данных. Для того чтобы изменить размещение, необходимо задать следующие параметры.

Параметр протоколирования	Назначение
`--bdb-logdir=`*dir_name*	Каталог журнального файла BDB
`--innodb_log_arch_dir=`*dir_name*	Каталог архива журналов регистрации InnoDB
`--innodb_log_group_home_dir=`*dir_name*	Каталог журнального файла InnoDB

Если задается один из параметров, связанных с InnoDB, необходимо задать сразу оба и установить для них одно и то же значение.

- Если под управлением ОС UNIX для запуска серверов используется `mysql_safe`, создается журнал регистрации ошибок (по умолчанию в каталоге данных). Задать имя журнала регистрации ошибок можно с помощью параметров `--err-log=`*file_name*. Заметим, что этот параметр должен

задаваться только для сценария mysqld_safe, а не mysqld, и что относительные пути интерпретируются относительно каталога, из которого был запущен сценарий mysqld_safe, а не относительно каталога данных, как для других регистрационных файлов. При использовании этого параметра задавайте *абсолютный путь*. Это гарантирует, что журнал регистрации ошибок всегда будет создаваться в одном и том же месте.

- При работе под управлением ОС UNIX может также потребоваться задать параметр --user для каждого сервера, чтобы указать, какая учетная запись будет использоваться для запуска каждого сервера. Вероятен вариант, когда для различных пользователей будут работать отдельные экземпляры MySQL, каждый из которых "держит" отдельный каталог данных.
- Под управлением ОС Windows различные серверы, установленные как службы, должны использовать уникальное имя службы.

Инсталляция и настройка нескольких серверов

Инсталлируемые на одном компьютере разные версии одного сервера следует размещать в различных каталогах. Наиболее простой способ отделить дистрибутив при инсталляции из исходного кода — воспользоваться параметром --prefix при запуске команды configure в процессе инсталляции каждой версии MySQL. Это приведет к инсталляции всех файлов в одном каталоге, который можно разделить по номерам версий. В данном разделе описывается один из способов, как можно это сделать, применив определенные соглашения по настройке, которые используются автором для разделения инсталляций MySQL.

В используемом автором размещении все инсталляции СУБД MySQL размещаются в одном общем каталоге: /var/mysql. Для того чтобы установить заданный дистрибутив, такая инсталляция производится в подкаталог, отражающий номер версии дистрибутива, каталога /var/mysql. Например, в качестве основного каталога версии MySQL 4.0.5 используется /var/mysql/40005. Это можно сделать с помощью настройки с параметром --prefix=/var/mysql/40005. Для задания специфических для серверов дополнительных значений, таких как номер порта TCP/IP и путь к файлу разъема, можно воспользоваться и другими параметрами. Используемые настройки делают номер порта TCP/IP равным номеру версии, размещают файл разъема непосредственно в базовом каталоге и присваивают каталогу данных имя data.

Для установки всех этих настроек автор использует сценарий configver, аналогичный следующему (обратите внимание, что параметром, задающим каталог данных, является --localstatedir, а не --datadir):

```
VERSION="40005"
PREFIX="/var/mysql/$VERSION"
# InnoDB включается по умолчанию начиная с MySQL 4:
# - до 4.x InnoDB включается с --with-innodb
# - начиная с 4.x InnoDB исключается из --without-innodb
HANDLERS="--with-berkeley-db"
OTHER="--enable-local-infile --with-embedded-server"
rm -f config.cache
./configure \
--prefix=$PREFIX \
```

```
--localstatedir=$PREFIX/data \
--with-unix-socket-path=$PREFIX/mysql.sock \
--with-tcp-port=$VERSION \
$HANDLERS $OTHER
```

Первая строка задает соответствующий номер версии и модифицирует соответствующим образом другие значения, в соответствии с тем, будут ли использоваться дескрипторы таблиц InnoDB и BDB, поддержка LOCAL для оператора LOAD DATA и т.д. После этого следующие команды настраивают, компилируют и устанавливают дистрибутив:

```
% sh config-ver
% make
% make install
```

После этого меняется базовый каталог инсталляции и инициализируются ее каталог данных и таблицы разрешения:

```
% cd /var/mysql/40005
% ./bin/mysql_install_db
```

Затем наступает очередь процедуры обеспечения безопасности инсталляции MySQL, вкратце описанной выше в разделе "Настройка процедур запуска и завершения работы сервера MySQL", и более детально описанной в главе 12, "Безопасность".

После этого все, что остается сделать, — это задать необходимые параметры в конфигурационном файле и провести подготовительные мероприятия для запуска. Один из способов, как это сделать, обсуждается в разделе "Использование сценария mysqld_multi для управления несколькими серверами" в этой главе

Стратегии определения параметров запуска

После того как серверы установлены, каким образом можно запустить их с соответствующим набором рабочих параметров, которые необходимы для каждого? Существует насколько вариантов процедуры запуска.

- При работе с различными серверами, построенными самостоятельно, параметры каждого сервера могут быть настроены по умолчанию, поэтому при запуске не надо задавать никаких параметров. Этому методу присущи определенные недостатки, поскольку в таком случае неясно, с каким параметрами запущен тот или иной сервер.
- Активизировать параметры во время запуска сервера можно в командной строке или в конфигурационном файле. Для задания большого числа параметров лучше всего воспользоваться конфигурационными файлами. Но дело в том, что каждый сервер должен пользоваться только своей конфигурационной информацией. Ниже приведены варианты решения этой задачи.
  - Использование параметра `--defaults-file` для явного определения файла, с которым будет работать сервер, и определения различных файлов для разных серверов. Это позволяет поместить все параметры отдельных серверов в соответствующие файлы, которые будут считываться в момент загрузки. Заметьте, что выбор этого варианта не требует использования таких глобальных конфигурационных файлов, как `/etc/my.cnf`.

- Разместите общие для всех серверов параметры в глобальном конфигурационном файле `/etc/my.cnf` и с помощью параметра `--defaults-extra-file` в командной строке задайте файл, который содержит только параметры, специфические для данного сервера. Например, воспользуемся группой `[mysqld]` в файле `/etc/my.cnf`, чтобы определить параметры, которые применимы ко всем серверам. Они не должны повторяться в конфигурационных файлах, отвечающих за отдельные серверы.

  Обязательно необходимо убедиться в том, что любой параметр, размещенный в общей параметрической группе, понятен всем серверам, с которыми вы работаете. Например, нельзя использовать параметр `local-infile` для того, чтобы активизировать использование `LOAD DATA LOCAL`, если есть хотя бы один сервер версии ниже 3.23.49, так как этот параметр появился только в этой версии. Его присутствие в общей параметрической группе может стать причиной ошибки для более старых серверов.
- Серверы ищут конфигурационный файл `my.cnf` в созданном во время построения сервера каталоге данных. Если каждый сервер имеет различные пути к каталогам данных, файлы `my.cnf` можно использовать для задания параметров, специфических для соответствующих серверов. Другими словами, конфигурационный файл `/etc/my.cnf` можно использовать для общих установок. Они будут использоваться всеми установленными серверами. И использовать конфигурационный файл *DATADIR*`/my.cnf` для установок, специфических для данного сервера, где *DATADIR* соответствует используемой версии сервера. (Заметим, что эта стратегия не действует, если расположение каталога данных задается во время запуска и если вы запустили несколько экземпляров некомпилируемого сервера.)
- Для запуска нескольких серверов используется сценарий `mysqld_multi`. Он позволяет задавать параметры для всех серверов в единственном файле, связав каждый сервер с конкретной, только с ним связанной параметрической группой.
- Под управлением ОС Windows можно запустить несколько служб, пользуясь специальными соглашениями по присвоению имен группам в конфигурационных файлах, характерными для этого стиля запуска сервера.

В следующих разделах показаны способы реализации этих стратегий и демонстрируется, как можно использовать сценарий `mysqld_multi` и запустить несколько серверов под управлением ОС Windows.

Использование сценария `mysqld_multi` для управления несколькими серверами

Сценарии `mysqld_safe` и `mysql.server`, которые обычно используются для запуска сервера под управлением ОС UNIX, гарантированно работают в конфигурации с одним сервером. Для того чтобы упростить работу с несколькими серверами, вместо них лучше использовать сценарий `mysqld_multi`.

Сценарий `mysqld_multi` работает на основании того, что настройке каждого сервера присваивается уникальный номер, которым обозначена настройка кон-

кретного сервера. Они заключаются в соответствующие группы конфигурационного файла [mysqldn], где n — число. Конфигурационный файл может также содержать группу [mysqld_multi], в которой задаются параметры для самого сценария mysqld_multi. Например, если были инсталлированы серверы MySQL 3.23.51, 4.0.5 и 4.1.0, их параметрические группы можно обозначить как [mysqld32351], [mysqld40005] и [mysqld40100] и установить параметры в файле /etc/my.cnf следующим образом:

```
[mysqld32351]
basedir=/var/mysql/32351
datadir=/var/mysql/32351/data
mysqld=/var/mysql/32351/bin/mysqld_safe
socket=/var/mysql/32351/mysql.sock
port=32351
local-infile=1
user=mysqladm
log=log
log-update=update-log
innodb_data_file_path = ibdata1:10M
[mysqld40005]
basedir=/var/mysql/40005
datadir=/var/mysql/40005/data
mysqld=/var/mysql/40005/bin/mysqld_safe
socket=/var/mysql/40005/mysql.sock
port=40005
local-infile=1
user=mysqladm
log=log
log-bin=binlog
innodb_data_file_path = ibdata1:10M:autoextend
[mysqld40100]
basedir=/var/mysql/40100
datadir=/var/mysql/40100/data
mysqld=/var/mysql/40100/bin/mysqld_safe
socket=/var/mysql/40100/mysql.sock
port=40100
local-infile=1
user=mysqladm
log=log
log-bin=binlog
skip-innodb
skip-bdb
language=french
default-character-set=utf8
```

Параметры размещения, заданные в этом конфигурационном файле для каждого сервера, соответствуют конфигурации каталогов, описанных ранее в этой главе в разделе "Инсталляция и настройка нескольких серверов". Кроме того, были заданы специфические для каждого сервера параметры, которые соответствуют различиям в типах регистрационных журналов, дескрипторов таблиц и т.д.

Для того чтобы запустить данный сервер, запустите сценарий mysqld_multi с командой start и номером параметрической группы сервера в командной строке:

```
% mysqld_multi --no-log start 32351
```

Параметр --no-log приводит к тому, что сообщения о состоянии передаются на терминал, а не в регистрационный журнал. Это упрощает информирование о состоя-

нии дел. Можно также задать больше одного сервера, дав перечень их номеров групп, разделенных запятыми в виде списка. Диапазон номеров серверов можно задать с помощью тире. Однако в перечне серверов не должно быть пробела:

```
% mysqld_multi --no-log start 32351,40005-40100
```

Для остановки сервера или получения отчета о состоянии, показывающего, работают ли они, воспользуйтесь командой `stop` или `report`, указав список серверов. Для этих команд сценарий `mysqld_multi` для обмена с серверами вызовет `mysqladmin`, таким образом, потребуется также указать имя пользователя и пароль для административной учетной записи:

```
% mysqld_multi --nolog --user=root --password=rootpass stop 32351
% mysqld_multi --nolog --user=root --password=rootpass report 32351,40100
```

Имя пользователя и пароль должны быть применимы ко всем серверам, которые будут управляться данной командой. Сценарий `mysqld_multi` пытается автоматически определить расположение `mysqladmin`, или путь можно задать явным образом в группе `[mysqld_multi]` конфигурационного файла. Кроме того,



нистратора, и его пароль для команд `stop` и `report`. Например:

```
[mysqld_multi]
mysqladmin=/usr/local/mysql/bin/mysqladmin
user=leeloo
password=multipass
```

Задав имя администратора и его пароль в файле, обязательно убедитесь в том, что он не будет доступен для всех!

Эксплуатация нескольких серверов в ОС Windows

Существует несколько способов запуска нескольких серверов под управлением ОС Windows. Один из них заключается в запуске серверов вручную, а другой — в использовании нескольких служб. Для запуска нескольких серверов вручную создайте конфигурационный файл, в котором дан перечень индивидуальных параметров для каждого в отдельности. Например, для того, чтобы работать с двумя серверами, которые используют одну и ту же программу, но разные каталоги данных, можно создать два конфигурационных файла, которые выглядят следующим образом:

файл `C:\my.cnf1`:

```
[mysqld]
basedir=C:/mysql
datadir=C:/mysql/data
port=3306
```

файл `C:\my.cnf2`:

```
[mysqld]
basedir=C:/mysql
datadir=C:/mysql/data2
port=3307
```

При запуске серверов из командной строки воспользуйтесь параметром `--defaults-file`, чтобы каждый из них читал свой собственный конфигурационный файл:

```
C:\> mysqld --defaults-file=C:\my.cnf1
C:\> mysqld --defaults-file=C:\my.cnf2
```

Клиенты подключаются, указывая номер порта, соответствующий серверу, к которому они планируют подключиться. То же самое касается использования `mysqladmin` для отключения серверов. Первая из следующих команд использует стандартный порт (3306), а вторая задает порт 3307 явным способом:

```
C:\> mysqladmin -p -u root shutdown
C:\> mysqladmin -P 3307 -p -u root shutdown
```

Системы Windows, базирующиеся на NT-технологии, имеют в своем арсенале поддержку служб, и вполне реально, что начиная с версии 4.0.2 можно будет задать имя службы во время установки сервера[3]:

```
C:\> mysql-nt --install service_name
```

Это позволяет вам запустить несколько серверов MySQL, указав различные имена служб. Такая возможность подчиняется следующим правилам.

- Без аргумента *service_name* сервер использует стандартное наименование службы (MySql) и берет параметры настройки из группы `[mysqld]` конфигурационного файла.
- С аргументом *service_name* сервер использует имя указанной службы и берет параметры настройки из группы `[service_name]` конфигурационного файла.
- Сервер, который работает с использованием стандартных имен служб, поддерживает имя именованного канала MySql. Любой сервер, для которого имя службы задается явным образом, не будет поддерживать именованные каналы и будет "прослушивать" только соединения TCP/IP, если только вы не задасте параметр `--socket` для отображения другого имени канала.
- Каждый сервер должен работать с разными каталогами данных.

Предположим, что вы хотите запустить два экземпляра `mysqld-nt`, воспользовавшись службой, именованными каналами MySql и mysqlsvc2 и таким же каталогом данных, которые показаны в предыдущем примере. Установить параметры для каждого сервера можно в одном из стандартных конфигурационных файлов (таких как `C:\my.cnf`) следующим образом:

```
# группа для стандартной службы (MySql)
[mysqld]
basedir=C:/mysql
datadir=C:/mysql/data
port=3306
enable-named-pipe
# группа для службы mysqlsvc2
[mysqlsvc2]
basedir=C:/mysql
datadir=C:/mysql/data2
port=3307
enable-named-pipe
socket=mysqlsvc2
```

[3] Начиная с MySQL 3.23.54 также допустимо задавать имя службы.

Для инсталляции и запуска служб воспользуйтесь следующими командами:

```
C:\> mysql-nt --install
C:\> net start MySql
C:\> mysql-nt --install mysqlsvc2
C:\> net start mysqlsvc2
```

Клиенты могут подключаться к стандартному серверу через стандартный порт или канал. Для подключения ко второму серверу его номер порта и имя канала необходимо задать явно:

```
C:\> mysql --port=3307
C:\> mysql --host=. --socket=mysqlsvc2
```

Для отключения серверов используйте команды `mysqladmin shutdown`, `net stop` или возможности диспетчера служб. При деинсталляции серверов остановите их, если они работают, а затем удалите их с параметром `-remove`, указав имя, которое было задано во время инсталляции:

```
C:\> mysql-nt --remove
C:\> mysql-nt --remove mysqlsvc2
```

Начиная с MySQL 4.0.3 можно задавать завершающий параметр `--defaults-file` в командной строке при установке сервера:

```
C:\> mysqld-nt --install service_name --defaults-file=file_name
```

Это предоставит вам альтернативные средства обеспечения специфических для серверов параметров. Имя файла должно запоминаться и использоваться каждый раз при запуске сервера, а также считывать параметры из группы `[mysqld]` этого файла. Для того чтобы воспользоваться синтаксисом установки сервера, необходимо задать имя службы; чтобы воспользоваться стандартной службой, непосредственно укажите имя MySql.

Установка серверов репликации

Одна из форм репликации баз данных заключается в копировании базы данных на другой сервер. Но, для того, чтобы копия была всегда актуальной, эта операция должна быть повторена позднее. Для того чтобы добиться постоянного обновления вторичной базы данных по мере внесения изменений в содержимое основной базы данных, воспользуйтесь возможностями репликации, имеющейся в MySQL. Это позволит вам всегда иметь копию базы данных и гарантирует, что изменения исходной базы данных автоматически копируются по временному принципу.

Концепция репликации

Репликация баз данных в MySQL базируется на следующих принципах.

- Во взаимоотношениях репликации один сервер является главным, а другой — подчиненным. Каждому серверу должен быть присвоен уникальный идентификатор репликации.
- Главный и подчиненный серверы стартуют в синхронизированном состоянии копий реплицируемой базы данных. После этого на главном сервере вносятся изменения, которые затем передаются на подчиненный сервер.

Изменения не вносятся непосредственно на реплицированных базах данных на подчиненном сервере.

- Средства обмена обновлениями опираются на двоичные журналы обновлений главного сервера, куда записываются обновления, которые будут отправлены на подчиненный сервер. Следовательно, для обеспечения механизма репликации на главном сервере должно быть включено двоичное протоколирование.
- Главный сервер должен иметь разрешение на подключение к главному серверу и на запрос обновлений. Состояние подчиненного сервера отслеживается на основании координат репликации, которые состоят из имени файла двоичного журнала, из которого подчиненный в данный момент сервер читает, и его положение внутри этого файла.
- Один главный сервер может иметь сразу несколько подчиненных серверов (обратное невозможно). Однако подчиненный сервер может служить главным сервером для другого подчиненного сервера, создавая тем самым цепочку серверов репликации.

Поддержка репликации – относительно новая вещь для СУБД MySQL и продолжает постоянно развиваться, поэтому иногда сложно отслеживать, какая возможность, имеющая отношение к репликации, была добавлена и когда это было сделано. Возможности репликации были добавлены в MySQL 3.23.15, но в целом, для того, чтобы иметь полноценную репликацию, лучше всего использовать самую последнюю версию сервера. Кроме того, необходимо рассмотреть ограничения возможности репликации между различными версиями сервера. Основные правила совместимости выглядят следующим образом.

- Подчиненные серверы версии 3.23.x не могут обмениваться данными с главными серверами версии 4.x.
- Подчиненные серверы версии 4.0.0 могут обмениваться данными с главными серверами версии 4.0.0.
- Подчиненные серверы версии 4.0.1 и выше могут производить обмен данными с главными серверами версий 3.23.x и 4.x.

В целом рекомендуется придерживаться следующих правил.

- Постарайтесь добиться соответствия версий СУБД MySQL между главным и подчиненными серверами. Например, попытайтесь, чтобы главный сервер версии 3.23.x работал с подчиненными серверами версии 3.23.x, а не главные серверы версии 4.x с подчиненными серверами 3.23.x или наоборот.
- Если имеются версии 3.23.x или 4.x, постарайтесь использовать самые последние. Это дает преимущество выбора из богатой палитры возможностей и самое большое количество снятых ограничений и решенных проблем.

Кроме совместимости версий серверов, они должны быть совместимы по возможностям. Например, если главный сервер реплицирует таблицы InnoDB, использующие внешние ключи и таблицы MyISAM, для которых требуются возможности RAID, подчиненный сервер должен иметь поддержку таблиц InnoDB и массивов RAID.

Установка взаимоотношений репликации главный–подчиненный

Следующая процедура описывает процесс установки взаимоотношений репликации главный–подчиненный между двумя серверами.

1. Присвойте серверам соответствующие идентификаторы, которые должны отличаться и быть положительными целыми числами длиной не больше 64 бит. Значения идентификаторов потребуются для параметра запуска сервера `server-id`, который используется каждым сервером.
2. Подчиненный сервер должен иметь свою учетную запись на главном сервере, чтобы он мог подключаться и запрашивать обновления. На главном сервере учетная запись создается следующим образом:

   ```
   GRANT REPLICATION SLAVE ON *.*
   TO 'slave_user'@'slave_host'
   IDENTIFIED BY 'slave_pass';
   ```

   Для этого необходимы привилегия `REPLICATION SLAVE` для MySQL 4.0.2 и выше и привилегия `FILE` для более ранних версий. (Привилегия изменилась при изменении таблиц разрешения, сделанных в MySQL 4.0.2.) Значения *`slave_user`* и *`slave_pass`* понадобятся при установке подчиненного сервера. Если эта учетная запись используется только для решения ограниченной задачи репликации, никаких других привилегий для этой учетной записи не требуется. Однако может потребоваться выделить дополнительные привилегии учетной записи, если планируется использовать учетную запись для подключения к главному серверу с подчиненного сервера "вручную" с помощью программы `mysql` в целях тестирования. В таком случае вы не будете так стеснены в правах. (Например, если `REPLICATION SLAVE` является единственной привилегией, которую имеет учетная запись, у пользователя не будет возможности даже просмотреть имена баз данных на главном сервере с помощью оператора `SHOW DATABASES`.)
3. Выполните первоначальную синхронизацию подчиненного и главного сервера, скопировав базы данных с главного сервера на подчиненный сервер. Как это можно сделать? Создайте резервную копию главного компьютера, а потом перенесите его на подчиненный компьютер и загрузите его в подчиненный сервер. Или скопируйте все базы данных по сети с главного сервера на подчиненный. Описание методик резервного копирования баз данных приведено в главе 13, "Резервное копирование и восстановление баз данных".

   Другой способ создания подчиненного сервера заключается в использовании оператора `LOAD DATA FROM MASTER`, появившегося в MySQL 4.0.0. Использовать этот оператор можно в соответствии со следующими правилами.

   - Все реплицируемые таблицы должны быть таблицами MyISAM.
   - Этот оператор можно применять при подключении к подчиненному серверу с использованием учетной записи, обладающей привилегией `SUPER`.
   - Учетная запись, используемая подчиненным сервером для подключения к главному серверу, должна иметь привилегии `RELOAD` и `SUPER`. Для того

чтобы присвоить этой учетной записи привилегии REPLICATION SLAVE, при настройке учетной записи необходимо модифицировать оператор GRANT, представленный выше, в виде:

```
GRANT REPLICATION SLAVE,RELOAD,SUPER ON *.*
TO 'slave_user'@'slave_host'
IDENTIFIED BY 'slave_pass';
```

Но надо помнить, что это относится к учетной записи, расположенной на главном сервере, и отличается от учетной записи на подчиненном сервере.

- Операция, выполняемая оператором LOAD DATA FROM MASTER, требует глобальной блокировки записи. При этом блокируются все обновления на главном сервере на протяжении всей передачи на подчиненный сервер.

  Некоторые из этих ограничений будут реализованы в будущем. Каким бы методом вы ни воспользовались при копировании баз данных с главного сервера на подчиненный, необходимо гарантировать полное отсутствие обновлений на главном сервере с момента начала процедуры резервного копирования и до того момента, когда главный сервер будет перенастроен для включения двоичной регистрации.

4. Выключите главный сервер, если он работает.
5. Внесите изменения в настройки сервера, добавив его идентификатор репликации, и активизируйте двоичную регистрацию. Для этого добавьте в конфигурационный файл следующие строки:

```
[mysqld]
server-id=master_server_id
log-bin=binlog_name
```

6. Перезапустите главный сервер; начиная с этого момента он будет регистрировать обновления, записывая их в двоичный журнал. (Если двоичное протоколирование уже было активизировано, перед перезапуском сервера зарезервируйте двоичные журналы. После запуска сервера подключитесь к нему и введите оператор RESET MASTER для очистки существующих двоичных журналов.)
7. Выключите подчиненный сервер, если он работает.
8. Настройте подчиненный сервер так, чтобы он "знал" идентификатор своего главного сервера репликации, где его можно найти и каким образом к нему подключиться. В простейшем случае два сервера будут работать на разных узлах, используя стандартный порт TCP/IP и всего четыре строки в группе [mysqld] конфигурационного файла, который считывает подчиненный сервер во время запуска:

```
[mysqld]
server-id=slave_server_id
master-host=master_host
master-user=slave_user
master-password=slave_pass
```

Значение *slave_server_id* является идентификатором репликации подчиненного сервера. Он должен отличаться от идентификатора главного сервера.

Значение *master_host* — это имя узла, на котором работает главный сервер. На ОС UNIX, если главный и подчиненный узел — это одно и то же, используйте IP-адрес 127.0.0.1, а не `localhost`. Это гарантирует, что подчиненный сервер использует соединение TCP/IP. (Для подключения к `localhost` используется файл разъема, но репликация через файл разъема не поддерживается.) Значения *slave_user* и *slave_pass* хранят имя и пароль учетной записи, которая была предварительно создана на главном сервере для использования ее при подключении подчиненного сервера для запроса обновлений. Эти строки должны находиться в конфигурационном файле, который доступен только для учетной записи администратора MySQL на подчиненном сервере, так как имя пользователя и его пароль являются конфиденциальной информацией. Так, например, никогда не используйте `/etc/my.cnf`, который доступен всем. Одна из возможностей заключается в использовании `my.cnf` в каталоге данных сервера, закрытом для чтения (подробно об этом рассказывается в главе 12, "Безопасность"). Если необходимо еще больше детализировать условия подключения к главному серверу, можно включить в конфигурационный файл строки master-port, которой задается номер порта в том случае, если главный сервер слушает нестандартный порт.

При ненадежном соединении между главным и подчиненным серверами можно изменить стандартные установки для задания перерыва между попытками соединения (60 секунд) или максимального количества попыток до отказа от соединения (86400 раз). Для этого используются параметры `master-connect-retry` и `master-retry-count`.

9. Перезапустите подчиненный сервер. Подчиненный сервер использует два источника информации для определения, имеет ли место процесс репликации. Одним из них является файл `master.info`, размещенный в каталоге данных, а другим — информация о настройках, которая задается параметрами запуска сервера. Во время первого запуска подчиненного сервера он не находит файл `master.info` и использует для определения порядка соединения значения параметров `master-xxx`, заданных в конфигурационном файле.

   Затем создается новый файл `master.info`, в который будет записываться эта информация и ее репликационный статус. Поэтому, когда подчиненный сервер запускается, он использует информацию из файла `master.info`, а не из конфигурационного файла. (Это означает, что если вы позднее поменяете информацию о главном узле в конфигурационном файле, потребуется удалить файл `master.info` и перезапустить подчиненный сервер, или он проигнорирует внесенные изменения.)

Описанная процедура опирается на предположение, что необходимо производить репликацию с одного сервера на другой, включая базу данных `mysql`, содержащую таблицу разрешения. Если необходимо иметь на двух узлах одинаковые учетные записи, репликацию базы данных `mysql` можно исключить. (Например, необходимо настроить свой собственный подчиненный сервер репликации, к которому не смогут подключиться другие пользователи, даже если они имеют собственную учетную запись на главном сервере.) Для того чтобы исключить базу данных `mysql`, добавьте в группу `[mysqld]` следующую строку при активизации двоичной регистрации:

```
binlog-ignore-db=mysql
```

Для исключения нескольких баз данных добавьте эту строку несколько раз, по одному разу на каждую базу данных.

После того как процедура репликации была настроена и заработала, для настройки или управления главного и подчиненного серверов может пригодиться несколько операторов. Подробнее об этих операторах можно узнать в приложении Г, "Синтаксис SQL". А вкратце о них можно сказать следующее.

- Операторы `SLAVE STOP` и `SLAVE START` останавливают и запускают функции сервера, связанные с репликацией. Эти операторы могут пригодиться для приостановки работы подчиненного сервера, например во время резервного копирования.
- Оператор `SHOW SLAVE STATUS` на подчиненном сервере отображает координаты репликации. Эта информация используется для определения необходимости использования двоичных журналов.
- Оператор `PURGE MASTER` на главном сервере выводит из действия двоичные журналы. Его можно использовать после `SHOW SLAVE STATUS` на каждом из подчиненных серверов.
- Оператор `CHANGE MASTER` на подчиненном сервере позволяет отменить несколько текущих параметров репликации подчиненного сервера, например, информацию о том, какие двоичные журналы обновления он должен считывать с главного сервера или в какой файл регистрации переключений он должен производить запись.

Начиная с MySQL 4.0.2 подчиненные серверы репликации работают с двумя потоками. Поток ввода-вывода отвечает за обмен данными с главным сервером, осуществляет запрос обновлений из него и записывает обновления в файл регистрации переключений. Поток SQL считывает журналы регистрации переключений и выполняет обновления, которые он там находит. (Журналы переключений служат средством, с помощью которого поток ввода-вывода передает изменения на поток SQL.) Для подчиненных серверов, которые работают в соответствии с двухпотоковой моделью, можно использовать операторы `SLAVE STOP` и `SLAVE START` для остановки и возобновления работы каждого потока в отдельности. Для этого в конец оператора необходимо добавить параметры `IO_THREAD` или `SQL_THREAD`. Например, оператор `SLAVE STOP SQL_THREAD` останавливает выполнение подчиненным оператором обновлений в журнал переключений, который позволяет подчиненному серверу продолжать считывать обновления с главного сервера и записывать их в журнал переключений.

Журналы переключений упорядочены в пронумерованной последовательности, аналогично двоичным журналам обновления. Существует также индексный файл журнала переключения, аналогичный индексу двоичного журнала регистрации обновлений. Стандартными именами журнала переключений и индексов являются *HOSTNAME*`-relay-bin.`*nnn* и *HOSTNAME*`-relay-bin.index`.

Стандартные значения можно изменить с помощью параметров `--relay-log` и `--relay-log-index` во время запуска сервера. Другим файлом статуса является информационный файл переключений, который по умолчанию имеет имя `relay-log.info` и может быть изменен параметром `--relay-log-info-file`.

Обновление MySQL

Первая представленная мировой общественности версия сервера MySQL имела номер 3.11.1. В настоящее время применяются версии серии 4.0.x и ведется разработка и тестирование версий серии 4.1.x. Обновления появляются достаточно часто и выходят как в виде рабочих версий (stable releases), так и опытных версий (development series). Разработчики MySQL, очевидно, работают безустанно, благодаря чему обновления к версиям появляются постоянно (несколько раз в год). Причем выпускаются обновления как для рабочих серий, так и для опытных. Эффективность развития заставляет администратора MySQL задумываться, обновлять ли используемую инсталляцию MySQL при выпуске новых версий. Материал, изложенный в данном разделе, поможет найти ответ на данный вопрос.

Первое, на что следует обратить внимание, — это на отличие новой версии от предыдущих. Для того чтобы постоянно находиться в курсе новостей, подпишитесь на список рассылки `announce@lists.mysql.com`. Каждое объявление включает новые сведения об изменениях. (Кроме того, можно обратиться к соответствующему разделу справочного руководства по MySQL, в котором выделена эта информация.) Затем попытайтесь ответить на следующие вопросы.

- Приходилось ли вам в прошлом сталкиваться с проблемами, которые устраняются в новой версии?
- Добавлены ли в новую версию возможности, которые действительно необходимы?
- Повышена ли в новой версии эффективность выполнения часто используемых операций?

Если все ответы на представленные выше вопросы отрицательны, особых причин обновлять версии нет. Если же ответ хотя бы на один вопрос утвердительный, следует еще немного подумать. Настоятельно рекомендуется на данном этапе хотя бы несколько дней подождать и поискать в рассылке, посвященной MySQL, отзывы других пользователей о новой версии. Эффективным ли было для них обновление? Какие ошибки и недоработки найдены?

На принимаемое решение могут оказать влияние и следующие факторы.

- Обновления официально используемых версий чаще всего устраняют существующие ошибки, а не добавляют новые возможности. Поэтому обновления официальных версий несут в себе меньше риска, чем обновления опытных. (Безусловно, администраторы, работающие на сервере опытной серии, зачастую вовсе не принимают в расчет подобный риск.)
- Вполне возможно, что после обновления MySQL возникнет необходимость и в обновлении других программ, связанных с клиентской библиотекой сервера MySQL. Так, например, обновление MySQL может повлечь за собой перестройку модулей PHP, Apache и Perl `DBD::mysql` для подключения новых версий клиентских библиотек к этим программам. (Первый признак необходимости в этом — выгрузка всех связанных с MySQL сценариев DBI и PHP после обновления MySQL.) Перестройка этих программ — несложная задача, однако если администратор все же желает ее избежать, возможно, лучше

отказаться от обновления MySQL. Эта проблема не так опасна, если использовать статически, а не динамически связанные программы. Однако в этом случае возрастают требования к оперативной памяти.

Если даже после учета всех этих моментов отсутствует однозначный ответ, что следует обновлять текущую инсталляцию, всегда можно протестировать новый сервер, установив его рядом с текущим. Его можно запустить параллельно с рабочим сервером или инсталлировать на отдельном компьютере. Если же возможности не позволяют выделить отдельный компьютер исключительно для тестирования, инсталлируйте новый сервер на одном компьютере с рабочим. В этом случае обязательно убедитесь, что для нового сервера установлены другие значения для таких параметров, как месторасположение инсталляции, каталог данных, номер сетевого порта и разъем, через которые осуществляется подключение к серверу. Более детально об этом см. в разделе "Эксплуатация нескольких серверов в ОС Windows".

Независимо от места инсталляции сервера, создайте для проверки его работы копию существующих баз данных. Как это сделать, рассказывается в главе 13, "Резервное копирование и восстановление баз данных".

Приняв решение об обновлении версии, поищите в справочном руководстве по MySQL обязательные инструкции по обновлению. Обычно в них нет ничего нового, но все равно досконально проверьте все, — это очень важно, так как в новых выпусках могут присутствовать шаги, несовместимые с шагами из предыдущих выпусков.

Если после обновления до версии, которая не является совместимой в обратном порядке с более старыми версиями, вы решаете вернуться на новую версию, это сделать уже будет не так просто. Например, при переходе с версии MySQL 4.0.x на версию 4.1.x и преобразовании таблиц в формат 4.1 они станут несовместимыми с серверами 4.0.x. В таком случае переход на предыдущую версию может оказаться затруднительным.

Не бойтесь использовать опытные версии серверов

Опытные версии серверов большинство администраторов стараются не использовать для работы с основными базами данных, однако профессионалы настоятельно рекомендуют по крайней мере протестировать работу такого опытного сервера, установив его отдельно и создав на нем копию рабочих данных. Чем больше пользователей попробуют работать с этими версиями, тем больше ошибок будет найдено. Поиск ошибок в MySQL — важный фактор развития системы, поскольку разработчики всегда стараются устранить проблемы, о которых сообщают пользователи.

Можно попробовать протестировать сервер на реальных запросах. Для этого достаточно назначить рабочий сервер в качестве главного сервера репликации, а тестируемый сервер — в качестве подчиненного сервера репликации, обеспечив его непрерывным потоком запросов. Это также позволяет посмотреть, как тестируемый сервер работает в зазеркаленном режиме.

12

Безопасность

В этой главе...

Внутренняя безопасность: защита доступа к каталогу данных

Внешняя безопасность: защита сетевого доступа

Установка безопасных соединений

В этой главе основное внимание уделяется действиям администратора, направленным на поддержание безопасности и целостности инсталляции MySQL. В главе 11, "Общее администрирование MySQL", уже были затронуты такие важные аспекты безопасности, как установка пароля основного MySQL-пользователя `root` и настройка учетных записей пользователей. Однако в предыдущей главе эти вопросы рассматривались как часть процесса инсталляции и запуска. В этой же главе детально описывается следующее.

- Почему безопасность так важна и от каких видов атак следует защищаться в первую очередь.
- Какой риск несет в себе подключение пользователей локального компьютера (внутренняя безопасность) и как его минимизировать.
- Какой риск несет в себе подключение пользователей к серверу через сеть (внешняя безопасность) и как его минимизировать.

Администратор MySQL полностью отвечает за безопасность содержимого баз данных, поэтому должен позаботиться о том, чтобы доступ к их записям пользователи получали только после жесткой авторизации. Необходимо обеспечить как внутреннюю, так и внешнюю безопасность.

Внутренняя безопасность связана с выполнением действий на уровне файловой системы. Она обеспечивает защиту каталога данных MySQL от атак пользователей, работающих на том же компьютере, на котором запущен сервер. Нет смысла долго настраивать таблицы разрешений, контролирующие доступ клиен-

тов через сеть, если из-за неправильно установленных полномочий каталога данных любой локальный пользователь может заменить файлы этих таблиц!

Внешняя безопасность предназначена для управления внешними клиентскими соединениями и соответственно необходима для защиты доступа к серверу MySQL через сетевые соединения. Таблицы разрешений MySQL должны быть настроены таким образом, чтобы доступ к базам данных предоставлялся только после правильного ввода имени и пароля.

Еще одна опасность представляет собой возможность третьего лица наблюдать за сетью и перехватывать поток данных между клиентом и сервером. Если это вас волнует, то при установке СУБД MySQL установите поддержку соединений с использованием протокола SSL.

В этой главе рассматриваются все описанные выше вопросы, а также приведены инструкции, выполнение которых позволит заблокировать неавторизированный доступ как изнутри, так и извне системы.

В главе часто упоминаются учетная запись, используемая для работы сервера MySQL и выполнения административных задач, а также имя учетной записи, `mysqladm`, и имя группы, `mysqlgrp`. Замените имена в примерах на используемые вами.

Внутренняя безопасность: защита доступа к каталогу данных

В этом разделе объясняется, как защитить инсталляцию MySQL от доступа непривилегированных пользователей. Это относится только к системам семейства UNIX. Я предполагаю, что если ваш сервер работает под управлением Windows, то отсутствуют подключенные пользователи и вы имеете полный контроль над машиной. (Другими словами, лучше, если сервер работает под управлением Windows, вне доступа пользователей вообще, и ограничен общий доступ.) Во время инсталляции MySQL создается несколько каталогов, которые требуют нестандартной защиты. Например, нет необходимости в доступе к исполняемому файлу сервера кого-либо, кроме администратора, а клиентские программы должны быть общедоступны, но только в режиме чтения.

Остальные файлы, подлежащие защите, создаются либо после начальной установки, либо вами во время первой настройки, либо самим сервером во время работы. В группу файлов, созданных вами, входят файлы настроек и файлы, связанные с протоколом SSL. Во время работы сервера создаются каталоги баз данных, файлы в них, которые соответствуют таблицам в базах данных, файлы журналов и файл разъема UNIX.

Очевидно, что необходимо постоянно поддерживать закрытость баз данных сервера. Владельцы баз данных обычно вполне обоснованно полагают, что содержимое их баз данных закрыто для других пользователей. Даже если они так не считают, все равно не стоит выставлять информацию на всеобщее обозрение или делать ее открытой из-за плохой защиты каталога данных. Файлы журналов также должны быть надежно защищены, поскольку они содержат тексты запросов. Ведь любой пользователь, обладающий доступом к файлам журналов, сможет легко отследить все транзакции с базами данных.

Важность подобной защиты еще больше подчеркивается тем фактом, что в файлах журналов регистрируются даже запросы с операторами GRANT и SET PASSWORD, включающие пароли. (Вообще-то, в MySQL используется шифрование паролей, однако оно применяется только после установления соединения с предварительной проверкой пароля. Процесс установки пароля включает выполнение таких операторов, как GRANT, INSERT и SET PASSWORD. В файлы журналов они заносятся в обычной текстовой форме.) Взломщику, получившему доступ к файлам журналов, достаточно запустить команду grep, чтобы раскрыть важную информацию операторов GRANT и PASSWORD.

Некоторые файлы должны быть доступны клиентским программам. Примером такого файла служит файл разъема UNIX. Естественно, вы захотите ограничить доступ к нему только чтением. Например, удаление пользователем файла разъема приведет к невозможности пользователей подключиться к серверу.

Как похитить данные

Сервер MySQL включает гибкую систему привилегий, реализованную через таблицы разрешений базы данных mysql. Изменяя содержимое этих таблиц, можно предоставить или запретить доступ определенным пользователям. Именно таким образом обеспечивается защита от несанкционированного доступа к данным по сети. Однако нет смысла тратить много времени и усилий на обеспечение надежной защиты баз данных от доступа сетевых пользователей, если локальный пользователь, работающий на компьютере с сервером наряду с администратором, сможет легко просматривать и изменять содержимое каталога данных. Этот вид защиты можно проигнорировать, лишь будучи полностью уверенным, что никто другой не сможет зарегистрироваться и работать на компьютере, используемом для запуска сервера MySQL.

Не следует предоставлять пользователям компьютера, на котором установлен сервер, права чтения для файлов каталога данных, поскольку они могут легко повредить файлы состояния и таблицы баз данных. Не менее опасен и доступ с правами чтения. Обладая возможностью чтения файлов таблицы, очень легко похитить эти файлы и позволить другому серверу MySQL самому показать содержимое таблицы в виде простого текста. Как? Да очень просто.

1. Проинсталлируйте собственную версию MySQL на компьютере с сервером, установив отдельные значения порта, разъема и каталога данных.
2. Запустите сценарий mysql_install_db для инициализации каталога данных. Это позволит получить доступ к серверу в качестве MySQL-пользователя root, получив полный контроль над механизмом доступа. Параллельно будет создана база данных test.
3. Скопируйте файлы, соответствующие похищаемой таблице или таблицам, в подкаталог test каталога данных сервера.
4. Запустите второй сервер. Доступ к таблицам открыт! Оператор SHOW TABLES FROM test указывает на наличие копий похищенных таблиц, а с помощью оператора SELECT * можно увидеть их содержимое.
5. Какой-нибудь недоброжелательный пользователь может изменить полномочия учетной записи анонимного пользователя сервера, в результате чего

к серверу для доступа к базе данных `test` сможет подключиться любой. В результате этих нехитрых действий содержимое похищенных таблиц становится открытым для всех.

Задумайтесь над этим. Вне всяких сомнений, ни один администратор не захочет столкнуться с подобной ситуацией, поэтому защитите себя, воспользовавшись инструкциями, приведенными ниже.

Защита инсталляции MySQL

Описанная здесь процедура показывает настройку прав доступа и режимов доступа к каталогам и файлам вашей инсталляции MySQL. При этом используются имя пользователя `mysqladm` и имя группы `mysqlgrp`. Им будут предоставлены права доступа к инсталляции. Также предполагается, что вся инсталляция находится в одном каталоге, а не разбросана по всей файловой системе. Основным каталогом инсталляции является `/usr/local/mysql`, а каталогом данных — `/usr/local/mysql/data`. По окончании процедуры я объясню, как справиться с некоторыми нестандартными размещениями каталогов. Пути к каталогам в вашей системе могут отличаться, поэтому вы должны адаптировать их исходя из существующих настроек. Если у вас несколько серверов, вам придется повторить процедуру для каждого из них.

Чтобы определить, расположены ли в каталоге данных какие-либо незащищенные файлы и подкаталоги, достаточно запустить в нем команду `ls -l`. Найдите все файлы и каталоги, для которых активизированы групповые или другие полномочия. Вот для примера краткий список незащищенных элементов каталога данных, среди которых есть и подкаталоги баз данных.

```
% ls -la /usr/local/mysql/data
total 10148
drwxrwxr-x 11 mysqladm wheel    1024 May  8 12:20 .
drwxr-xr-x 22 root     wheel     512 May  8 13:31 ..
drwx------  2 mysqladm mysqlgrp  512 Apr 16 15:57 menagerie
drwxrwxr-x  2 mysqladm wheel     512 Jun 25 1998  mysql
drwx------  7 mysqladm mysqlgrp 1024 May  7 10:45 sampdb
drwxrwxr-x  2 mysqladm wheel    1536 Jun 25 1998 test
drwx------  2 mysqladm mysqlgrp 1024 May  8 18:43 tmp
```

Некоторые из каталогов баз данных имеют соответствующие полномочия: `drwx------`, которые позволяют читать, записывать и выполнять доступ только владельцу, но запрещают доступ кому-либо еще.

К остальным каталогам предоставлен полный доступ. Атрибут `drwxrwxr-x` предоставляет доступ и чтение всем пользователям, даже не состоящим в группе `mysqlgrp`. Такая ситуация стала результатом использования сервера в течение долгого периода времени. Менее ограничивающие полномочия созданы старыми версиями серверов, которые были менее жестки в отношении прав доступа, чем новые. (Обратите внимание, что каталоги `menagerie` и `tmp`, наиболее ограничивающие доступ, имеют более позднее время создания.) Последние версии MySQL обязательно проверят, чтобы эти файлы смогли прочитать только пользователи, под именем которых работает сервер.

Для проверки основного каталога инсталляции MySQL используйте команду `ls -l`.

Например, вы могли бы получить такой результат:

```
% ls -la /usr/local/mysql
total 44
drwxrwxr-x 13 mysqladm mysqlgrp 1024 May  7 10:45 .
drwxr-xr-x 24 root wheel        1024 May  1 12:54 ..
drwxr-xr-x  2 mysqladm mysqlgrp 1024 Jul 16 20:58 bin
drwxrwxr-x 12 mysqladm wheel    1024 May  8 12:20 data
drwxr-xr-x  3 mysqladm mysqlgrp  512 May  7 10:45 include
drwxr-xr-x  2 mysqladm mysqlgrp  512 May  7 10:45 info
drwxr-xr-x  3 mysqladm mysqlgrp  512 May  7 10:45 lib
drwxr--r-x  2 mysqladm mysqlgrp  512 Jul 16 20:58 libexec
drwxr-xr-x  3 mysqladm mysqlgrp  512 May  7 10:45 man
drwxr-xr-x  6 mysqladm mysqlgrp 1024 May  7 10:45 mysql-test
drwxr-xr-x  3 mysqladm mysqlgrp  512 May  7 10:45 share
drwxr-xr-x  7 mysqladm mysqlgrp 1024 May  7 10:45 sql-bench
```

Как уже указывалось, полномочия и права доступа к каталогу данных должны быть изменены. Еще одно изменение может касаться ограничения доступа к каталогу `libexec`, в котором находится сервер `mysqld`. Доступ к нему не нужен никому, кроме администратора, поэтому его можно сделать личным каталогом пользователя `mysqladm`.

Для решения описанных проблем используйте следующую процедуру. Нужно сделать все доступным только пользователю `mysqladm`, кроме тех частей инсталляции, к которым пользователи должны иметь доступ.

1. Если работает сервер MySQL, завершите его работу:

   ```
   % mysqladmin -p -u root shutdown
   ```

2. Измените имена пользователя и группы на имена учетной записи администратора для всей инсталляции MySQL с помощью следующей команды (выполнять ее нужно, будучи пользователем `root`):

   ```
   # chown -R mysqladm.mysqlgrp /usr/local/mysql
   ```

   Довольно популярен способ сделать все принадлежащим пользователю `root`, кроме каталога данных. Это можно сделать следующим образом:

   ```
   # chown -R root.mysqlgrp /usr/local/mysql
   # chown -R mysqladm.mysqlgrp /usr/local/mysql/data
   ```

   Если вы отдадите основные права доступа пользователю `root`, вы должны будете выполнять большинство последующих шагов как пользователь `root`. В противном случае выполняйте все как пользователь `mysqladm`.

3. Для базового каталога и всех его подкаталогов, к которым должны иметь доступ пользователи, измените режим таким образом, чтобы пользователь `mysqladm` имел полный доступ, а все остальные — только право чтения. Он может быть уже установлен, но если нет, измените его. Например, полномочия для базового каталога могут быть установлены так:

   ```
   % chmod 755 /usr/local/mysql
   % chmod u=rwx,go=rx /usr/local/mysql
   ```

   Подобным образом можно установить полномочия для каталога `bin`, в котором хранятся клиентские программы:

   ```
   % chmod 755 /usr/local/mysql/bin
   ```

```
% chmod u=rwx,go=rx /usr/local/mysql/bin
```

Каталоги, к которым не требуется доступ пользователей, можно сделать личными каталогами пользователя mysqladm. В качестве примера можно привести каталог libexec, в котором установлен сервер.

Каждая из команд установит этот режим:

```
% chmod 700 /usr/local/mysql/libexec
% chmod u=rwx,go-rwx /usr/local/mysql/libexec
```

4. Сделайте все ваши каталоги и файлы личными каталогами пользователя mysqladm, чтобы предотвратить доступ к вашим данным другим пользователям. Это можно сделать с помощью команды

```
% chmod -R go-rwx /usr/local/mysql/data
```

После выполнения всех шагов базовый каталог с инсталляцией вашей MySQL будет иметь такие права доступа и полномочия:

```
% ls -la /usr/local/mysql
total 44
drwxr-xr-x 13 mysqladm mysqlgrp 1024 May  7 10:45 .
drwxr-xr-x 24 root     wheel    1024 May  1 12:54 ..
drwxr-xr-x  2 mysqladm mysqlgrp 1024 Jul 16 20:58 bin
drwx------ 12 mysqladm mysqlgrp 1024 May  8 12:20 data
drwxr-xr-x  3 mysqladm mysqlgrp  512 May  7 10:45 include
drwxr-xr-x  2 mysqladm mysqlgrp  512 May  7 10:45 info
drwxr-xr-x  3 mysqladm mysqlgrp  512 May  7 10:45 lib
drwx------  2 mysqladm mysqlgrp  512 Jul 16 20:58 libexec
drwxr-xr-x  3 mysqladm mysqlgrp  512 May  7 10:45 man
drwxr-xr-x  6 mysqladm mysqlgrp 1024 May  7 10:45 mysql-test
drwxr-xr-x  3 mysqladm mysqlgrp  512 May  7 10:45 share
drwxr-xr-x  7 mysqladm mysqlgrp 1024 May  7 10:45 sql-bench
```

Как видите, все принадлежит пользователю mysqladm, который включен в группу mysqlgrp. (Строка ".." соответствует каталогу /usr/local/mysql. Хорошо, что этот каталог принадлежит пользователю root. Вы, наверное, не хотели бы, чтобы непривилегированные пользователи имели дело с каталогом инсталляции.) Каталог данных будет иметь более жесткие полномочия:

```
% ls -la /usr/local/mysql/data
total 10148
drwx------ 11 mysqladm mysqlgrp 1024 May  8 12:20 .
drwxr-xr-x 22 mysqladm mysqlgrp  512 May  8 13:31 ..
drwx------  2 mysqladm mysqlgrp  512 Apr 16 15:57 menagerie
drwx------  2 mysqladm mysqlgrp  512 Jun 25 1998 mysql
drwx------  7 mysqladm mysqlgrp 1024 May  7 10:45 sampdb
drwx------  2 mysqladm mysqlgrp 1536 Jun 25 1998  test
drwx------  2 mysqladm mysqlgrp 1024 May  8 18:43 tmp
```

Здесь строка ". ." соответствует каталогу данных, который является основным каталогом MySQL. Исключением в такой политике доступа может быть необходимость доступа к определенным файлам. Например, если вы создадите файл my.cnf в каталоге данных и захотите поместить в этот файл параметры клиента, вам необходимо будет приоткрыть доступ к каталогу, иначе клиентские программы не смогут прочитать файл. То же самое относится к файлу разъема UNIX. Чтобы разрешить доступ к этим файлам клиентских программ без предоставле-

ния полного доступа к каталогу данных, используйте следующую команду:

```
% chmod go+x /usr/local/mysql/data
```

Как уже говорилось, при выполнении этих процедур подразумевается, что все файлы, имеющие отношение к СУБД MySQL, находятся в одном базовом каталоге. Если это не так, вам придется найти каждый каталог MySQL и выполнить процедуру для каждого из них. Например, если ваш каталог данных находится в `/var/mysql/data`, а не в каталоге `/usr/local/mysql`, вам придется задать две команды изменения прав доступа соответственно:

```
# chown -R mysqladm.mysqlgrp /usr/local/mysql
# chown -R mysqladm.mysqlgrp /var/mysql/data
```

Или же допустим, что вы создали каталог `innodb` в каталоге инсталляции MySQL для хранения всех файлов типа InnoDB. По умолчанию все эти файлы помещены в каталоге данных. Если вы поместите их в каталог `innodb`, то, вероятно, захотите задать им такой же режим доступа, как и у каталога данных. То же самое относится ко всем файлам, которые вы перенесете из каталога данных, например к файлам журнала.

Еще одна сложность возникает, если некоторые каталоги в корне инсталляции на самом деле являются ссылками, указывающими в другое место. Если ваша версия утилиты chown не работает со ссылками, вам необходимо отследить их самостоятельно и изменить права доступа к объекту. Сделать это можно при помощи команды `find`:

```
# find /usr/local/mysql -follow -print | xargs chown mysqladm.mysqlgrp
```

Аналогичные способы применимы для изменения режимов доступа. Например, если команда chmod не следует ссылкам, используйте вместо нее такую команду:

```
% find /usr/local/mysql/data -follow -print | xargs chmod go-rwx
```

Учтите, что эти инструкции неприменимы, если файлы сервера и клиентских программ находятся в основных каталогах системы вместе с другими программами. (Например, сервер может находиться в каталоге `/usr/sbin`, а клиентские программы — в каталоге `/usr/bin`.) В этом случае права и режимы доступа должны быть такими же, как и у других программ в этих каталогах.

Так как отныне доступ к каталогу данных разрешен только пользователю `mysqladm`, вы должны быть уверены, что сервер всегда работает под управлением пользователя `mysqladm`. Это легко можно сделать, задав пользователя в разделе `[mysqld]` файла `/etc/my.cnf` или `my.cnf` в каталоге данных:

```
[mysqld]
user=mysqladm
```

Таким образом, сервер будет работать под управлением пользователя `mysqladm`, независимо от того, запущен ли он пользователем `root` или `mysqladm`. Дополнительную информацию о работе сервера с использованием какой-то конкретной учетной записи см. в главе 11. После обеспечения защиты инсталляции MySQL можно перезапустить сервер.

Защита файла разъема

Для подключения клиентов к компьютеру в ОС UNIX сервер использует доменный файл разъема. Он обычно общедоступен, так что клиентские программы

могут его использовать. Но, несмотря на это, он не должен находиться в каталоге, к которому любой пользователь имеет полный доступ. Например, файл разъема часто находится в каталоге /tmp, но в некоторых системах UNIX пользователи могут удалять файлы, им не принадлежащие. Это значит, что кто угодно сможет удалить файл разъема и, в результате, нарушить установку подключений клиентских программ к серверу, пока сервер не перезапустится, чтобы создать новый файл разъема. Лучше будет, если пользователи смогут создавать в каталоге /tmp файлы, а удалять их смогут только владельцы. В некоторых инсталляциях файл разъема помещен в каталог данных, вследствие чего ни один клиент не сможет получить доступ к файлу разъема, кроме пользователей root или mysqladm, так как данные в этом каталоге принадлежат пользователю mysqladm. В этом случае придется открыть каталог для всех пользователей.

```
% chmod go+x /usr/local/mysql/data
```

Кроме того, вы можете изменить каталог, в котором будет создаваться файл разъема. Сделать это можно, указав путь в глобальном файле настроек либо перекомпилировав и указав новый путь. Если вы решите изменить путь в файле настроек, укажите его для сервера и клиентов. Вы можете поместить файл разъема в базовом каталоге следующим образом:

```
[mysqld]
socket=/usr/local/mysql/mysql.sock
[client]
socket=/usr/local/mysql/mysql.sock
```

Перекомпилирование – это сложное, но более основательное решение, так как не все клиенты работают с файлами настроек. (В основном это клиенты сторонних разработчиков.) После перекомпиляции новое местоположение файла разъема будет известно клиентским библиотекам, и любой программе, использующей эти библиотеки, будет известен путь, независимо от того, использует она файлы настроек или нет.

Защита файлов настроек

Файлы настроек подвержены потенциальной опасности, так как в них содержатся настройки, которые не должны разглашаться. Нельзя делать файлы настроек общедоступными, если в них содержатся идентификаторы учетных записей и пароли. Файл /etc/my.cnf обычно доступен, так как в нем прописаны общие настройки клиентов. (Это значит, что в нем не должны содержаться настройки сервера, такие как копии паролей, например.) Файлы пользовательских настроек должны быть доступны только пользователям, в чьих корневых каталогах они находятся. С файлами настроек, которые находятся в каталоге данных, можно поступить разными способами, в зависимости от того, для чего вы их используете. Убедиться в правильности режимов и прав доступа к файлам пользовательских настроек можно, запустив программу, которая ищет такие файлы и исправляет ошибки.

Примером такой программы послужит сценарий на языке Perl, chk_mysql_opt_files.pl:

```
#! /usr/bin/perl -w
# chk_mysql_opt_files.pl - проверка файлов my.cnf и проверка того, что
# права владения и режим настроены верно. Владельцем файла должен быть
```

```
пользователь,
# в корневом каталоге которого найден файл. Разрешения типа
# "group" и "other" должны быть отключены.
# Этот сценарий должен работать как root. При его выполнении вводом
является файл
# паролей. Если ваш файл имеет вид /etc/passwd, он должен быть запущен:
# chk_mysql_opt_file.pl /etc/passwd
# Для ОС Mac OS X используется база данных netinfo:
# nidump passwd . | chk_mysql_opt_file.pl
use strict;
while (<>)
{
  my ($uid, $home) = (split (/:/, $_))[2,5];
  my $cnf_file = "$home/.my.cnf";
  next unless -f $cnf_file; # есть файл .my.cnf?
  if ((stat ($cnf_file))[4] != $uid) # проверить владельца
  {
    warn "Замена владельца $cnf_file на $uid\n"
    chown ($uid, (stat ($cnf_file))[5], $cnf_file);
  }
  my $mode = (stat ($cnf_file))[2];
  if ($mode & 077) #Проверить биты доступа group/other
  {
    warn sprintf ("Изменение режима %s с %o на %o\n",
            $cnf_file, $mode, $mode & ~077);
    chmod ($mode & ~077, $cnf_file);
  }
}
exit (0);
```

Сценарий `chk_mysql_opt_files.pl` можно найти в каталоге `admin` дистрибутива sampdb. Запускать его надо как пользователь `root`, так как при его работе необходимо изменять права и режимы доступа файлов других пользователей. Для автоматического запуска сделайте его процессом утилиты `cron`.

Одним из стандартных файлов настроек является файл `my.cnf`, который по умолчанию находится в каталоге данных. Если вы сделаете каталог данных личным каталогом пользователя `mysqladm`, тогда клиентские программы не смогут прочесть этот файл. Это значит, что если возникнет необходимость поместить клиентские настройки в этот файл, вам придется "приоткрыть" доступ к каталогу данных. Для этого включите разрешение выполнения для каталога, а также убедитесь в том, что сам файл доступен.

```
% chmod go+x /usr/local/mysql/data
% chmod +r /usr/local/mysql/data/my.cnf
```

Однако после этого файл станет непригоден для хранения настроек сервера, например паролей.

Внешняя безопасность: защита сетевого доступа

Система безопасности MySQL достаточно гибка, поскольку позволяет настроить привилегии доступа пользователей множеством различных способов. Как правило, установка привилегий реализуется с помощью операторов GRANT и

REVOKE, которые изменяют таблицы разрешений, управляющие клиентским доступом. Тем не менее некоторые администраторы все еще используют старые версии MySQL, которые не поддерживают эти операторы (до версии MySQL 3.22.11). Иногда также администраторы замечают, что при установке с помощью операторов GRANT и REVOKE привилегии функционируют не так, как хотелось бы. В подобных ситуациях весьма полезным может оказаться знание таблиц разрешений MySQL и принципов использования их сервером при определении полномочий. Администратор, владеющий подобными знаниями, может добавлять, удалять или изменять привилегии пользователей посредством изменения собственно таблиц разрешений. Более того, исследование таблиц позволяет гораздо быстрее диагностировать проблемы, связанные с доступом.

Автор предполагает, что читатель уже ознакомился с материалом раздела "Управление пользовательскими учетными записями" главы 11, "Общее администрирование MySQL", и разобрался в принципах работы операторов GRANT и REVOKE. Их применение — удобный способ установки пользовательских учетных записей MySQL и связанных с ними привилегий. Хотя эти операторы представляют собой лишь оболочку, реальные же действия выполняются в таблицах разрешений сервера MySQL.

(В разделе "Настройка учетных записей без помощи оператора GRANT" в главе 11, "Общее администрирование MySQL", объясняется, как модифицировать таблицы разрешений таким образом, чтобы получить те же результаты, что и при использовании операторов GRANT.)

Структура и содержимое таблиц разрешений MySQL

Управление доступом к базам данных MySQL для клиентов, подключившихся к серверу через сеть, осуществляется с помощью содержимого таблиц разрешений. Эти таблицы входят в состав базы данных `mysql` и инициализируются в процессе инсталляции MySQL на компьютере (процесс инсталляции детально описывается в приложении A, "Получение и инсталляция программного обеспечения"). Таблицы называются `user`, `db`, `host`, `tables_priv` и `columns_priv` и используются следующим образом.

- `user`. Данная таблица содержит список всех пользователей, которые могут подключаться к серверу, а также их пароли и все глобальные привилегии, если таковые применяются. Все указанные в этой таблице привилегии являются глобальными и применимы ко всем базам данных. Так, например, если активизировать в этой таблице привилегию DELETE, соответствующий пользователь сможет удалить записи из любой таблицы. Следовательно, необходимо тщательно все обдумать, прежде чем присваивать такую привилегию.

  Из-за привилегированной природы привилегий, хранящихся в таблице `user`, лучше оставить все привилегии в ней выключенными и описать более специфические привилегии в других таблицах, накладывающие большие ограничения. Однако существуют два исключения.

  - Привилегированным пользователям, таким как `root`, и другим административным учетным записям для работы с сервером необходимы глобальные привилегии. Таких учетных записей должно быть несколько.

- Существует несколько "безопасных" привилегий. Это права создания временных таблиц, блокирования таблиц и возможности использования оператора SHOW DATABASES. Во многих инсталляциях эти операции будут разрешены, но в некоторых существует более жесткий контроль, и в них они запрещены.

В таблице user также присутствуют столбцы настроек протокола SSL, отвечающие за установку безопасных подключений, и столбцы управления ресурсами, которые не допустят монополизирования сервера определенной учетной записью.

- db. Данная таблица содержит список всех баз данных и пользователей, обладающих полномочиями на доступ к ним. Определенные в этой таблице привилегии применимы ко всем таблицам соответствующей базы данных.
- host. Эта таблица используется вместе с таблицей db и определяет привилегии доступа к базам данных определенного компьютера. Уровень этого контроля более высокий, чем при использовании одной лишь таблицы db. Операторы GRANT и REVOKE не могут изменять содержимое этой таблицы, поэтому на некоторых компьютерах она вообще не используется.
- tables_priv. Эта таблица определяет привилегии доступа к таблицам. Эти привилегии применимы ко всем столбцам определенной таблицы.
- columns_priv. Данная таблица определяет привилегии доступа к столбцам. Эти привилегии применимы к отдельным столбцам соответствующей таблицы.

В табл. 12.1–12.3 представлено описание структур пяти таблиц разрешений, которые разбиты по типу столбца. Таблицы разрешений содержат столбцы двух видов: столбцы области, определяющие область применения записи, и столбцы привилегий, определяющие присваиваемые привилегии. Столбцы привилегий можно разделить на столбцы административных действий и столбцы, связанные с операциями над базами данных и таблицами. В таблице user присутствуют дополнительные глобальные столбцы соединений SSL и управления ресурсами. Некоторые таблицы разрешений включают и столбцы другого рода, однако здесь они не рассматриваются.

Таблица 12.1. Столбцы области применения таблицы разрешений

Столбцы области доступа				
user	db	tables_priv	columns_priv	host
Table	Table	Table	Table	Table
Host	Host	Host	Host	Host
User	User	User	User	
Password	Db	Db	Db	Db
		Table_name	Table_name	
			Column_name	

Таблица 12.2. Столбцы привилегий таблицы разрешений

Столбцы административных привилегий		
user Table	db Table	host Table
Create_tmp_table_priv	Create_tmp_table_priv	Create_tmp_table_priv
Execute_priv		
File_priv		
Grant_priv	Grant_priv	Grant_priv
Lock_tables_priv	Lock_tables_priv	Lock_tables_priv
Process_priv		
Reload_priv		
Repl_client_priv		
Repl_slave_priv		
Show_db_priv		
Shutdown_priv		
Super_priv		
Столбцы привилегий баз данных/таблиц		
user Table	db Table	host Table
Alter_priv	Alter_priv	Alter_priv
Create_priv	Create_priv	Create_priv
Delete_priv	Delete_priv	Delete_priv
Drop_priv	Drop_priv	Drop_priv
Index_priv	Index_priv	Index_priv
Insert_priv	Insert_priv	Insert_priv
References_priv	References_priv	References_priv
Select_priv	Select_priv	Select_priv
Update_priv	Update_priv	Update_priv
tables_priv Table	columns_priv Table	
Table_priv	Column_priv	

Таблица 12.3. Столбцы управления ресурсами и SSL (только таблица user)

Столбцы SSL	Столбцы управления ресурсами
ssl_type	max_connections
ssl_cipher	max_questions
x509_issuer	max_updates
x509_subject	

Система таблиц разрешений включает таблицы tables_priv и columns_priv для настройки привилегий доступа к таблицам и столбцам. Отсутствие таблицы row_priv объясняется невозможностью предоставления в MySQL привилегий на уровне строк. Нельзя, например, предоставить пользователю доступ только к отдельным строкам таблицы. Обеспечить подобную возможность реально можно только посредством написания соответствующей программы. Для блокировки данных на уровне записей можно использовать функцию GET_LOCK(), описанную в приложении B, "Операторы и функции".

Изменение структуры таблицы разрешений

Структура таблиц разрешений со временем изменяется, поэтому если вы взглянете на таблицы в базе данных вашего сервера, то увидите, что некоторые таблицы или столбцы отсутствуют.

- Таблицы `tables_priv` и `colums_priv` впервые появились в MySQL 3.22.11 (вместе с оператором `GRANT`). Поэтому администраторы старых версий MySQL могут найти в базе данных `mysql` своего сервера только таблицы `user`, `db` и `host`.
- Столбцы `SSL` в таблице `user` появились в MySQL 4.0.0.
- Столбцы управления ресурсами в таблице `user` появились в MySQL 4.0.2.
- Несколько столбцов привилегий в таблице `user` появились в MySQL 4.0.2. Это столбцы `Create_tmp_table_priv`, `Execute_priv`, `Lock_tables_priv`, `Repl_client_priv`, `Repl_slave_priv`, `Show_db_priv` и `Super_priv`. В MySQL 4.0.4 в таблицы `db` и `hosts` были добавлены столбцы `Create_tmp_table_priv` и `Lock_tables_priv`.

Если структура вашей таблицы разрешений не совпадает с описанием, вы можете обновить их, запустив сценарий `mysql_fix_privileges_tables`. Этому сценарию необходимо подключиться к серверу как пользователю `root`, поэтому команда задается с указанием пароля пользователя `root`:

```
% mysql_fix_privilege_tables root-password
```

Если вы обновили MySQL до версии 4.0.2, новые столбцы привилегий, добавленные в таблицу `user` сценарием `mysql_fix_privilege_tables`, инициализированы для существующих неанонимных записей таблицы `user`.

- Столбцу `Show_db_priv` присвоено существующее значение столбца `Select_priv`.
- Столбцам `Execute_priv` и `Super_priv` присвоено существующее значение столбца `Process_priv`.
- Столбцам `Repl_client_priv` и `Repl_slave_priv` присвоено существующее значение столбца `File_priv`.
- Столбцам `Create_tmp_table_priv` и `Lock_tables_priv` присвоено значение "`Y`".

Вообще, лучше обновить сервер до версии 4.0.4 или выше, чем до версии 4.0.2 или 4.0.3. Отсутствие столбцов `Create_tmp_table_priv` и `Lock_tables_priv` в таблице `db` в этих версиях приводит к невозможности обеспечения доступа на уровне баз данных, что вызывает определенные трудности в администрировании.

Столбцы области применения таблицы разрешений

Столбцы области применения таблицы разрешений определяют, когда применяются записи таблиц. Каждый элемент таблицы разрешений включает значения столбцов `User` и `Host`. Эти значения определяют, что данный элемент применяется при подключении определенного пользователя к определенному компьютеру. Например, запись таблицы `user` со значениями столбцов `Host`, `User` `localhost` и `bill` будет использоваться для подключения пользователя `bill`, но никак не пользователя `betty`. (Таблица `Host` является исключением в этом отношении. Она используется особым способом, который в данном разделе не рассматривается.) Кроме того, другие таблицы могут содержать дополнительные столбцы. Таблица `db`, например, включает также столбец `Db`, определяющий базу данных, к которой применяется запись. Аналогичным образом таблицы `tables_priv` и `columns_priv` содержат поля области, которые ограничивают область применения записи до уровня отдельной таблицы базы данных или отдельного столбца таблицы.

Столбцы привилегий таблицы разрешений

Таблицы разрешений включают также столбцы привилегий. Именно они определяют, какие привилегии предоставляются пользователю, описанному в

столбцах области. Поддерживаемые сервером MySQL привилегии приведены в списке, представленном ниже. В этом списке указываются названия привилегий, используемые в строке оператора GRANT. В большинстве случаев названия столбцов привилегий таблиц user, db и host сходны. Столбец Select_priv, например, соответствует привилегии SELECT.

Административные привилегии

Следующие привилегии позволяют выполнять административные операции, управляющие функционированием сервера или предоставляющие возможность присваивать привилегии другим пользователям.

- CREATE TEMPORARY TABLES. Позволяет создавать временные таблицы с помощью оператора CREATE TEMPORARY TABLE.
- EXECUTE. Позволяет выполнять хранимые процедуры. В настоящее время не поддерживается. Будет работать в последующих версиях MySQL (MySQL 5) при наличии поддержки хранимых процедур.
- FILE. Позволяет дать серверу задание на считывание или запись файлов. Эту привилегию рекомендуется присваивать лишь в особых случаях, поэтому сервер также предпринимает определенные меры предосторожности, позволяющие установить границы применения этой привилегии.
  - Так, пользователи могут считывать файлы, которые общедоступны для чтения. Таким образом их нельзя считать сколько-нибудь защищенными.
  - Невозможно записать файл, которого еще не существует на диске. Это позволяет избегать путаницы с критически важными файлами сервера, например /etc/passwd, или файлами чужих баз данных. (Отсутствие подобного ограничения может привести к полной замене содержимого таблиц разрешений базы данных mysql.)

  Исходя из этих предосторожностей, эта привилегия не должна представляться просто так. Она таит в себе опасность. Подробно об этом рассказывается в разделе "Как минимизировать риск при работе с таблицами разрешений". Присваивая привилегию FILE, следует обязательно убедиться, что запущенный сервер не обладает правами UNIX-пользователя root, который может создавать новые файлы в любом каталоге файловой системы. При запуске сервера под управлением учетной записи пользователя без особых привилегий создавать файлы можно только в доступных для такого пользователя каталогах.
- GRANT OPTION. Позволяет предоставить другому пользователю привилегии, имеющиеся у администратора, включая саму возможность присвоения привилегий.
- LOCK TABLES. Позволяет блокировать таблицы операторами LOCK TABLES. Эта привилегия применима только к таблицам с привилегией SELECT, но позволяет блокировать чтение и запись в таблицу, а не только чтение. Привилегия не распространяется на блокировку, установленную во время выполнения запроса. Такого рода блокировки устанавливаются и снимаются независимо от настроек вашей привилегии LOCK TABLES.

- PROCESS. Поскольку сервер MySQL является многопотоковым, каждому клиентскому соединению предоставляется отдельный поток. Потоки можно еще представить и как процессы сервера. В MySQL версии 4.0.2 привилегия PROCESS позволяет просматривать информацию о выполняемых потоках с помощью оператора SHOW PROCESSLIST или команды processlist пользователя mysqladmin. Она также позволяет вам останавливать потоки оператором KILL или командой kill пользователя mysqladmin. (Просмотреть информацию о своих потоках или остановить их можно без привилегии PROCESS. Эта привилегия дает право управлять любыми потоками, в том числе потоками других пользователей.) Привилегия PROCESS также позволяет выполнять команду debug и при подключении к серверу аннулирует параметр max_connections, благодаря чему вы сможете подключиться к порту, который зарезервирован для административных подключений, если все обычные порты заняты. В MySQL версии 4.0.2 или более поздней привилегия PROCESS позволяет только просматривать потоки. Возможность останавливать их обеспечивается привилегией SUPER, так же как и возможность использовать mysqladmin-команду debug и зарезервированный порт подключения.
- RELOAD. Позволяет выполнять множество операций администрирования сервера, например запускать SQL-оператор FLUSH и RESET или выполнять такие mysqladmin-команды, как reload, refresh, flush-hosts, flush-logs, flushprivileges, flush-status, flush-tables и flush-threads.
- REPLICATION CLIENT. Позволяет запрашивать местонахождение главного и подчиненного серверов.
- REPLICATION SLAVE. Позволяет клиенту подключаться к главному серверу и запрашивать обновления подчиненного сервера. Предоставляется учетным записям, используемым подчиненными серверами. В MySQL до версии 4.0.2 вместо нее используется привилегия FILE.
- SHOW DATABASES. Позволяет использовать оператор SHOW DATABASES.
- SHUTDOWN. Позволяет завершать работу сервера с помощью команды mysqladmin shutdown.
- SUPER. Позволяет останавливать потоки сервера, использовать команду mysqladmin debug и получать доступ к зарезервированному порту, как было сказано в описании привилегии PROCESS. Также позволяет использовать операторы CHANGE MASTER, PURGE MASTER LOGS и SET для изменения глобальных переменных сервера и уровня изоляции глобальных транзакций. Привилегия SUPER позволяет использовать дешифрование DES, основанное на ключах, хранящихся в ключевом файле DES.

Привилегии баз данных и таблиц

Для работы с базами данных и таблицами применяются следующие привилегии.

- ALTER. Позволяет использовать оператор ALTER TABLE. Для обработки таблиц необходимо предоставить дополнительные привилегии, в зависимости от того, какие операции необходимо выполнить.
- CREATE. Позволяет создавать базы данных и таблицы, но не индексы, кроме тех, которые были изначально объявлены оператором CREATE TABLE.

- DELETE. Позволяет удалять записи из таблиц.
- DROP. Позволяет удалять таблицы и базы данных, но не индексы.
- INDEX. Позволяет создавать и удалять индексы в таблице.
- INSERT. Позволяет вставлять новые записи в таблицы.
- REFERENCES. В настоящее время не используется. Может использоваться для определения того, кто может настроить внешние ключи.
- SELECT. Позволяет извлекать данные из таблиц с помощью операторов SELECT. Эту привилегию необязательно присваивать для исполнения операторов SELECT, не связанных с таблицами, например SELECT NOW() или SELECT 4/2.
- UPDATE. Позволяет изменять записи таблицы.

Некоторые операции требуют комбинации привилегий. Например, оператор REPLACE может вызвать оператор DELETE, а затем оператор INSERT, поэтому требуются привилегии DELETE и INSERT.

Структура столбцов привилегий

В таблицах user, db и host каждая привилегия определена в отдельном столбце. Все эти столбцы описаны типом ENUM("N", "Y"), и по умолчанию каждая привилегия имеет значение "N" (отключена). Например, столбец Select_priv определяется так:

```
Select_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N'
```

Привилегии таблиц tables_priv и column_priv представлены типом SET, благодаря чему привилегии могут определяться различными комбинациями в одном столбце. Например, столбец Table_priv таблицы tables_priv определяется следующим образом:

```
SET('Select','Insert','Update','Delete','Create','Drop',
'Grant','References','Index','Alter')
```

Столбец Column_priv таблицы column_priv определяется так:

```
SET('Select','Insert','Update','References')
```

Привилегий столбцов гораздо меньше, чем привилегий таблиц, поскольку на уровне столбца можно выполнить гораздо меньше операций. Так, например, можно создать новую таблицу, но невозможно создать отдельный столбец.

Таблицы tables_priv и columns_priv новее остальных трех, поэтому они используют более эффективный тип SET. (Возможно, таблицы user, db и host в будущем реорганизуют для использования столбцов SET.)

Таблица user, помимо всего прочего, содержит несколько привилегий, которые отсутствуют во всех других таблицах разрешений: File_priv, Process_priv, Reload_priv и Shutdown_priv. Эти привилегии применяются к выполняемым сервером операциям, не связанным с отдельной базой данных или таблицей. Ведь при необходимости завершения работы сервера необязательно проверять, над какой базой данных в настоящее время ведется работа.

SSL-столбцы таблицы разрешений

Несколько столбцов таблицы user отвечает за безопасные соединения в соответствии со спецификациями протокола SSL. Основным столбцом является

ssl_type, который указывает, требуется ли безопасное соединение и каков тип такого соединения. Он представлен в виде типа ENUM с четырьмя возможными значениями.

- 'NONE'. Значение по умолчанию. Указывает, что безопасные соединения не требуются. Используется при настройке учетной записи, когда не определяется оператор REQUIRE, или при использовании оператора REQUIRE NONE.
- 'ANY'. Указывает, что безопасное соединение требуется, но тип соединения не важен. В своем роде – это "общее" требование. Значение устанавливается при указании REQUIRE SSL в операторе GRANT.
- 'X509'. Указывает, что требуется соединение, использующее сертификат X509. Значение устанавливается при указании REQUIRE X509.
- 'SPECIFIED'. Указывает, что безопасное соединение должно соответствовать указанным требованиям. Значение устанавливается при указании любой комбинации значений ISSUER, SUBJECT или CIPHER оператора REQUIRE.

При любых значениях столбца ssl_type, кроме 'SPECIFIED', сервер игнорирует другие SSL-столбцы при попытках пользователей установить соединение. При значении 'SPECIFIED' сервер проверяет остальные столбцы, и, если находятся столбцы без пустых значений, пользователь обязан предоставить соответствующие данные. Ниже приведены эти столбцы.

- ssl_cipher. Если столбец непустой, в нем указывается метод шифрования, который должен использоваться клиентом при подключении.
- x509_issuer. Если столбец непустой, в нем указывается значение эмитента, которое должно присутствовать в сертификате X509, предоставленном клиентом.
- x509_subject. Если столбец непустой, в нем указывается значение темы (subject), которое должно присутствовать в сертификате X509, предоставленном клиентом.

Столбцы ssl_cipher, x509_issuer и x509_subject представлены в таблице user в виде столбцов типа BLOB.

Столбцы управления ресурсами таблицы разрешений

Столбцы управления ресурсами в таблице user позволяют ограничивать меру использования ресурсов сервера любой учетной записью.

- max_connections. Количество возможных подключений учетной записи к серверу в час.
- max_questions. Количество запросов в час.
- max_updates. То же, что и max_questions, но ограничивает только запросы, модифицирующие данные.

Для каждого столбца значение "0" означает отсутствие ограничений.

При перезапуске сервера счетчики обновляются. Обновление происходит также при перезагрузке таблиц разрешений или выполнении оператора FLUSH USER_RESOURCES.

Как сервер управляет доступом клиентов

Сервер MySQL выполняет контроль за доступом клиентов в два этапа. Первый этап имеет место при попытке подключения, когда сервер "заглядывает" в таблицу `user` и ищет все соответствующие имени пользователя записи, например название компьютера, с которого можно подключаться, и пароль. Если ни одна запись с аналогичными значениями не найдена, пользователю отказывается в подключении. Если соответствующие записи найдены и таблица `user` содержит столбцы SSL или столбцы управления ресурсами, сервер также проверяет эти столбцы.

- Если пользователь превысил разрешенное количество подключений, ему будет отказано в подключении.
- Если запись в таблице `user` требует безопасные соединения, сервер проверяет, соответствует ли предоставленный пользователем сертификат требованиям столбцов SSL. Если нет, ему отказывается в подключении.

Если же вся предоставленная пользователем информация соответствует информации, записанной в таблице, соединение устанавливается и сервер переходит ко второму этапу. Для безопасных соединений используется шифрование. В процессе этого этапа каждый раз при получении от пользователя запроса сервер обращается к таблицам разрешений и проверяет, обладает ли пользователь достаточными привилегиями для выполнения того или иного запроса. (Если в таблице `user` присутствуют столбцы управления ресурсами, сервер проверяет ограничения на запросы и обновления. Он выполняет это перед проверкой привилегий, так как нет смысла их проверять, если ограничения были превышены.) Второй этап продолжается до завершения сеанса работы с сервером.

Настоящий раздел посвящен описанию правил, используемых сервером MySQL для проверки соответствия записей таблиц разрешений с входящими запросами клиента. В нем описываются типы значений, применяемые в столбцах области таблиц разрешений, способы комбинирования информации; рассказывается о привилегиях в таблицах разрешений и порядке проверки записей таблицы.

Содержимое столбцов области доступа

В некоторых столбцах области доступа необходимо вводить буквенные значения, однако в большинстве из них можно использовать также специальные символы.

- `Host`. Значение столбца `Host` может представлять собой имя компьютера или его IP-адрес. Значение `localhost` соответствует локальному компьютеру. Запись будет совпадать, если вы будете использовать значение `localhost` или `127.0.0.1` либо если вы подключитесь с помощью именованного канала в системах Windows NT со значением '.'.

Однако запись `localhost` не будет соответствовать при подключении с использованием настоящего имени компьютера или его IP-адреса. Представим ситуацию, когда локальный компьютер имеет имя `cobra.snake.net` и для пользователя bob имеются две записи в таблице `user`. Одна запись в столбце `Host` имеет значение `localhost`, а вторая — `cobra.snake.net`. Запись `localhost` будет использоваться при подключении любым из этих способов в системе UNIX или Windows:

```
% mysql -p -u bob -h localhost
% mysql -p -u bob -h 127.0.0.1
```

В системе Windows запись также будет соответствовать, если пользователь bob подключится таким образом:

```
C:\> mysql -p -u bob -h .
```

Соединение с использованием записи `localhost` будет использовать файл разъема в системе UNIX и протокол TCP/IP в системе Windows. Соединение с использованием записи `127.0.0.1` на обеих платформах использует протокол TCP/IP. Соединение "." в системе Windows будет использовать именованный канал.

Запись `Host` со значением `cobra.snake.net` будет работать, если пользователь bob подключится с компьтера с именем `cobra.snake.net`, используя имя сервера или IP-адрес. В обоих случаях будет использоваться протокол TCP/IP.

Значения столбца `Host` также можно задать с помощью шаблона. Так, можно использовать распространенные в SQL групповые символы "`%`" и "`_`", имеющие такое же значение, как и в строке запроса с оператором `LIKE`. (`REGEX`-шаблоны для этих целей неприменимы.) Символы шаблона SQL можно применять при определении как имен, так и IP-адресов. Строке `%.kitebird.com`, например, соответствуют все компьютеры домена `kitebird.com`, а строке `%.edu` — компьютеры всех подключенных к сети организаций системы образования. Аналогичным образом, строка `192.168.%` описывает все компьютеры подсети класса B 192.168, а строка `192.168.3.%` — все компьютеры подсети класса C 192.168.3.

Значение "`%`" определяет все компьютеры и дает возможность пользователю подключиться из любой точки. Пустое значение столбца `Host` аналогично значению "`%`". (Исключение: в таблице `db` пустое значение столбца `Host` указывает на необходимость проверки таблицы `host` для получения более детальной информации. Этот процесс подробно описывается в разделе "Проверка запроса".)

В версиях MySQL серии 3.23 можно также задавать IP-адреса с помощью маски сети, отражающей разрядность сетевого номера. Строка `192.168.128.0/255.255.255.0`, например, определяет 24-разрядный сетевой номер. Ей соответствует любой компьютер, первые 24 разряда IP-адреса которого аналогичны разрядам адреса `192.168.128`.

- `User`. Имена пользователей могут задаваться буквенными или пустыми значениями. В последнем случае сможет подключиться любой пользователь. Значение "`%`" и пустое значение в столбце `User` — это не одно и то же. Оно соответствует пользователю с именем "`%`". Надо признать, такое имя встречается довольно редко.

  Если при проверке входящего соединения пользователю соответствует в таблице `User` запись с пустым значением столбца `User`, данный клиент рассматривается как анонимный.

- `Password`. Значения этого столбца могут быть либо пустыми, либо буквенными. Специальные символы в них использовать нельзя. Пустое значение не является аналогией "любого пароля". Когда в таблице определено пустое значение столбца `Password`, пользователь не должен определять пароль вовсе.

  Значения паролей хранятся в зашифрованном, а не в обычном текстовом виде. Пользователи не смогут подключиться к серверу, если в столбец

Password записываются обычные текстовые значения! Оператор GRANT и команда mysqladmin password автоматически шифруют пароль. Поэтому при непосредственной записи паролей с помощью команд INSERT, REPLACE, UPDATE и SET PASSWORD следует зашифровать их значения с помощью команды PASSWORD("new_password").

- Db. В таблицах db и host значения можно определить как в виде текстовых значений, так и с помощью SQL-символов "%" и "_". Пустое значение и значение % позволяют получить доступ к любой базе данных. Значения столбца Db в таблицах columns_priv и tables_priv обязательно должны представлять собой текстовые имена баз данных. Использование специальных символов и пустых значений запрещается.
- Table_name, Column_name. Значения этих столбцов представляют собой текстовые записи имен таблиц и столбцов. Использование специальных символов и пустых значений запрещается.
- Регистр символов в некоторых столбцах области имеет важное значение, а в остальных столбцах значения могут быть записаны как строчными, так и прописными буквами. Регистры символов в таких столбцах воспринимаются сервером по-разному (табл. 12.4). Необходимо заметить, что регистр значений столбца Table_name всегда имеет значение, хотя учет регистра в именах таблиц в запросах зависит от файловой системы, под управлением которой работает сервер (под управлением UNIX выбор регистра важен, а под управлением Windows — нет).

Таблица 12.4. Учет регистра в столбцах области таблиц разрешений

Столбец	Учет регистра
Host	Нет
User	Да
Password	Да
Db	Да
Table_name	Да
Column_name	Нет

Способы хранения паролей в таблице user

Перед помещением паролей в таблицу user сервер MySQL шифрует их функцией PASSWORD(), чтобы предотвратить отображение их в виде простого текста даже для пользователей, имеющих право на чтение этой таблицы. Казалось бы, что функция PASSWORD() использует тот же вид шифрования, что используется при шифровании паролей UNIX, но это не так. Эти два вида шифрования объединяет то, что они однонаправлены и данные не подлежат дешифровке. MySQL использует алгоритм, отличный от UNIX. Это значит, что если вы будете использовать пароль UNIX в качестве пароля MySQL, не рассчитывайте, что строки шифрованных паролей совпадут. Если вы захотите использовать в программах шифрование UNIX, используйте функцию CRYPT(). (Если вам интересно, какие возможности шифрования доступны для использования в программах, обратитесь к приложению Б, "Типы данных".)

Проверка запроса

Каждый раз при получении запроса сервер проверяет в таблицах разрешений, обладает ли пользователь, отправивший запрос, достаточными полномочиями

для его исполнения. Ограничения ресурсов накладываются значениями столбцов max_questions и max_updates таблицы user, при условии, что ваша версия MySQL достаточно новая и в ней присутствуют эти столбцы. Для этих целей таблицы user, db, tables_priv и column_priv просматриваются в установленном порядке до тех пор, пока не будет определено, что доступ разрешен, либо не будут проверены все таблицы разрешений. Более детально этот процесс выглядит следующим образом.

1. При первоначальном подключении сервер проверяет записи таблицы user, пытаясь определить глобальные привилегии подключающегося пользователя. Если таковые имеются и, более того, являются достаточными для выполнения запроса, сервер обрабатывает запрос.
2. Если имеющихся глобальных привилегий недостаточно, сервер ищет дополнительные привилегии в таблице db. Если найденные привилегии позволяют выполнить запрос, сервер его обрабатывает.
3. Если набора глобальных привилегий и привилегий баз данных недостаточно, сервер продолжает поиск сначала в таблице tables_priv, а потом и в columns_priv.
4. Если после проверки всех таблиц разрешений достаточных полномочий не найдено, сервер отклоняет попытку выполнения запроса.

Говоря на языке булевой алгебры, привилегии таблиц разрешений используются сервером следующим образом:

```
user OR db OR tables_priv OR columns_priv
```

Некоторые читатели уже, возможно, задаются вопросом, почему в этой записи указываются четыре таблицы разрешений, хотя на самом деле их пять. Совершенно верно. Правильней было бы переписать ее следующим образом:

```
user OR (db AND host) OR tables_priv OR columns_priv
```

Такое простое выражение показано потому, что более сложное выражение никогда не будет использоваться во многих инсталляциях MySQL из-за того, что при использовании этого выражения операторы GRANT и REVOKE не оказывают действия на таблицу host. На нее действует только прямое применение операторов INSERT, UPDATE и им подобных. Это значит, что при использовании обычной административной политики управления пользовательскими учетными записями, в которой применяются операторы GRANT и REVOKE, ваша таблица host никогда не будет использоваться, и вы, возможно, совсем забудете о ней.

Однако, если вы хотите использовать таблицу host, ознакомьтесь с принципами ее работы.

1. Проверяя привилегии уровня базы данных, сервер ищет запись для клиента в таблице db. Пустое значение столбца Host означает примерно следующее: проверьте таблицу host, чтобы определить, какие компьютеры могут получить доступ к базе данных.
2. Сервер ищет в таблице host записи с таким же значением столбца Db, как и в записи таблицы db. Если компьютер подключающегося клиента не указан ни в одной записи таблицы host, следовательно, привилегии для

работы с таблицами этому пользователю не предоставлены. Если значение столбца Host хотя бы одной записи этой таблицы соответствует компьютеру подключающегося пользователя, записи таблиц db и host объединяются, что приводит к определению реальных привилегий уровня базы данных клиента.

Привилегии объединяются посредством логического оператора AND. Это означает, что пользователь будет иметь привилегии только в том случае, когда записи о нем будут представлены в обеих таблицах. Таким образом, пользователю может присваиваться базовый набор привилегий в записи таблицы db. Согласно записям таблицы host, получить их можно будет только с определенных компьютеров. Так, например, можно предоставить доступ к базам данных со всех компьютеров домена, но убрать эти привилегии для пользователей компьютеров, расположенных в менее защищенной области.

Из всего этого можно сделать вывод, что проверка привилегий — достаточно сложный процесс, особенно если учесть, что сервер проверяет привилегии при каждом получении запроса. Хотя на самом деле реализуется он весьма быстро, поскольку сервер не ищет информацию в таблицах разрешений для каждого запроса. Вместо этого содержимое таблиц записывается в память при загрузке, а данные затем быстро проверяются с помощью копий, занесенных в память. Такая организация работы повышает производительность операций проверки доступа, хотя имеет и обратную сторону — сервер не замечает изменения привилегий при непосредственном изменении таблиц разрешений.

Например, если новый пользователь создается администратором путем добавления новой записи в таблицу user с помощью оператора INSERT, он не сможет сразу же подключиться к серверу. Начинающих администраторов (а иногда и достаточно опытных) это зачастую весьма сильно раздражает, хотя решение этой проблемы предельно просто: достаточно указать серверу перезагрузить таблицы разрешений сразу после внесения изменений. Это можно осуществить с помощью оператора FLUSH PRIVILEGES или команды mysqladmin flush-privileges либо mysqladmin reload.

Нет необходимости перезагружать таблицы разрешений при использовании операторов GRANT, REVOKE или SET PASSWORD для настройки или модификации учетной записи пользователя.

Сервер помещает эти операторы в операции изменения таблиц разрешений, а затем автоматически обновляет их копии, находящиеся в оперативной памяти.

Порядок сравнения столбцов

Сервер MySQL сортирует записи таблиц разрешений в определенном порядке, а затем, согласно этому порядку, ищет соответствующие соединениям данные. Первое найденное совпадение определяет используемую запись. Важно понимать, в каком порядке записи располагаются сервером MySQL, особенно в таблице user.

Этот момент мешает многим пользователям в понимании проблемы безопасности MySQL.

Считывая содержимое таблицы user, сервер сортирует ее записи согласно значениям столбцов Host и User. Доминирующая роль здесь принадлежит значению столбца Host (т.е. записи с одинаковыми значениями столбца Host размещаются ря-

дом, а затем упорядочиваются по значению User). Однако выполняемая сортировка *не является* лексикографической, или, скорее, является таковой лишь частично. Основной принцип – более высокий приоритет символьных значений по сравнению со специальными символами. Это означает, что если пользователь подключается к компьютеру boa.snake.net, а в таблице имеются записи со значениями boa.snake.net и %.snake.net, будет выбрана именно первая запись.

Аналогичным образом, значение %.snake.net имеет больший приоритет, чем %.net, которое, в свою очередь, более приоритетно, чем %. Точно так же обрабатываются и IP-адреса. Для пользователя, подключающегося с компьютера с IP-адресом 192.168.3.14, подходят все записи со значением Host, но они отсортированы в таком порядке приоритетности значений:

```
192.168.3.14
192.168.3.%
192.168.%
192.%
%
```

Головоломка с привилегиями

В этом разделе представлен особый случай, который показывает, как полезно понимание порядка сортировки таблицы user при установке соединений. В нем также показано решение проблемы, свойственной новым инсталляциям MySQL, если судить хотя бы по частоте ее проявления в списке адресатов. Например, администратор MySQL настраивает новую инсталляцию, включая настройку стандартной записи root и записи анонимных пользователей таблицы user. Хороший администратор присвоит пароли учетным записям root, а анонимные пользователи останутся без паролей. Теперь представьте, что администратору необходимо создать учетную запись для пользователя, который будет подключаться с нескольких компьютеров. Проще всего создать учетную запись, указав в операторе GRANT спецификатор имени компьютера с символом "%", таким образом, пользователь сможет подключаться откуда угодно:

```
GRANT ALL ON sampdb.* TO 'fred'@'%' IDENTIFIED BY 'cocoa';
```

Этим оператором администратор присваивает пользователю fred все привилегии для работы с базой данных sampdb и позволяет ему подключаться с любого компьютера. К сожалению, в результате пользователь fred сможет подключиться с любого компьютера, кроме самого сервера! Например, когда fred подключается с boa.snake.net, соединение проходит удачно:

```
% mysql -p -u fred -h cobra.snake.net sampdb
Enter password: cocoa
mysql>
```

Но если fred попробует подключиться с сервера cobra.snake.net, то ему будет отказано в подключении, несмотря на то, что от ввел правильный пароль:

```
% mysql -p -u fred sampdb
Enter password: cocoa
ERROR 1045: Access denied for user: 'fred@localhost' (Using password: YES)
```

Эта ситуация происходит, если в таблице user содержатся стандартные записи анонимных пользователей (записи с пустыми именами пользователей). В системе

UNIX эти записи создаются инициализационным сценарием mysql_install_db, а в Windows присутствуют в заранее инициализированной таблице user, которая поставляется вместе с дистрибутивами. В доступе отказывается потому, что при попытках сервера проверить имя и пароль пользователя fred после сортировки одна из анонимных записей оказывается как раз перед записью пользователя fred. Анонимная запись требует подключение без пароля, поэтому происходит несоответствие результатов.

Почему же так происходит? Чтобы понять это, необходимо разобраться, как сценарий mysql_install_db создает исходные таблицы разрешений и как сервер проверяет соответствие данных клиентских соединений и записей таблицы user. При инициализации баз данных в процессе работы mysql_install_db создает записи в таблице user со следующим значениями таблиц Host и User[1].

```
+-----------------+------+
| Host            | User |
+-----------------+------+
| localhost       | root |
| cobra.snake.net | root |
| localhost       |      |
| cobra.snake.net |      |
+-----------------+------+
```

Первые две записи позволяют пользователю root подключиться к локальному компьютеру, определив либо localhost, либо собственно имя компьютера. Вторые две записи разрешают анонимное соединение с локальным компьютером. При добавлении записи для пользователя fred таблица user принимает следующий вид.

```
+-----------------+------+
| Host            | User |
+-----------------+------+
| localhost       | root |
| cobra.snake.net | root |
| localhost       |      |
| cobra.snake.net |      |
| %               | fred |
+-----------------+------+
```

В процессе запуска сервера он считывает эти записи и сортирует их (сначала по имени компьютера, а затем по пользователям). В результате более определенные значения оказываются первыми, а менее определенные — последними.

```
+-----------------+------+
| Host            | User |
+-----------------+------+
| localhost       | root |
| localhost       |      |
| cobra.snake.net | root |
| cobra.snake.net |      |
| %               | fred |
+-----------------+------+
```

[1] Детальное описание изначальных записей таблицы user представлено в главе 11.

Две записи для компьютера localhost располагаются рядом. Запись для пользователя root размещается первой, поскольку она является более полной. Аналогичным образом и в таком же порядке размещаются и записи компьютера cobra.snake.net.

Все эти записи имеют в столбце Host буквенные значения безо всяких специальных символов, поэтому и располагаются перед записью для пользователя fred, включающей один специальный символ. Соответственно записи анонимных пользователей имеют приоритет перед записью пользователя fred.

В результате, когда пользователь fred пытается подключиться с компьютера localhost, одна из "пустых" записей располагается перед записью со значением "%" столбца Host. Пустое поле пароля в записи анонимного пользователя не соответствует паролю cocoa пользователя fred, поэтому в подключении отказывается. Одним из последствий этого является то, что пользователь fred может регистрироваться в системе, *совсем не указывая пароля*. Но в этом случае пользователь является анонимным, и ему не будут предоставлены все привилегии учетной записи fred@%.

Несмотря на то что очень удобно использовать специальные символы при создании учетной записи пользователя с возможностью подключаться с нескольких узлов, у пользователя могут возникнуть проблемы при подключении с локального узла, причиной которых будут анонимные учетные записи в таблице user. Как же решить эту проблему? Существуют два способа.

Во-первых, можно создать еще одну запись для пользователя fred со значением узла localhost:

```
GRANT ALL ON sampdb.* TO 'fred'@'localhost' IDENTIFIED BY 'cocoa';
```

После этого таблица user будет отсортирована так:

```
+-----------------+------+
| Host            | User |
+-----------------+------+
| localhost       | fred |
| localhost       | root |
| localhost       |      |
| cobra.snake.net | root |
| cobra.snake.net |      |
| %               | fred |
+-----------------+------+
```

Теперь, когда пользователь fred подключается с локального узла, запись со значениями localhost и fred расположится перед анонимными учетными записями. При подключении с любого другого узла будет использоваться запись со значениями "%" и "fred". Неудобство в двух учетных записях заключается в том, что если вы захотите предоставить пользователю привилегию или изменить пароль, вам придется делать это дважды.

Во-вторых, из таблицы user можно полностью удалить все анонимные записи. Для этого нельзя использовать оператор REVOKE, так как он только аннулирует привилегии и не удаляет записи из таблицы. Необходимо использовать оператор DELETE:

```
% mysql -u root mysql
mysql> DELETE FROM user WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

Порядок сортировки оставшихся записей будет выглядеть так:

```
+-----------------+------+
| Host            | User |
+-----------------+------+
| localhost       | root |
| cobra.snake.net | root |
| %               | fred |
+-----------------+------+
```

Теперь пользователь `fred` сможет подключиться с любого узла, так как уже нет мешающих записей. Вообще, для облегчения вашего администрирования я бы посоветовал удалять анонимные учетные записи из таблиц. По-моему, эти записи не имеют смысла и не избавляют от проблем, а создают новые.

Хотя представленный в этом разделе материал описывает одну специфическую ситуацию, он позволяет сделать следующие общие выводы. Если присвоенные пользователям привилегии функционируют не так, как того ожидает администратор, необходимо заглянуть в таблицы разрешений и посмотреть, нет ли в них записей, значения столбца `Host` которых являются более определенными, чем введенные администратором. Именно в этом зачастую и кроется причина проблемы. Это свидетельствует также и о том, что администраторам необходимо более точно определять записи пользователей (добавляя новые записи для каждого конкретного уровня доступа).

Как минимизировать риск при работе с таблицами разрешений

В этом разделе приведены некоторые рекомендации, которые следует учитывать при присвоении привилегий, а также описываются проблемы, связанные с неправильным выбором администратора. Избегайте создания анонимных учетных записей. Даже если у них нет привилегий, способных причинить вред, позволяя пользователю подключиться, вы можете предоставить доступ пользователю, который будет собирать информацию о ваших базах данных и таблицах.

Отыщите учетные записи без паролей и либо удалите их, либо назначьте. Для этого используйте такой запрос в базе данных `mysql`:

```
mysql> SELECT Host, User FROM user WHERE Password = '';
```

Если это не вызвано острой необходимостью, не используйте шаблоны в спецификаторах имен компьютеров при создании учетных записей. Увеличение количества узлов, с которых пользователь сможет подключиться, также увеличивает шансы установления соединения мошенником, выдающим себя за этого пользователя.

Необходимо весьма аккуратно присваивать глобальные привилегии суперпользователей. Другими словами, не следует активизировать привилегии в записях таблицы `user`. Эти привилегии глобальны и позволяют воздействовать на функционирование всего сервера и получать доступ ко всем таблицам любой базы данных. Используйте с этой целью другие таблицы, чтобы ограничить область применения привилегий отдельными базами данных, таблицами и столбцами.

Не предоставляйте никаких привилегий для доступа к базе данных `mysql`. Ведь пользователь, обладающий привилегиями доступа к базе данных с таблицами разрешений, может легко изменить свои привилегии и получить доступ и к

другим базам данных. Присвоение пользователю достаточных привилегий для изменения таблиц базы данных mysql дает такому пользователю и привилегию GRANT OPTION. Возможность непосредственного изменения таблицы практически равноценна возможности запуска оператора GRANT.

Будьте осторожны с привилегией GRANT OPTION. Два пользователя с разными привилегиями и при условии, что им обоим предоставлена привилегия GRANT OPTION, смогут расширить свои права.

Привилегию FILE также рекомендуется присваивать с большой осторожностью. Так, обладающий этой привилегией пользователь может запросто выполнить следующие операторы:

```
CREATE TABLE etc_passwd (pwd_entry TEXT);
LOAD DATA INFILE '/etc/passwd' INTO TABLE etc_passwd;
```

Выполнив эти операторы, пользователь сможет получить доступ к файлу паролей сервера, выполнив оператор SELECT:

```
SELECT * FROM etc_passwd;
```

Имя любого общедоступного файла на узле сервера может быть заменено на /etc/passwd в операторе LOAD DATA. Фактически наличие привилегии FILE дает возможность пользователю раскрыть содержимое *любого* читаемого файла сервера, даже работая через сеть. Кроме того, привилегия FILE может использоваться для повреждения баз данных в системах, работающих с недостаточными файловыми правами. Именно поэтому администратор должен сделать содержимое каталога данных читаемым только для сервера. Если файлы таблиц баз данных открыты для всеобщего обзора, прочитать их сможет не только пользователь учетной записи сервера, но и любой другой пользователь, обладающий привилегией FILE и подключающийся к серверу через сеть.

Как это реализуется, показано в следующем примере.

1. Создайте таблицу со столбцом LONGBLOB.

   ```
   USE test;
   CREATE TABLE tmp (b LONGBLOB);
   ```

2. Используйте эту таблицу для считывания содержимого каждого файла всех нужных таблиц. Например, у пользователя есть таблица MyISAM с именем x в базе данных other_db. Таблица представлена тремя файлами: x.frm, x.MYD и x.MYI. Вы можете прочесть эти файлы и скопировать их в соответствующие файлы базы данных test:

   ```
   LOAD DATA INFILE './other_db/x.frm' INTO TABLE tmp
      FIELDS ESCAPED BY '' LINES TERMINATED BY '';
   SELECT * FROM tmp INTO OUTFILE 'x.frm'
      FIELDS ESCAPED BY '' LINES TERMINATED BY '';
   DELETE FROM tmp;
   LOAD DATA INFILE './other_db/x.MYD' INTO TABLE tmp
      FIELDS ESCAPED BY '' LINES
   ERMINATED BY '';
   SELECT * FROM tmp INTO OUTFILE 'x.MYD'
      FIELDS ESCAPED BY '' LINES TERMINATED BY '';
   DELETE FROM tmp;
   LOAD DATA INFILE './other_db/x.MYI' INTO TABLE tmp
      FIELDS ESCAPED BY '' LINES TERMINATED BY '';
   SELECT * FROM tmp INTO OUTFILE 'x.MYI'
      FIELDS ESCAPED BY '' LINES TERMINATED BY '';
   ```

3. После выполнения этих операторов в каталоге базы данных `test` такж дут находиться файлы с именами `x.frm`, `x.MYD` и `x.MYI`. Другими слов база данных `test` будет содержать таблицу `x`, которая является украден копией таблицы из базы данных `other_db`.

Чтобы избежать подобных атак, установите права доступа к содержимом талога данных в соответствии с инструкциями, представленными в раз "Внутренняя безопасность: защита доступа к каталогу данных". Можно также запуске сервера использовать параметр `--skip-show-database`. Это запр пользователям запускать операторы `SHOW DATABASE` и `SHOW TABLES` для данных, к которым им запрещен доступ, и не позволит пользователям случа найти базу данных или таблицу, доступ к которым им не должен предоставлят

Опасность использования привилегии `FILE` увеличивается, если сервер My работает под управлением пользователя `root`. Работа под управлением пользова `root` опасна сама по себе, а в комбинации с использованием привилегии `FILE` эт разумно вдвойне, так как пользователь `root` может создавать файлы где угодно. П зователь с привилегией `FILE` также может создавать файлы, даже если он работа удаленном компьютере. Сервер не станет создавать уже существующий файл, но гда предоставляется возможность создать новые файлы, которые изменят работу вера или нарушат защиту. Например, если отсутствует любой из фай `/etc/resolv.conf`, `/etc/hosts.equiv`, `/etc/hosts.lpd` или `/etc/sudo` пользователь, который будет использовать сервер MySQL для их создания, м сильно повлиять на поведение компьютера, на котором установлен сервер. Чтоб бежать этих проблем, не запускайте `mysqld` под управлением пользователя (см. раздел "Запуск сервера MySQL непривилегированным пользователем" главы

Привилегия `PROCESS` должна быть предоставлена только доверенным по вателям. С привилегией `PROCESS` пользователь может использовать опер `SHOW PROCESSLIST` для просмотра текста запросов, выполняемых сервером. позволяет пользователю шпионить за другими пользователями и просматри личную информацию. Не предоставляйте пользователям привилегию `REL` если они в ней не нуждаются. Привилегия `RELOAD` позволяет выполнять о торы `FLUSH` и `RESET`, которые могут быть использованы не по назначению.

- Если вы настроили сервер на протоколирование обновлений или двои обновлений с использованием определенного количества файлов, ка оператор `FLUSH LOGS` создает новый файл журнала. Пользователь с пр легией `RELOAD`, который выполняет много операций обновления журн может спровоцировать создание сервером большого количества файло
- Пользователь с привилегией `RELOAD` может обмануть механизм управлени сурсами, перезагружая таблицы разрешений оператором `FLUSH PRIVIL` или `FLUSH USER_RESOURCES`, который сбрасывает счетчики в нуль.
- Неоднократно используемый оператор `FLUSH TABLES` заставляет се очищать кэш открытых таблиц, понижая его производительность и н зволяя воспользоваться его преимуществами. Оператор `RESET QU CACHE` оказывает то же действие на производительность, не позволяя тать с кэшем запросов.
- Оператор `RESET MASTER LOGS` заставляет главный сервер реплик удалять журналы двоичных обновлений, независимо от того, использу

они или нет, удаляя тем самым информацию, необходимую для поддержания целостности репликации.

Иногда и привилегия ALTER используется таким образом, о котором даже и не подозревают администраторы. Предположим, например, что администратор желает предоставить пользователю `user1` доступ к таблице `table1`, но не к таблице `table2`. Обладающий привилегией ALTER пользователь может легко обойти это ограничение, переименовав с помощью оператора ALTER TABLE таблицу `table2` в `table1`.

Настройка учетных записей без помощи оператора GRANT

Администраторы MySQL, работающие с версиями до 3.22.11, не могут использовать операторы GRANT (или REVOKE) для создания пользователей и настройки привилегий доступа. Для достижения тех же целей они могут непосредственно изменять содержимое таблиц разрешений. Гораздо проще это сделать тем администраторам, которые знают, каким образом оператор GRANT изменяет таблицы разрешений. По сути, те же операции можно выполнять и самостоятельно, используя операторы INSERT. Для изменения таблиц разрешений оператор GRANT выполняет следующие операции.

- В строке оператора GRANT определяется имя пользователя, имя компьютера и иногда пароль. В таблице `user` для пользователя создается запись, и соответствующие значения записываются в столбцы `User`, `Host` и `Password`. Если администратор определяет какие-либо глобальные привилегии оператором GRANT, они записываются в столбцы, определяющие привилегии соответствующей записи таблицы `user`.
- При определении привилегий уровня базы данных имя пользователя и компьютера заносятся соответственно в столбцы `User` и `Host` таблицы `db`. Имя таблицы, по которой предоставляются привилегии, записывается в столбец `Db`, а сами привилегии — в столбец привилегий.
- Подобные операции выполняются и при определении полномочий на уровне таблиц и столбцов. Для записи имени пользователя, компьютера, названия базы данных и, если необходимо, имя таблицы или столбца заносятся соответствующие значения в поля таблиц `tables_priv` и `columns_priv`. Присваиваемые привилегии записываются в столбцы привилегий.

Все описанные выше операции можно выполнить, не прибегая к помощи оператора GRANT. Не забывайте также, что после непосредственного изменения таблиц разрешений необходимо перезагрузить их. Иначе изменения окажутся незамеченными. Для повторной загрузки можно запустить команду `mysqladmin reload`[2]. Если вы забудете перезагрузить таблицы, то будете удивляться, почему сервер не делает то, что вы от него хотите.

---

[2] Кроме того, сервер перезагружает таблицы разрешений с помощью команды `mysqladmin flush-privileges` и оператора FLUSH PRIVILEGES, но так как вы не используете оператор GRANT из-за того, что у вас установлена старая версия сервера, которая не поддерживает его, в нем, вероятно, отсутствует команда `support mysqladmin flush-privileges` или оператор FLUSH PRIVILEGES.

Представленный ниже оператор GRANT создает суперпользователя со всеми привилегиями, включая возможность наделения привилегиями других пользователей.

```
GRANT ALL ON *.* TO 'ethel'@'localhost' IDENTIFIED BY 'coffee'
WITH GRANT OPTION;
```

Этот оператор создает запись для пользователя ethel@localhost в таблице user и активизирует для нее все привилегии. Те же привилегии можно присвоить и с помощью обыкновенного оператора INSERT:

```
INSERT INTO user VALUES('localhost','ethel',PASSWORD('coffee'),
'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
```

И это только один оператор INSERT! Более того, в зависимости от используемой версии MySQL, он может еще и не работать. Этот оператор предполагает, что таблица user содержит четырнадцать столбцов привилегий. Такое количество их было в момент введения поддержки оператора GRANT. Если структура таблиц разрешений была случайно изменена, число столбцов привилегий может несколько отличаться. Чтобы проверить количество столбцов в каждой таблице разрешений, воспользуйтесь оператором SHOW COLUMNS, а затем соответствующим образом измените операторы INSERT. Заметьте также, что оператор GRANT шифрует ваши пароли, чего не делает оператор INSERT, поэтому для шифрования паролей в операторах INSERT необходимо использовать функцию PASSWORD().

Следующий оператор GRANT также создает пользователя со статусом суперпользователя, однако только для одной привилегии:

```
GRANT RELOAD ON *.* TO 'flush'@'localhost' IDENTIFIED BY 'flushpass';
```

Этот оператор уже рассматривался в главе 11, "Общее администрирование MySQL", когда речь шла о создании пользователя flush. Равноценный предыдущей строке оператор INSERT в этом примере не настолько сложен, как в предыдущем. Ведь гораздо проще указать имена столбцов и определить значение только в одном столбце привилегий. Во всех других столбцах по умолчанию будет установлено значение "N":

```
INSERT INTO user (Host,User,Password,Reload_priv)
VALUES('localhost','flush',PASSWORD('flushpass'), 'Y');
```

Привилегии уровня базы данных присваиваются с помощью предложения ON db_name, а не ON *.*:

```
GRANT ALL ON sampdb.* TO 'boris'@'localhost' IDENTIFIED BY 'ruby';
```

Эти привилегии не являются глобальными, поэтому их не следует записывать в таблицу user. Запись в этой таблице создать нужно (чтобы пользователь смог подключиться), однако для определения привилегий доступа к базе данных необходимо создать запись также и в таблице db:

```
INSERT INTO user (Host,User,Password)
  VALUES('localhost', 'boris',PASSWORD('ruby'));
INSERT INTO db
  VALUES('localhost','sampdb','boris',
'Y','Y','Y','Y','Y','Y','N','Y','Y','Y');
```

Столбец со значением "N" соответствует привилегии GRANT OPTION. Для оператора GRANT уровня базы данных с окончанием WITH GRANT OPTION в этом столбце можно установить значение "Y".

Настройка привилегий уровня таблицы и столбца осуществляется посредством выполнения оператора INSERT для таблиц tables_priv и columns_priv. Безусловно, если версия MySQL не включает оператор GRANT, то она не включает и эти таблицы, поскольку появились они одновременно. Если же администратор по определенным причинам желает манипулировать данными таблиц tables_priv и columns_priv без помощи оператора GRANT, он должен проявлять осторожность, поскольку невозможно активизировать привилегии в отдельных столбцах. Необходимо установить в одном из столбцов, tables_priv.Table_priv или columns_priv.Column_priv, значение SET, состоящее из активизируемых привилегий. Например, чтобы предоставить пользователю привилегии SELECT и INSERT для таблицы, установите в столбце Table_priv соответствующей записи таблицы tables_priv значение "Select,Insert".

Для изменения привилегий пользователей, уже имеющих учетную запись MySQL, вместо INSERT используется команда UPDATE. Ее можно применять как для добавления, так и для удаления привилегий.

Чтобы полностью удалить пользователя и все его привилегии, удалите с помощью команды DELETE его записи из всех таблиц разрешений. Например, для удаления учетной записи mike@%.snake.net воспользуйтесь следующими операторами:

```
DELETE FROM user WHERE User = 'mike' AND Host = '%.snake.net';
DELETE FROM db WHERE User = 'mike' AND Host = '%.snake.net';
DELETE FROM host WHERE User = 'mike' AND Host = '%.snake.net';
```

При наличии таблицы tables_priv или columns_priv выполните такие операторы:

```
DELETE FROM tables_priv WHERE User = 'mike' AND Host = '%.snake.net';
DELETE FROM columns_priv WHERE User = 'mike' AND Host = '%.snake.net';
```

Если администратор желает избежать непосредственного изменения таблиц разрешений, можно воспользоваться сценариями mysqlaccess и mysql_setpermissions, входящими в дистрибутив MySQL.

Установка безопасных соединений

В системе UNIX MySQL версии 4 появилась поддержка безопасных, зашифрованных соединений по протоколу SSL. По умолчанию инсталляция с поддержкой SSL позволяет клиенту опционально запрашивать безопасные соединения. (Обыкновенное нешифрованное соединение работает с большей скоростью, тогда как шифрованное соединение защищено, но в какой-то мере медленнее, так как процедура шифрования требует дополнительных вычислительных ресурсов.) Администраторы могут с помощью оператора GRANT указать, что данной учетной записи *требуется* безопасное подключение.

Нет особой необходимости в использовании протокола SSL для подключений к локальному узлу по именованному каналу с помощью файла разъема UNIX или к IP-адресу 127.0.0.1 (сетевой интерфейс обратной петли). Преимущества протокола SSL проявляются, когда информация, которую вы передаете по сети, может быть перехвачена. Для того чтобы воспользоваться преимуществами поддержки SSL для шифрованных соединений между сервером и клиентскими программами, воспользуйтесь такой процедурой.

- Убедитесь, что таблица `user` содержит столбцы SSL.
- Убедитесь, что сервер и клиентские программы скомпилированы с поддержкой OpenSSL.
- Запустите сервер с ключами, указывающими местоположение сертификатов и ключевых файлов, которые необходимы для установки безопасного соединения.
- Для безопасного подключения клиентской программой запустите ее с ключами, указывающими местоположение сертификатов и ключевых файлов.

Далее детально описывается этот процесс.

Для использования протокола SSL для безопасных соединений таблица `user` базы данных `mysql` должна содержать столбцы SSL, описанные в разделе "SSL-столбцы таблицы разрешений". Если у вас установлена MySQL версии 4.0.0 и выше, ваша таблица `user` должна содержать эти столбцы. Если вы обновили более раннюю версию до MySQL 4.x, столбцы могут отсутствовать. Для обновления таблицы запустите сценарий `mysql_fix_privilege_tables`. Ваш дистрибутив MySQL должен поддерживать OpenSSL. Либо найдите исполняемый дистрибутив со встроенной поддержкой OpenSSL, либо скомпилируйте его из исходного кода. В последнем случае у вас должен быть установлен OpenSSL. Если его нет, зайдите на Web-узел по адресу `www.openssl.org`. После этого скомпилируйте MySQL, запустив команду `configure` с ключами `--with-vio` и `--with-openssl`. После запуска вашего сервера с включенной поддержкой OpenSSL убедитесь, что он поддерживает протокол SSL, подключившись с помощью клиента `mysql` и задав такой запрос:

```
mysql> SHOW VARIABLES LIKE 'have_openssl';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_openssl  | YES   |
+---------------+-------+
```

Если `YES` не отображается, значит, поддержка SSL была установлена неправильно.

После включения поддержки SSL сервер и его клиенты смогут безопасно соединяться. Для установки безопасного соединения с обеих сторон используются такие файлы.

- Сертификат, выданный бюро сертификации. Бюро сертификации является доверенной третьей стороной. Его сертификат используется для проверки подлинности сертификатов клиента и сервера. Довольно часто этот сертификат покупают у коммерческих организаций, однако вы можете сгенерировать свой собственный сертификат.
- Файл сертификата, который представляет одну сторону соединения другой.
- Ключевой файл, который используется для шифровки и дешифровки потока данных.

Сначала должны быть установлены сертификаты сервера и ключевые файлы. Каталог `ssl` дистрибутива `sampdb` содержит файлы заготовок, которые могут быть использованы для этого:

- `ca-cert.pem` — сертификат бюро сертификации;
- `server-cert.pem` — сертификат сервера;
- `server-key.pem` — публичный ключ сервера.

Скопируйте эти файлы в каталог данных вашего сервера, затем добавьте пару строк в группу `[mysqld]` файла настроек, например, такой как `/etc/my.cnf` в системе UNIX. Настройки должны указывать путь к сертификатам и ключевым файлам. Например, если `/usr/local/mysql/data` — это каталог данных, параметры должны выглядеть так:

```
[mysqld]
ssl-ca=/usr/local/mysql/data/ca-cert.pem
ssl-cert=/usr/local/mysql/data/server-cert.pem
ssl-key=/usr/local/mysql/data/server-key.pem
```

Вы можете поместить сертификаты и ключевые файлы куда угодно, но при условии, что этот каталог будет доступен только для сервера. После изменения файла настроек перезапустите сервер.

На данном этапе клиентские программы все еще подключаются к серверу по незашифрованным соединениям. Чтобы заставить клиентскую программу использовать безопасные соединения, необходимо иметь ключевые файлы и сертификаты и на клиентской стороне. Каталог `ssl` дистрибутива `sampdb` содержит все необходимые файлы. Вы можете использовать тот же самый сертификат бюро сертификации (`ca-cert.pem`); клиентский сертификат и ключевой файл называются `client-cert.pem` и `client-key.pem` соответственно. Скопируйте эти файлы в любой каталог вашей учетной записи, а затем добавьте несколько строк в файл `.my.cnf` в вашем корневом каталоге, чтобы указать клиентской программе их местонахождение. Предположим, при работе с клиентом будут использоваться шифрованные соединения. Для этого необходимо скопировать файлы SSL в корневой каталог пользователя `/u/paul` и добавить в файл `.my.cnf` следующие строки:

```
[mysql]
ssl-ca=/u/paul/ca-cert.pem
ssl-cert=/u/paul/client-cert.pem
ssl-key=/u/paul/client-key.pem
```

Аналогичным образом вы сможете настроить свою учетную запись. (Убедитесь, что ваш сертификат и ключевой файл доступны только вам.) После изменения файла `.my.cnf` запустите `mysql` и задайте запрос `\s` или команду `STATUS`. Строка `SSL` в выводе должна указывать шифрование соединения:

```
mysql> \s
--------------
./mysql Ver 12.10 Distrib 4.1.0-alpha, for apple-darwin5.5 (powerpc)
Connection id: 1
Current database:
Current user: root@localhost
SSL: Cipher in use is EDH-RSA-DES-CBC3-SHA
...
```

Для определения значений переменных состояния SSL можно использовать такой запрос:

```
SHOW STATUS LIKE 'Ssl%';
```

Присутствие SSL-параметров в группе [mysql] заставляет mysql использовать SSL-соединения по умолчанию. Если вы добавите комментарий или удалите их из файла настроек, mysql будет использовать простое, нешифрованное соединение. Можно игнорировать настройки, запустив mysql таким образом:

```
% mysql --skip-ssl
```

Если вы хотите использовать протокол SSL в других программах, его настройки в группе [mysql] можно скопировать в другие группы настроек программ. Однако, если вы поместите их в общую группу [client], это приведет к тому, что любая клиентская программа перестанет "понимать" протокол SSL. Поэтому это не очень удачное решение.

Вместо указания параметров протокола SSL в файле настроек вы можете воспользоваться командной строкой. Например, находясь в домашнем каталоге пользователя, можно вызвать клиент mysql следующим образом (вводите все в одной строке):

```
% mysql --ssl-ca=ca-cert.pem --ssl-cert=client-cert.pem
--ssl-key=client-key.pem
```

Однако этот способ обременителен, если его часто использовать.

До сих пор речь шла о количестве учетных записей, опционально использующих протокол SSL. Для определенной учетной записи можно запретить или, наоборот, разрешить использование нешифрованных соединений. Это можно делать как для новых учетных записей, так и для уже существующих.

Для создания такой новой учетной записи используйте оператор GRANT, но с предложением REQUIRE, что определяет условия, которым должно соответствовать соединение.

Допустим, вы хотите создать пользователя laura, который будет подключаться к серверу cobra.snake.net с компьютера rat.snake.net, чтобы получить доступ к базе данных finance. Если вы хотите, чтобы шифрованными были только эти соединения, используйте такой оператор:

```
GRANT ALL ON finance.* TO 'laura'@'rat.snake.net'
IDENTIFIED BY 'moneymoneymoney'
REQUIRE SSL;
```

Для большей безопасности используйте предложение REQUIRE X509. Тогда пользователь laura при подключении должен будет предоставить действительный пользовательский сертификат X509. (Файл будет указываться параметром ssl-cert.) Пока сертификат действителен, его содержимое не имеет значения. Чтобы применить более жесткие требования, используйте комбинацию значений CIPHER, ISSUER и SUBJECT в операторе REQUIRE. Значение CIPHER указывает на тип шифровки, на котором должно основываться ваше соединение. ISSUER или SUBJECT указывает на то, что пользовательский сертификат должен быть получен из определенного источника или должен предназначаться определенному получателю. Эти параметры сужают круг действительных сертификатов до сертификатов с определенным содержанием. Например, этот оператор GRANT требует определенного издателя пользовательского сертификата и указывает использование шифрования EXP1024-RC4-SHA:

```
GRANT ALL ON finance.* TO 'laura'@'rat.snake.net'
IDENTIFIED BY 'moneymoneymoney'
```

```
REQUIRE ISSUER '/C=US/ST=WI/L=Madison/O=sampdb/OU=CA/CN=sampdb'
CIPHER 'EXP1024-RC4-SHA';
```

Чтобы настроить уже существующую учетную запись на использование SSL-соединений, используйте оператор GRANT USAGE указанного ниже вида, где *require_options* определяет параметры протокола SSL.

```
GRANT USAGE ON *.* TO 'user_name'@'host_name' REQUIRE require_options;
```

GRANT USAGE ON *.* оставляет привилегии пользователя неизмененными, и оператор изменяет только атрибуты SSL.

Если учетной записью уже используется SSL и вы хотите отменить его использование, воспользуйтесь оператором GRANT USAGE вместе с предложением REQUIRE NONE:

```
GRANT USAGE ON *.* TO 'user_name'@'host_name' REQUIRE NONE;
```

Сертификатов и ключевых файлов, которые содержатся в дистрибутиве sampdb, вам вполне хватит для установки шифрованных соединений. Но поскольку они общедоступны, нельзя сказать, что такие соединения полностью безопасны. После проверки этими файлами работы протокола SSL желательно заменить их на файлы, сгенерированные вами. Инструкции по созданию собственных сертификатов и ключевых файлов находятся в файле ssl/README дистрибутива sampdb; кроме того, сертификат также можно купить.

13

Резервное копирование и восстановление баз данных

В этой главе...

Координация работы с сервером
Устранение неполадок
Восстановление таблиц и данных

В общем-то, сервер MySQL должен безо всяких проблем начать работать сразу же после инсталляции. Однако иногда проблемы все же возникают, и причины их появления могут быть самые разные, от внезапного отключения электричества до сбоя в работе аппаратных средств и некорректного завершения работы сервера MySQL (например, с помощью команды `kill -9` или из-за выключения компьютера). Многие из этих ситуаций администратор не в состоянии предусмотреть даже при большом желании, а результатом их появления может стать повреждение таблиц баз данных, вызываемое чаще всего незавершенной операцией записи или изменения.

В этой главе рассказывается, как минимизировать риск и быть в полной готовности, если непредвиденное все-таки случится. Здесь описаны приемы создания резервных копий баз данных, проведения операций проверок и восстановления таблиц. Кроме того, описываются процедуры копирования базы данных, имеющие целью пересылку базы данных на другой сервер, так как эта операция подобна операции резервного копирования.

Для того чтобы быть готовым к возникновению таких проблем в будущем, необходимо предпринять следующие меры.

- Используйте возможности автовосстановления сервера MySQL.
- Предусмотрите резервирование базы данных по определенному графику. Если случится наихудшее и произойдет катастрофический сбой системы,

резервные копии помогут вам в проведении мероприятий по восстановлению. Кроме того, следует активизировать двоичный журнал, который позволит сохранить информацию об операциях восстановления, сделанных после резервного копирования. Нагрузка на сервер, связанная с ведением журнала двоичного копирования, – мизерная (примерно 1%), поэтому она не накладывает никаких ограничений.

- Периодически проводите запланированное превентивное обслуживание. Проведение периодических процедур проверки таблиц поможет обнаружить проблемы, пока они не достигли катастрофических размеров.

Если все же, невзирая на все ваши усилия, произошло повреждение таблиц или случилась потеря данных, попробуйте различные варианты.

- Проверьте свои таблицы, а затем, если они повреждены, восстановите их. Незначительные повреждения можно устранить в помощью средств восстановления таблиц, предусмотренных MySQL.
- Если только проверок и восстановления таблиц для продолжения работы будет недостаточно, восстановите данные с помощью резервных копий и двоичных журналов. Сначала с помощью резервных копий восстановите таблицы по состоянию их на момент резервного копирования. После этого с помощью журналов восстановите все обновления, сделанные после резервного копирования. Таким образом можно восстановить состояние таблицы базы данных на момент сбоя.

В вашем распоряжении будут не только возможности самого сервера MySQL, но и несколько других возможностей, имеющихся в дистрибутиве MySQL.

- Во время старта сервер может проверять таблицы и исправлять многие найденные ошибки. Такую возможность трудно переоценить при перезапуске сервера после сбоя. Одни проверки производятся автоматически, в то время как другие активизируются только с помощью соответствующих параметров.
- Для создания резервных копий баз данных можно прибегнуть к помощи программ `mysqldump` и `mysqlhotcopy`.
- Сервер может выполнить некоторые типы операций по обслуживанию и восстановлению таблиц с помощью таких операторов SQL, как `CHECK TABLE` и `REPAIR TABLE`. Для выполнения этих операторов программа `mysqlcheck` также имеет интерфейс командной строки.
- Кроме того, для проверки таблиц на наличие ошибок можно использовать программы `myisamchk` и `isamchk`.

Одни из этих программ (`mysqlcheck` и `mysqldump`) работают совместно с сервером. Они подключаются к серверу и передают на него информацию о том, какой тип резервного копирования или проверок таблиц необходимо выполнить. При этом используются операторы SQL. Другие программы (`myisamchk` и `isamchk`) работают непосредственно с файлами, в которых хранятся таблицы. Однако, из-за того, что сервер тоже может работать с этими файлами, такие программы в действительности являются "конкурентами" серверу, и следует что-то

предпринять для того, чтобы предотвратить взаимовлияние. Например, если вы восстанавливаете таблицы с помощью myisamchk, необходимо сделать все, чтобы предотвратить доступ к файлу со стороны сервера, поскольку это может привести к гораздо более серьезным проблемам, чем те, с которыми вы столкнулись!

Необходимость взаимодействия с сервером становится еще более насущной при выполнении административных функций, описанных в этой главе (начиная с задач резервирования и заканчивая задачей выполнения восстановления таблиц). Поэтому повествование этой главы начинается с того, каким образом можно блокировать сервер. После этого речь пойдет о том, как встретить во всеоружии проблемы и использовать разные приемы восстановления.

Под управлением ОС UNIX операции, для которых требуется непосредственный доступ к табличным или другим файлам, расположенным в каталоге данных, должны выполняться при подключении администратора MySQL. Только это дает вам право на использование файлов. В этой книге именем такой учетной записи является mysqladm. Кроме того, к файлам может получить доступ пользователь root, но в этом случае необходимо убедиться в том, что все файлы, задействованные в операции, имеют одни и те же характеристики доступа и владельца.

Более полный список параметров, поддерживаемых описанными здесь программами, представлен в приложении Г, "Синтаксис SQL", и приложении Д, "Программы MySQL".

Взаимосвязь isamchk и myisamchk

В этой главе подробно описана программа myisamchk, предназначенная для проверки и восстановления таблиц MyISAM. Другой аналогичной программой является программа isamchk, которая во многом подобна программе myisamchk, но используется для работы с таблицами ISAM. Непосредственное описание программы isamchk здесь сведено к минимуму, поскольку, учитывая повышенную производительность и возможности таблиц MyISAM, таблицы ISAM больше не используются. Однако, если работать с таблицами ISAM все же необходимо, следует помнить, что обе программы настолько похожи, что большинство инструкций, имеющих отношение к программе myisamchk, которые читатель встретит в этой главе, можно применить и к программе isamchk. Кроме того, необходимо иметь в виду следующее.

- Все, что касается файлов данных и индексов .MYD и .MYI, когда речь идет о программе myisamchk, можно применять к файлам данных и индексов .ISD и .ISM для таблиц ISAM.
- Вы никогда не нанесете никакого вреда таблицам, если попытаетесь обработать их с помощью другой программы. В результате вы получите только предупреждающее сообщение. Например, ниже приведена командная строка, по которой isamchk должна проверить таблицы типа MyISAM, расположенные в текущем каталоге.

```
% isamchk *.MYI
```

Но isamchk работает только с таблицами ISAM, поэтому в результате будут получены только диагностические сообщения.

- Программа myisamchk поддерживает несколько параметров, с которыми не работает isamchk. Все они приведены в приложении Д, "Программы MySQL".

Координация работы с сервером

Некоторые типы административных действий выполняются посредством подключения к серверу и задания ему команд, определяющих выполняемые действия. Предположим, что вам необходимо провести какие-либо проверки на целостность или восстановление MyISAM-таблиц. Это можно сделать с помощью команды CHECK TABLE или REPAIR TABLE. В таком случае сервер обратится к

файлам .frm, .MYD и .MYI, представляющим таблицу. В целом это является наилучшим возможным способом. Таким образом, можно снять с себя все вопросы по координации доступа к таблицам и возложить их непосредственно на сервер. Это значит, что они стали неактуальны и вам нет необходимости заботиться о них.

Другой способ проверить или восстановить таблицы заключается в вызове программы myisamchk, которая также осуществляет доступ к файлам. Однако в этом случае операции с таблицами выполняются не самим сервером, а внешней по отношению к нему программой. Это порождает проблему координации доступа к таблицам. Необходимо предотвратить все попытки изменения таблицы на время работы с ней программы myisamchk. Если об этом вовремя не позаботиться, то вполне возможно, что параллельные усилия по доступу к таблице смогут повредить ее и сделать непригодной для дальнейшего использования. Очевидно, одновременная запись сервером и программой myisamchk в одну и ту же таблицу приведет к непоправимым последствиям, но даже запись одной из них в то время, когда другая производит чтение, может привести к обескураживающим результатам. Программа, осуществляющая чтение в то время, когда другая программа вносит изменения в таблицу, может быть приведена в неопределенное состояние.

Аналогичная проблема возникает и в других ситуациях. Например, некоторые методы резервирования предполагают копирование табличных файлов. Необходимо запретить серверу вносить изменения в таблицы во время выполнения процедуры резервного копирования. Это гарантирует целостность файлов резервной копии. Некоторые методы заключаются в полной замене поврежденных таблиц неповрежденными копиями файлов. В таком случае необходимо запретить серверу доступ к таблице вообще.

Один из способов полностью исключить взаимодействие с сервером — выключить его. Не работая, он не может осуществить доступ к таблицам, с которыми работает другая программа. Но нежелание администраторов так делать вполне объяснимо — главным результатом такого подхода является полная недоступность сервера. Процедура, которая описана в этом разделе, позволит избежать взаимодействия работающего сервера и операций, внешних по отношению к нему.

Для координации работы с сервером воспользуйтесь протоколом блокировки. Сервер имеет два типа блокировки.

- Сервер использует внутренний механизм блокировки для того, чтобы запросы от разных клиентов не смешивались, — например, чтобы запросы SELECT, поступившие от одного клиента, не перемешивались с запросами UPDATE другого.
- Сервер также может использовать внешнюю блокировку для предотвращения модификации табличных файлов другими программами во время их использования. Такая возможность базируется на блокировке на уровне файловой системы вашей операционной системы.

  Обычно причина использования сервером внешних блокировок кроется во взаимодействии с программами типа myisamchk при проведении текущих проверок таблиц. Однако внешняя блокировка на некоторых системах не работает. В этом случае нельзя полагаться на внешнюю блокировку и вместо нее следует использовать протокол внутренней блокировки. Кроме того,

внешняя блокировка пригодна только для операций, которые требуют доступа типа "только чтение" к таблицам. Ею нельзя пользоваться при необходимости доступа "чтение/запись". (Например, если необходимо не только проверить, но и восстановить таблицу, необходимо использовать внутренний механизм блокировки.)

Изложенный в этом разделе материал имеет отношение только к работе с типами таблиц, в которых каждая таблица представлена своим собственным файлом (это типы MyISAM, BDB и ISAM). И не имеет никакого отношения к типу InnoDB, при котором все таблицы располагаются в едином табличном пространстве InnoDB.

Координирование взаимодействия с применением внутренней блокировки

Чтобы предотвратить доступ сервера к таблице, когда вы работаете с ней, описанные здесь протоколы блокировки используют внутренний механизм блокировки. Основная идея заключается в том, что вы подключаетесь к серверу с помощью `mysql` и задаете оператор `LOCK TABLE` таблице, с которой планируете начать работу. Затем `mysql` работает вхолостую (т.е. он остается активизированным, но при этом ничего не делает с таблицей, за исключением того, что держит ее заблокированной), а над табличными файлами производятся все запланированные действия. После выполнения этих операций вы возвращаетесь к сеансу работы с `mysql` и снимаете блокировку, чтобы сообщить серверу, что он может продолжать работу с таблицей.

Тип используемого протокола блокировки зависит от того, требуется ли к табличным файлам только доступ с правами на чтение или с правами на чтение/запись. Для операций проверки или копирования файлов вполне достаточно доступа с правами на чтение. Для операций, при которых производится обновление файлов, таких как восстановление или замена поврежденных файлов, потребуется доступ с правами на чтение/запись.

В соответствии с протоколами блокировки для наложения и снятия блокировок используются операторы `LOCK TABLE` и `UNLOCK TABLE`. Кроме того, используется оператор `FLUSH TABLE`, чтобы сервер немедленно внес все отложенные изменения на диске, и как средство информирования сервера о том, что он должен повторно открыть таблицу, когда войдет в нее следующий раз. При этом может использоваться форма оператора `FLUSH TABLE` с аргументом, задающим имя таблицы, которая обновляет только заданную таблицу. Если версия вашего сервера MySQL старше 3.23.23, необходимо использовать форму `FLUSH TABLES`, которая не принимает имени таблицы и сбрасывает весь табличный кэш.

Операторы `LOCK`, `FLUSH` и `UNLOCK` необходимо выполнять на протяжении одного сеанса `mysql`. Если сначала таблицу заблокировать, а затем выйти из `mysql`, блокировка будет снята. С этого момента сервер посчитает в праве возобновить работу с таблицей. А это приводит к тому, что работать с таблицами становится небезопасно.

Проще всего выполнить процедуры блокировки в двух отдельных окнах: в одном — для работы `mysql`, а в другом — для выполнения работ с табличными фай-

лами. Это позволит вам оставить работающим mysql, пока вы будете работать с таблицами. Если же вы не используете в работе многооконную среду, вам потребуется остановить и возобновить выполнение mysql, пользуясь возможностями управления заданиями, которые имеет используемая вами оболочка.

Блокировка таблицы для доступа с правами "только чтение"

Данный протокол предназначен для выполнения операций, при которых вам требуется только читать табличные файлы, например копировать файлы или проверять их целостность. Его возможностей достаточно для блокировки в таком случае чтения; сервер предотвратит обновление таблиц другими клиентами и позволит им только чтение их. Этот протокол не может использоваться для изменения таблицы.

1. Вызовите mysql в окне A и для включения блокировки и обновления таблицы на диск воспользуйтесь следующими командами:

```
% mysql db_name
mysql> LOCK TABLE tbl_name READ;
mysql> FLUSH TABLE tbl_name;
```

   Эта блокировка блокирует операции записи в таблицу другими клиентами во время ее проверки. Оператор FLUSH приводит к закрытию табличных файлов с обновлением всех несохраненных еще изменений, которые могли находиться в кэше, в таблицы на диск.

2. При работе mysql вхолостую, перейдите в окно B, чтобы иметь возможность работать с табличными файлами. Например, таблицу MyISAM можно проверить следующим образом:

```
% myisamchk tbl_name
```

3. После того как вы завершите работу с таблицей, вернитесь в сеанс mysql в окно A и снимите с таблицы блокировку:

```
mysql> UNLOCK TABLE;
```

Вполне возможно, что ваша работа с таблицей выявит необходимость дальнейших действий. Например, при проверке таблицы с помощью программы myisamchk могут обнаружиться проблемы, которые потребуют вмешательства. Процедура исправления требует доступа с правами "чтение/запись", который можно получить, воспользовавшись другим протоколом, описанным ниже.

Блокировка таблицы для доступа с правами "чтение/запись"

Этот протокол предназначен для выполнения операций, при которых вам потребуется вносить такие изменения в табличные файлы, как восстановление табличных файлов. Для этого необходимо наложить блокировку записи, чтобы полностью предотвратить доступ сервера к таблице, пока вы работаете с ним.

За исключением двух различий процедура блокировки таблицы при восстановлении таблицы аналогична процедуре блокировки при проверке. Во-первых, необходимо применить блокировку записи, а не чтения. Поскольку в таблицу будут вноситься изменения, сервер не должен иметь доступа вообще. Во-вторых, после работы с таблицей необходимо запустить еще один оператор FLUSH

TABLE. Некоторые операции, такие как восстановление таблицы программой myisamchk, создают новый индексный файл, а сервер может не заметить это, если не обновить кэш таблицы снова. Для того чтобы заблокировать полный доступ, выполните следующие действия.

1. Вызовите mysql в окне A и для включения блокировки и обновления таблицы воспользуйтесь следующими командами:

```
% mysql db_name
mysql> LOCK TABLE tbl_name WRITE;
mysql> FLUSH TABLE tbl_name;
```

2. При работе mysql вхолостую перейдите в окно B, чтобы иметь возможность работать с табличными файлами. Например, можно проверить таблицу MyISAM:

```
% myisamchk --recover tbl_name
```

   Этот пример приведен исключительно для иллюстрации. Задаваемые вами команды зависят только от выполняемых задач. (Заметим, что на случай возникновения какой-либо ошибки следует предусмотреть операцию копирования табличных файлов.)

3. После того как вы завершите работу с таблицей, вернитесь в сеанс mysql в окне A и снимите с таблицы блокировку:

```
mysql> FLUSH TABLE tbl_name;
mysql> UNLOCK TABLE;
```

Блокировка всех баз данных с правом чтения

Удобным методом запрета изменений клиентами во все таблицы сразу является одновременная блокировка чтения всех таблиц во всех базах данных. Для этого необходимо задать следующий оператор:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

Для того чтобы снять блокировку, необходимо задать команду

```
mysql> UNLOCK TABLES;
```

Пока таблицы заблокированы таким образом, клиенты могут производить чтение из них, но не могут делать запись. Это очень хорошо с точки зрения обеспечения условий для операций копирования во всех каталогах баз данных. С другой стороны, это будет не совсем дружественным шагом по отношению к клиентам, которым необходимо вносить изменения в базу данных, поэтому держать блокировку сервера нужно не дольше, чем это необходимо.

Предотвращение взаимодействия с помощью внешней блокировки

В некоторых случаях для координирования работы с сервером можно использовать внешнюю блокировку. Это можно сделать, работая с табличными файлами. Например, если ваша система поддерживает механизм внешней блокировки, программы myisamchk и isamchk знают, как ее можно использовать для координации работы с сервером. Однако это может быть исполь-

зовано только для действий, требующих только доступа с разрешением чтения, таких как проверка таблиц. Для таких операций, как восстановление таблиц, для которых требуется доступ типа "чтение/запись", на внешнюю блокировку полагаться нельзя.

Внешняя блокировка основывается на блокировке файлов, но операции восстановления в исполнении программ `myisamchk` и `isamchk` копируют табличные файлы в новые файлы, которые впоследствии используются для замещения исходных файлов. Сервер ничего не "знает" о новых файлах, что делает совершенно бессмысленными все попытки управления доступом посредством блокировок файлов.

Начиная с MySQL 4 внешние блокировки по умолчанию во всех системах являются отключенными. Их можно включить, если есть уверенность, что они функционируют в вашей системе правильно. Но лучше все-таки вместо них воспользоваться механизмом внутренней блокировки.

Для того чтобы определиться с возможностью использования сервером внешней блокировки, проверьте соответствующую переменную сервера. Начиная с MySQL 4 эта переменная называется `skip_external_locking`, до этого — `skip_locking`, но можно проверить, какую из них использует ваш сервер, выполнив следующий запрос:

```
mysql> SHOW VARIABLES LIKE 'skip%locking';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| skip_external_locking | ON    |
+-----------------------+-------+
```

В зависимости от значения переменной `skip_external_locking` (или `skip_locking`) выполните следующее.

- Если значение переменной `skip_external_locking` равно `ON`, то внешняя блокировка отключена и сервер не имеет информации о том, когда программа `myisamchk` или `isamchk` осуществляет доступ к таблице. Если необходимо оставить сервер в работоспособном состоянии при непосредственной работе с табличными файлами, можно прибегнуть к использованию внутренней блокировки, чтобы сервер не работал с указанной таблицей. При блокировках типа "только чтение" или "чтение/запись" необходимо использовать соответствующий протокол.
- Если значение переменной `skip_external_locking` равно `OFF`, то внешняя блокировка включена и программу `myisamchk` или `isamchk` можно использовать для таких операций чтения, как проверка таблиц. Сервер и программа будут взаимодействовать при осуществлении доступа к таблице. Однако, перед тем, как запустить одну из программ, необходимо обновить табличный кэш с помощью команды `mysqladmin flush-tables` и убедиться в том, что никто не сможет обновлять таблицу до тех пор, пока вы не завершите работу с ней. Для восстановления таблиц может потребоваться обновить их, поэтому внешней блокировкой при этом пользоваться нельзя. Для этого необходимо прибегнуть к помощи внутреннего протокола блокировки доступа "чтение/запись".

Устранение неполадок

Ниже приведены общие стратегии, которые администраторы MySQL могут применять для того, чтобы сохранять базы данных неповрежденными.

- Постараться извлечь максимальную выгоду из возможностей автоматического восстановления, которыми располагает сервер.
- Разработать политику регулярного создания резервных копий баз данных, которая всегда позволит вам опереться на что-то, если базы данных будут безвозвратно утеряны.

Кроме того, необходимо планировать регулярное предупредительное обслуживание. Эти методы обсуждаются ниже, в разделе "Резервирование и копирование баз данных", так как они используют программы обслуживания, описанные в этом разделе.

Использование возможностей автовосстановления сервера

Передней линией вашей обороны, оберегающей целостность базы данных MySQL, является использование возможности автовосстановления сервером во время запуска. При запуске сервер выполняет определенные типы проверок таблиц. Это помогает решить проблемы, возникшие во время предыдущего краха сервера или компьютера, на котором работает сервер.

В СУБД MySQL предусмотрена возможность восстанавливаться после различных ошибок, причем для этого ничего, кроме запуска сервера, делать не надо. В большинстве случаев он сам внесет все необходимые коррективы. Вот перечень вероятных действий сервера.

- Если дескриптор таблицы InnoDB активизирован, он может выполнять проверку на наличие различных проблем автоматически. Выполненные транзакции, присутствующие в повторениях журнала, но еще не перенесенные в таблицы, прогоняются повторно (переделываются).
- Невыполненные транзакции, которые выполнялись во время сбоя, отменяются в соответствии с информацией, содержащейся в журналах отмены. В результате можно с полной уверенностью гарантировать, что ваши таблицы InnoDB будут находиться в целостном состоянии, отображая результаты всех транзакций, которые выполнялись до краха.
- При активизированном дескрипторе BDB-таблиц сервер тоже пытается произвести автовосстановление, опираясь на содержимое своих регистрационных файлов.
- Дескриптор MyISAM может осуществлять автоматические операции проверки и восстановления.

Для управления процессом восстановления предусмотрен параметр сервера `--myisam-recover=level`, где значение *level* может быть пустым для отключения возможности проверки вообще или списком, разделенным запятыми и состоящим из одного или более значений: `DEFAULT` (равносильно полному отсутст-

вию параметра), BACKUP (создать резервную копию таблицы, если она была изменена), FORCE (запустить восстановление, даже если больше одной строки данных будет утеряно) или QUICK (быстрое восстановление). Параметр `--myisam-recover` впервые был добавлен в MySQL 3.23.25.

Если автовосстановление таблиц типа InnoDB или BDB оказывается неудачным из-за невосстанавливаемых ошибок, сервер прекращает работу, предварительно записав соответствующее сообщение в журнал ошибок. Для того чтобы сервер запустился, невзирая на это, необходимо запустить процедуру автовосстановления вручную. О том, как это сделать, рассказывается ниже, в разделе "Восстановление табличного пространства InnoDB или таблиц BDB".

Автоматическая проверка во время запуска для таблиц ISAM не предусмотрена и вряд ли когда-либо будет предусмотрена, так как поддержка таблиц ISAM в MySQL уже приостановлена в пользу таблиц MyISAM. И настоятельно рекомендуется рассмотреть возможность преобразования таблицы ISAM в таблицы типа MyISAM. Для этого достаточно воспользоваться известным оператором ALTER TABLE:

```
ALTER TABLE tbl_name TYPE = MYISAM;
```

С помощью программы `mysql_convert_table_format` можно преобразовать все таблицы в базе данных с помощью одной команды. Этот сценарий написан на языке Perl, и для его работы необходимо, чтобы был установлен интерфейс DBI. Просмотреть, как он работает, можно с помощью параметра `--help`.

Если вы не хотите преобразовывать таблицы ISAM, можно сделать так, чтобы перед запуском сервера они проверялись с помощью программы `isamchk`. Кроме того, если версия вашего сервера ниже 3.23.25 (т.е. до добавления параметра `--myisam-recover`), можно проверить таблицы MyISAM с помощью программы `myisamchk` до того, как сервер будет запущен. Раздел "Планирование превентивного обслуживания" посвящен обсуждению организации проверки таблиц с помощью этих программ перед запуском сервера.

Резервирование и копирование баз данных

На случай, если таблицы будут утеряны или повреждены, очень важно резервировать базы данных. В случае серьезного краха системы для восстановления состояния таблиц базы данных при наименьших возможных потерях потребуется резервная копия.

Еще одна угроза потери данных исходит от самих пользователей, которые могут ошибочно использовать операторы DROP DATABASE или DROP TABLE, а потом требуют от администратора немедленного восстановления данных. Иногда сам администратор MySQL может непреднамеренно нанести ущерб, например при попытке откорректировать табличный файл, используя текстовый редактор (`vi` или `emacs`). Это непременно нанесет вред таблице!

Методы, которые используются для создания резервных копий, могут также помочь в решении задачи копирования баз данных на другой сервер. Чаще всего база данных переносится на сервер, работающий на другом узле, но можно переносить данные и на сервер, который работает на том же компьютере. Это можно сделать при тестировании новой версии сервера MySQL на реальных данных, с которыми работает основной сервер.

Другое применение резервных копий заключается в установке сервера репликации, поскольку одним из первых шагов при создании подчиненного сервера репликации является получение моментального снимка сервера в определенный момент времени. Таким моментальным снимком является резервная копия, и ее загрузка синхронизирует данные подчиненного и главного сервера.

После этого обновления, сделанные на главном сервере, реплицируются на подчиненный сервер с использованием стандартного механизма репликации. Сам механизм репликации описан в главе 11, "Общее администрирование MySQL".

Основные методы резервирования баз данных заключаются в использовании программы `mysqldump` или прямого копирования файлов базы данных (например, с помощью `mysqlhotcopy`, `cp`, `tar` или `cpio`). Каждый метод имеет свои преимущества и недостатки.

- Программа `mysqldump` работает совместно с сервером MySQL. Методы прямого копирования предусматривают использование операций копирования файлов, которые будут по отношению к серверу внешними. При этом необходимо предпринять шаги, которые бы гарантировали, что сервер не будет модифицировать таблицы во время копирования. Эта проблема аналогична той, которая возникает при использовании резервных копий баз данных, полученных с помощью файловой системы: если таблица базы данных обновляется во время резервного копирования с помощью механизмов файловой системы, невозможно будет гарантировать целостность резервируемых таблиц, а следовательно, такие резервные копии мало чем смогут помочь при восстановлении таблиц.
- Программа `mysqldump` работает медленнее, чем методы непосредственного копирования, так как операция дампа включает передачу информации по сети. Методы прямого резервного копирования работают на уровне файловой системы и не требуют сетевого трафика.
- Программа `mysqldump` генерирует текстовые файлы, переносимые на другие машины, включая и работающие на различных аппаратных архитектурах. Поэтому такие файлы используются для копирования баз данных. Файлы, сгенерированные методами непосредственного копирования, не всегда могут быть перенесены на другие машины. Это зависит от использования таблицами машинно-независимых форматов хранения данных.

  Таблицы типа ISAM не удовлетворяют этим ограничениям. Например, копирование файлов с ОС Solaris, работающей на архитектуре SPARC, на ОС Solaris, также работающей на данной архитектуре, будет выполняться, а копирование с ОС Solaris на архитектуре SPARC на ОС Solaris, работающей на архитектуре Intel или Mac OS X, не удастся. Таблицы формата MyISAM и InnoDB обычно машинно-независимы.

  Для этих типов таблиц непосредственно скопированные файлы могут быть перенесены на сервер, работающий на другой аппаратной платформе. Более подробное обсуждение характеристик различных типов таблиц, включая те, которые считаются переносимыми, можно найти в приложении B, "Операторы и функции".

Независимо от того, какой метод резервного копирования вы выберете, существуют определенные принципы, следуя которым при необходимости восстановить содержимое базы данных вы добьетесь лучших результатов.

- Регулярно выполняйте резервное копирование. Для этого разработайте график и строго следуйте ему.
- Задайте двоичное протоколирование обновлений на сервере (см. раздел "Ведение файлов журналов" в главе 11). Двоичные журналы могут помочь при восстановлении баз данных после краха. После восстановления баз данных с помощью файлов с резервными копиями до состояния, в котором они были в момент резервного копирования, посредством запросов из журналов обновления следует восстановить обновления таблиц, произведенные с момента резервного копирования. Таким образом, будет восстановлено состояние таблиц, в котором они находились в момент краха.
- Для файлов резервного копирования используйте простую и ясную схему присвоения имен. Такие имена, как `backup1`, `backup2` и т.д., не несут смысловой нагрузки. Когда настанет черед операций восстановления, будет потеряно слишком много времени на определение содержания этих файлов. Практика показала, что лучше всего использовать имена резервных копий, которые содержат имена баз данных и их даты, например:

```
% mysqldump sampdb > /archive/mysql/sampdb.2002-10-02
% mysqldump menagerie > /archive/mysql/menagerie.2002-10-02
```

- Во избежание переполнения диска периодически удаляйте устаревшие резервные копии. Один из способов это сделать — воспользоваться методом ротации файлов, описанным в главе 11. Аналогичными принципами можно руководствоваться при удалении устаревших файлов.
- Выполняйте резервное копирование с применением возможностей резервного копирования файловой системы. В случае полного краха, в результате которого пропадет не только каталог данных, но и весь диск со всеми резервными копиями ваших файлов, у вас будет под рукой такая копия. Кроме того, необходимо резервировать и сами журналы.
- Не размещайте файлы с резервными копиями в файловой системе, которая используется для хранения баз данных. Это позволит снизить вероятность переполнения файловой системы, вызванного созданием резервных копий. Кроме того, если эта файловая система располагается на другом устройстве, вы сможете еще больше уменьшить масштабы ущерба, которые может повлечь за собой сбой устройства, поскольку в таком случае сбой в работе одного из устройств не сможет уничтожить каталог данных и все резервные копии одновременно.

Применение программы `mysqldump` для резервирования и копирования баз данных

При использовании программы `mysqldump` для создания файлов резервных копий базы данных в формате SQL по умолчанию записывается файл, состоящий из операторов `CREATE TABLE`, создающих резервируемые таблицы, и операторов

INSERT, содержащих данные для столбцов таблицы. Для воссоздания таблицы можно взять файл вывода программы mysqldump и использовать его в качестве ввода программы mysql для перезагрузки в MySQL. (Для чтения вывода программы mysqldump в формате SQL программу mysqlimport использовать нельзя!)

Базу данных можно перенести в текстовый файл с помощью команды

```
% mysqldump sampdb > /archive/mysql/sampdb.2002-10-02
```

Начало выведенного файла будет выглядеть следующим образом:

```
-- MySQL dump 9.06
--
-- Host: localhost Database: sampdb
---------------------------------------------------------
-- Server version 4.0.3-beta-log
--
-- Table structure for table 'absence'
--
  CREATE TABLE absence (
  student_id int(10) unsigned NOT NULL default '0',
  date date NOT NULL default '0000-00-00',
  PRIMARY KEY (student_id,date)
) TYPE=MyISAM;
--
-- Dumping data for table 'absence'
--
INSERT INTO absence VALUES (3,'2002-09-03');
INSERT INTO absence VALUES (5,'2002-09-03');
INSERT INTO absence VALUES (10,'2002-09-06');
...
```

Остальная часть файла состоит из операторов CREATE TABLE и INSERT.

Файлы резервных копий могут достигать большого размера, поэтому, возможно, потребуется уменьшить их. Одним из способов сделать это является использование параметра --opt, который оптимизирует процесс создания файлов меньшего размера:

```
% mysqldump --opt sampdb > /archive/mysql/sampdb.2002-10-02
```

Кроме того, файл дампа можно сжать. Например, для того, чтобы сжать файл резервной копии, можно использовать команду, аналогичную следующей:

```
% mysqldump --opt sampdb | gzip > /archive/mysql/sampdb.2002-10-02.gz
```

Если вы считаете, что с большими файлами дампов трудно работать, то можно заархивировать содержимое отдельных таблиц, задав имя базы данных в командной строке программы mysqldump с последующим за ним списком таблиц, а не всех таблиц базы данных. Это поможет разбить дамп на небольшие и более управляемые файлы. В следующем примере показано, как сохранить отдельные таблицы в отдельных файлах:

```
% mysqldump --opt sampdb student score event absence > gradebook.sql
% mysqldump --opt sampdb member president > hist-league.sql
```

Параметр --opt используется при создании резервных файлов, которые будут периодически применяться для обновления содержимого другой базы данных. Это происходит потому, что он автоматически активизирует параметр --add-drop-table, который задает программе mysqldump в файле ставить перед каж-

дым оператором CREATE TABLE оператор DROP TABLE IF EXISTS для той же самой таблицы. Это означает, что впоследствии, при загрузке файла резервной копии во вторую базу данных, в случае существования таблицы ошибки не произойдет. Данный метод можно использовать для периодической перезагрузки копий данных из баз данных вашего рабочего сервера при наличии еще одного тестового сервера, который не является подчиненным сервером репликации.

Если вам необходимо перенести базу данных на другой сервер, вам может даже не потребоваться создавать файлы с резервными копиями. Убедитесь в том, что база данных существует на другом узле, а затем передайте базу данных по сети. Это позволит непосредственно считывать вывод из mysqldump. Например, для того, чтобы скопировать базу данных sampdb с локального узла на сервер boa.snake.net, необходимо поступить следующим образом:

```
% mysqladmin -h boa.snake.net create sampdb
% mysqldump --opt sampdb | mysql -h boa.snake.net sampdb
```

Если у вас нет учетной записи MySQL на локальном узле, который позволит получить доступ к серверу boa.snake.net, но вы имеете такую учетную запись на самом узле boa.snake.net, для удаленного запуска команд MySQL можно воспользоваться оболочкой ssh:

```
% ssh boa.snake.net mysqladmin create sampdb
% mysqldump --opt sampdb | ssh boa.snake.net mysql sampdb
```

Впоследствии, если вам потребуется обновить базу данных sampdb на узле boa.snake.net, повторите команду mysqldump.

Ниже приведены другие параметры программы mysqldump, которые могут пригодиться вам в работе.

- Комбинация параметров --flush-logs и --lock-tables может помочь в расстановке контрольных точек в базе данных. Параметр --lock-tables блокирует все резервируемые таблицы, а параметр --flush-logs закрывает и открывает вновь журналы регистрации. При создании последовательности журналов обновления или двоичных журналов обновления новый журнал будет содержать только те запросы, которые модифицируют базу данных после контрольной точки. Это синхронизирует журнал регистрации по времени резервного копирования. (Однако блокирование всех таблиц не совсем благоприятно влияет на доступ клиентов, выполняющих во время резервного копирования операции обновления.)

  При использовании параметра --flush-logs для установки контрольных точек на журналах во время резервного копирования лучше всего копировать сразу всю базу данных. Во время проведения операций восстановления общепринято брать содержимое журналов для всей базы данных целиком. При резервировании отдельных таблиц контрольные точки журналов синхронизировать значительно труднее. (Параметра, предназначенного для управления обновлениями отдельных таблиц, не существует, поэтому вам придется делать это самому.)

- По умолчанию программа mysqldump, перед тем как записать все содержимое таблицы в дамп, считывает его в память. Но это не обязательно и в действительности при создании больших таблиц является причиной появления оши-

бок. Для того чтобы записать данные в файл дампа, необходимо задать параметр --quick. Для того чтобы еще больше оптимизировать процесс получения дампа, вместо параметра --quick используйте параметр --opt. Параметр --opt включает все другие параметры, которые позволяют ускорить процесс получения дампа данных. Кроме того, при этом файл дампа записывается таким образом, который позволяет восстанавливать данные значительно быстрее.

Выполнение резервирования с использованием параметра -opt, вероятно, является наиболее общепринятым методом из-за преимущества в скорости резервного копирования. Однако следует помнить, что параметр --opt оптимизирует процедуру резервного копирования, а не доступ клиентов к базе данных. Параметр --opt препятствует доступу пользователей к таблицам базы данных, так как при этом блокируются все таблицы базы данных. В этом можно очень легко убедиться. Попробуйте выполнить резервное копирование при максимальных нагрузках на базу данных, и очень скоро вам начнут звонить по телефону, чтобы выяснить, что случилось.

- Параметр --delayed является полной противоположностью параметра --opt. Он приводит к созданию программой mysqldump вместо операторов INSERT операторов INSERT DELAYED. Параметр --delayed позволяет решить задачу загрузки файла дампа в другую базу данных, если необходимо минимизировать влияние этой операции на другие запросы, которые могут одновременно выполняться в этой базе данных.
- Обычно имя базы данных указывается в командной строке mysqldump с указанием имен таблиц. Для одновременного копирования сразу нескольких баз данных можно воспользоваться параметром --databases. В этом случае mysqldump будет интерпретировать все имена, указанные в командной строке, как имена баз данных и копировать все таблицы из каждой из них. Для того чтобы скопировать сразу все базы данных сервера, воспользуйтесь параметром --all-databases. В этом случае аргументы с именами баз данных или таблиц не потребуются. Но при использовании параметра --all-databases при копировании дампа на другой сервер следует быть предельно осторожными: этот дамп содержит таблицы разрешения из базы данных mysql. А замещение таблиц разрешения другого сервера в наши планы не входит.
- Параметр --compress поможет при копировании базы данных на другую машину, поскольку он уменьшает число байтов, передаваемых по сети:

```
% mysqldump --opt sampdb | mysql --compress -h boa.snake.net sampdb
```

  Параметр --compress предназначен для обмена данными с сервером, размещенным на удаленном узле. Сжатие применяется только к сетевому трафику и не влияет на таблицы в принимающей базе данных.
- По умолчанию программа mysqldump копирует структуру таблицы (операторы CREATE TABLE) и содержимое таблицы (операторы INSERT). Для того чтобы скопировать что-то одно, можно задать параметр --no-create-info или --no-data.

Программы mysqldump имеют также множество других параметров. Об этом подробнее можно узнать в приложении Д, "Программы MySQL".

Использование методов непосредственного резервирования и копирования

Другим способом резервирования баз данных или таблиц, не предусматривающим использование программы mysqldump, является непосредственное копирование табличных файлов с помощью средств, которыми располагает файловая система. Обычно это делается с использованием таких утилит, как cp, tar или cpio. При использовании метода прямого копирования вы должны убедиться в том, что другие таблицы в момент копирования не используются. Если сервер обновляет таблицу во время ее копирования, копирование как таковое теряет смысл. Лучшим способом обеспечить целостность копий является отключение сервера, копирование файлов и перезапуск сервера.

Если невозможно отключить сервер, можно воспользоваться протоколом блокировки доступа чтения, описанным выше, в разделе "Координация работы с сервером". Это обеспечит блокировку обновления таблиц базы данных сервером во время ее копирования.

Предположим, что сервер выключен или таблицы, предназначенные для копирования, заблокированы. Вот пример, который наглядно продемонстрирует копирование всех таблиц базы данных sampdb в резервный каталог. Для каталога данных /usr/local/mysql/data команды будут иметь такой вид:

```
% cd /usr/local/mysql/data
% cp -r sampdb /archive/mysql
```

Отдельные таблицы можно зарезервировать следующим образом:

```
% cd /usr/local/mysql/data/sampdb
% cp member.* /archive/mysql/sampdb
% cp score.* /archive/mysql/sampdb
...
```

После завершения резервного копирования сервер можно запустить вно В том случае, если сервер все же работал, а были заблокированы только табли блокировку можно снять.

Методы прямого копирования можно применить и при копировании б данных с одной машины на другую. Например, команду cp можно заменить мандой scp. Если на узле boa.snake.net каталогом данных является ката /var/mysql/data, следующая команда позволит скопировать на этот узел к лог базы данных sampdb:

```
% cd /usr/local/mysql/data
% scp -r sampdb boa.snake.net:/var/mysql/data
```

Заметим, что во время копирования базы данных на другие узлы вступак силу дополнительные ограничения.

- Машины должны иметь подобную архитектуру, или копируемые табли должны быть представлены переносимым типом таблиц. В противном чае полученные на другом узле таблицы могут иметь довольно странное держимое.
- Необходимо воспрепятствовать попыткам серверов на обоих узлах внос изменения в файлы во время их копирования. Безопаснее всего на вр работы с таблицами остановить оба сервера.

- Методы прямого копирования, подобные только что описанным, лучше всего работают с таблицами типа MyISAM и ISAM, которые представляют таблицы уникальными наборами файлов в каталоге базы данных. Для таких типов, как InnoDB, необходимо предпринять дополнительные предосторожности. Подробнее об этом — ниже, в разделе "Восстановление табличного пространства InnoDB или таблиц BDB".

Резервирование с помощью программы mysqlhotcopy

Начиная с версии 3.23.11 дистрибутивы MySQL включают сценарий `mysqlhotcopy`, являющийся сценарием Perl DBI, который предназначен для резервного копирования базы данных. Слово "hot", присутствующее в имени этой программы, имеет непосредственное отношение к тому факту, что резервные копии делаются во время работы сервера и не рассчитаны на отключение сервера. Ниже описаны преимущества использования программы `mysqlhotcopy`:

- Эта программа работает быстрее, чем `mysqldump`, так как копирует файлы непосредственно, а не с помощью запросов через сервер, как это делает программа `mysqldump`. (Это значит, что ее необходимо запускать непосредственно на узле, т.е. она не работает с удаленными серверами.)
- Данная программа автоматически управляет протоколом блокирования, необходимым для предотвращения обновления таблиц сервером во время их копирования. Программа `mysqlhotcopy` делает это, используя механизм внутреннего блокирования (см. раздел "Координация работы с сервером").
- Эта программа обновляет журналы, синхронизирующие контрольные точки файлов резервных копий и журналов, и упрощает использование резервных копий при восстановлении (если такая необходимость возникнет в будущем).

Есть несколько способов активизировать `mysqlhotcopy`. Предположим, что требуется скопировать базу данных `sampdb`. Следующая команда создаст каталог `sampdb_copy` на сервере в каталоге данных и скопирует в него все файлы, принадлежащие базе данных `sampdb`:

```
% mysqlhotcopy sampdb
```

Для копирования базы данных в подкаталог `sampdb`, расположенный в указанном каталоге, укажите этот каталог после имени базы данных. Например, скопировать базу данных `sampdb` в каталог `/archive/2002-09-12/sampdb` можно с помощью следующей команды:

```
% mysqlhotcopy sampdb /archive/2002-09-12
```

Для того чтобы просмотреть, что программа `mysqlhotcopy` будет делать для любой заданной команды, при запуске программы задайте параметр `-n`. При этом она будет работать в режиме "имитации выполнения", в котором она только печатает команды, не выполняя их.

Создание резервной копии с помощью оператора BACKUP TABLE

Оператор `BACKUP TABLE`, появившийся в MySQL 3.23.25, обеспечивает резервное копирование сервером отдельных таблиц. Для этого он копирует таб-

личные файлы и работает только для таблиц MyISAM. Чтобы воспользоваться данным оператором, необходимо задать имена таблиц, которые будут копироваться, и каталог, в котором копируемые файлы будут сохранены. Например:

```
BACKUP TABLE tbl1, tbl2, tbl3 TO '/archive/sampdb';
```

Для выполнения этой операции заданный каталог уже должен существовать, вы должны иметь привилегию FILE и привилегию SELECT для таблиц. Оператор BACKUP TABLE закрывает таблицу, чтобы все отложенные обновления были выполнены, затем копирует все файлы описания и файлы данных (`.frm` и `.MYD`) из каталога базы данных. Он не копирует индексный файл `.MYI`, так как последний можно легко воссоздать на основании данных из двух других файлов.

Оператор BACKUP TABLE блокирует по одной таблице. Это значит, что при резервировании нескольких таблиц вполне вероятна ситуация, когда файлы резервной копии окажутся в нецелостном состоянии. Предположим, что мы резервируем таблицы `tbl1` и `tbl2`. Оператор BACKUP TABLE заблокирует таблицу `tbl1` только на время ее резервного копирования, поэтому она может быть обновлена во время резервного копирования `tbl2`. Это значит, что когда BACKUP TABLE завершает свою работу, содержимое таблицы `tbl1` уже будет отличаться от файлов копии. Для того чтобы гарантировать резервирование таблиц как единой группы данных, все резервируемые таблицы можно заблокировать на время работы оператора BACKUP TABLE. Эту блокировку можно снять оператором UNLOCK TABLES:

```
LOCK TABLES tbl1 READ, tbl2 READ;
BACKUP TABLE tbl1, tbl2 TO 'backup_dir_path';
UNLOCK TABLES;
```

Скопированную с помощью оператора BACKUP TABLE таблицу можно вновь загрузить на сервер с помощью оператора RESTORE TABLE (подробно об этом речь пойдет в разделе "Резервирование и копирование баз данных").

Резервное копирование табличного пространства InnoDB или таблиц BDB

Таблицы InnoDB и BDB могут архивироваться программой `mysqldump`, аналогично всем другим типам таблицы. Кроме того, можно воспользоваться методом непосредственного копирования, но при этом необходимо соблюдать следующие правила.

- В отличие от других типов, таблицы InnoDB не представляются отдельными файлами (за исключением того, что каждая имеет отдельный файл описания `.frm`). Они размещаются в одном табличном пространстве InnoDB, которое представляет собой один или несколько файлов. Для прямого копирования табличного пространства InnoDB необходимо скопировать все составляющие их файлы. Это необходимо делать после выключения сервера, чтобы быть в полной уверенности, что сервер завершил все отложенные транзакции. Необходимо также скопировать все файлы `.frm`, имеющие отношение к таблицам InnoDB, файлы регистрации InnoDB и конфигурационный файл, в котором задана конфигурация табличного пространства. (Конфигурационный файл необходимо скопировать, поскольку он может потребоваться для повторной инициализации табличного пространства на случай утери текущего конфигурационного файла.)

Для этих же целей используется утилита InnoDB Hot Backup, которую можно получить на Web-узле `innodb.com`. Это коммерческое средство, позволяющее резервировать таблицы InnoDB при работающем сервере.

- При прямом копировании таблиц BDB копируются все таблицы BDB, с которыми работает сервер, а также все регистрационные файлы для типа BDB. Это необходимо делать на выключенном сервере. Дескриптор BDB требует, чтобы при запуске сервера велись журналы, следовательно, для восстановления BDB-таблиц также потребуются журналы. Файлы BDB-журналов по умолчанию создаются в каталоге данных и им присваиваются имена вида `log.nnnnnnnnnn` с суффиксом, состоящим из десяти цифр.

Использование репликации для резервирования

Резервное копирование очень важно, но оно при исполнении администраторами MySQL своих обязанностей несет в себе конфликт интересов. С одной стороны, необходимо максимально увеличить доступность сервера для пользователей, что, прежде всего, подразумевает возможность обновления ими базы данных. С другой стороны, резервные копии могут принести при восстановлении максимальную пользу, при условии, что файлы резервных копий и контрольные точки в регистрационных журналах синхронизированы. Решить эти противоречащие задачи довольно трудно, так как лучшим способом синхронизировать резервные копии и контрольные точки журналов является обновление журналов во время резервного копирования с блокированием обновления посредством отключения сервера или одновременным копированием всех таблиц (например, с помощью параметра `--opt` программы `mysqldump`). К сожалению, блокировка возможности обновлений блокирует доступ клиентам к таблицам на протяжении выполнения всей процедуры резервного копирования.

Механизм репликации помогает устранить этот конфликт. Вместо того чтобы производить резервное копирование на главном сервере, это можно совершенно спокойно сделать на подчиненном. В таком случае отпадает необходимость отключать главный сервер или делать его на время выполнения резервного копирования каким-то образом недоступным для клиентов. Поэтому необходимо приостановить репликацию на подчиненный сервер с помощью команды `SLAVE STOP` и произвести резервное копирование на подчиненном сервере. (Если используется прием прямого копирования, необходимо также задать оператор `FLUSH TABLES`.) После этого с помощью оператора `SLAVE START` возобновите репликацию, и подчиненный сервер подхватит любые обновления, произведенные во время резервного копирования. В зависимости от используемого вами метода резервного копирования может даже не потребоваться останавливать репликацию. Например, при резервировании одной базы данных можно использовать программу `mysqlhotcopy` или `mysqldump` с соответствующими параметрами для одновременной блокировки всех таблиц. В таком случае сервер будет продолжать работать, не пытаясь при этом обновлять таблицы, заблокированные на время резервного копирования. По завершении резервного копирования блокировка снимается и подчиненный сервер автоматически продолжает обработку обновлений.

Использование резервных копий для переименования баз данных

Несмотря на то что в СУБД MySQL команда переименования баз данных не предусмотрена, СУБД MySQL позволяет производить и эту операцию. Сделайте с помощью программы `mysqldump` дамп базы данных, создайте пустую базу данных с новым именем, а затем загрузите файл дампа в новую базу данных. После этого старую базу данных можно удалить. Вот пример переименования базы данных из `db1` в `db2`:

```
% mysqldump db1 > db1.sql
% mysqladmin create db2
% mysql db2 < db1.sql
% mysqladmin drop db1
```

Проще всего переименовывать базу данных так: отключить сервер, переименовать каталог базы данных и снова запустить сервер. Однако такая стратегия приемлема только в случае, когда в базе данных отсутствуют таблицы BDB или InnoDB. Этот прием не работает для таблиц BDB, так как путь к каждой его таблице закодирован в файле `.db`. Этот прием не работает и в случае с таблицами InnoDB, поскольку имя базы данных для каждой таблицы хранится в табличном пространстве InnoDB, на которое переименование каталога базы данных никак не влияет.

Но какой бы способ переименования базы данных вы ни применили, всегда следует помнить, что права доступа к ней управляются из таблиц разрешения в базе данных `mysql`. Если в таблицах разрешения есть записи, которые имеют отношение непосредственно к переименованной базе данных, необходимо ввести в эти записи соответствующие изменения, учитывающие изменение имени. Для базы данных, переименованной из `db1` в `db2`, используются такие операторы:

```
mysql> UPDATE db SET Db = 'db2' WHERE Db = 'db1';
mysql> UPDATE tables_priv SET Db = 'db2' WHERE Db = 'db1';
mysql> UPDATE columns_priv SET Db = 'db2' WHERE Db = 'db1';
mysql> UPDATE host SET Db = 'db2' WHERE Db = 'db1';
```

Для таблицы пользователей таких операторов не требуется, так как в ней столбец `Db` отсутствует.

Восстановление таблиц и данных

Повреждение таблиц может произойти по самым различным причинам и различается масштабами своих последствий. Если вы достаточно удачливы, могут повредиться одна или две таблицы (например, если компьютер отключился из-за простого падения напряжения). В этом случае сам сервер сможет восстановить повреждение во время рестарта. Если удача отвернулась от вас, могут потребоваться более радикальные действия: замена всего каталога данных (например, если вышел из строя диск, вместе с ним и каталог данных). Восстановление также может потребоваться и во многих других ситуациях, например, в том случае, когда пользователи ошибочно удалили базу данных, таблицу или содержимое таблицы. Независимо от причины случившегося, необходимо провести определенные восстановительные работы.

- Если таблицы не утеряны, а только повреждены, можно попытаться восстановить их с помощью оператора `CHECK TABLE` либо программы `mysqlcheck` или

myisamchk. Вполне вероятно, что вам может не потребоваться сортировать их с помощью резервных файлов вообще, если характер повреждения позволяет восстановить таблицу с помощью операции восстановления.

- При полной утере таблиц их потребуется восстановить.

В начале этого раздела рассказывается о процедурах проверки и восстановления таблиц, которые могут пригодиться при минимальных повреждениях. В их число входят как интерактивные процедуры, которые могут использоваться по мере надобности, так и неинтерактивные, которые могут использоваться в процедурах запланированного превентивного обслуживания. В заключение этого раздела речь пойдет о процедурах восстановления таблиц и баз данных в случае их полной утери или при их повреждении в процессе восстановления.

Проверка и восстановление таблиц баз данных

Обычно процедура обнаружения и исправления ошибок состоит из следующих этапов.

1. Проверка таблицы на наличие ошибок. Если ошибки в таблице отсутствуют, процедура завершается. В любом другом случае возникает необходимость в их устранении.
2. Создание копий файлов таблиц перед началом восстановления на случай негативного сценария событий. Такой сценарий маловероятен, но если это произойдет, можно сделать новую копию таблицы из скопированных файлов и попробовать применить различные методы восстановления.
3. Попытка исправления таблицы. Если эта попытка оказывается удачной, на этом можно закончить, если нет — остается лишь восстановить базу данных из архива и журналов обновлений.

Выполнение последнего этапа описанной процедуры предполагает, что администратор аккуратно выполнил резервирование и активизировал двоичную регистрацию запросов на обновление таблиц. В любом другом случае его ждут большие неприятности. Настоятельно рекомендуется внимательно прочитать главу 11, "Общее администрирование MySQL", в которой подробно рассказывается об использовании программы mysqldump и регистрации обновлений. Мало кто из администраторов захочет столкнуться с проблемой безвозвратной потери таблиц из-за того, что ранее поленился активизировать регистрацию.

Проверить или восстановить таблицы можно с помощью программы myisamchk, которая работает непосредственно с табличными файлами. Или можно дать команду серверу проверить или восстановить таблицы с использованием операторов CHECK TABLE или REPAIR TABLE (или с помощью программы mysqlcheck, которая подключается к серверу и выполняет эти операторы для вас). Преимущество использования операторов SQL или программы mysqlcheck заключается в том, что сервер сам выполняет эту работу. При работе с программой myisamchk необходимо быть полностью уверенным в том, что сервер не работает с табличными файлами, пока с ними работаете вы.

Как уже говорилось в этой главе, если у вас имеется выбор между тем, чтобы работу выполнял сервер или внешняя программа, лучше остановиться на первом варианте.

В таком случае не надо беспокоиться о проблемах координирования доступа к таблице. Причинами, вследствие которых может потребоваться воспользоваться внешней программой myisamchk, являются следующие.

- Программой myisamchk можно пользоваться при неработающем сервере. Для работы операторов CREATE TABLE и REPAIR TABLE требуется работающий сервер.
- Программе myisamchk можно задать использование буферов больших размеров, чтобы проверка и восстановление выполнялись быстрее. Лучше, если у вас будут очень большие таблицы.
- Программа myisamchk может использоваться с более старыми версиями серверов. Операторы CHECK TABLE и REPAIR TABLE впервые появились в MySQL 3.23.13 и 3.23.14, тогда как myisamchk появилась в 3.23.0. (Конечно, это различие со временем сглаживается, так как серверы версии ниже 3.23.13 становятся редкостью.)

Примечание

Информация, приведенная здесь для myisamchk, может быть применима для использования с isamchk (см. в начале этой главы примечание "Взаимосвязь isamchk и myisamchk").

Проверка и восстановление таблиц с помощью myisamchk

Прежде чем приступить к проверке и восстановлению таблиц с помощью программы myisamchk, может потребоваться отключить сервер, чтобы гарантировать, что он не будет работать с табличными файлами в то время, когда с ними работаете вы.

Если необходимо оставить сервер работающим, обязательно прочтите раздел "Координация работы с сервером" настоящей главы. В этом разделе обсуждаются соответствующие протоколы блокировок, которые предотвращают использование таблиц одновременно с выполнением их проверки или процедур восстановления.

Вызов myisamchk

Программа myisamchk не может знать о том, где располагаются таблицы. Поэтому при запуске необходимо задать пути ко всем необходимым таблицам. Лучше всего это сделать непосредственно в каталогах, в которых находятся таблицы, поэтому, до того, как будет запущена программа myisamchk, перейдите в каталог, где размещена база данных. Затем укажите, какую таблицу следует проверить или восстановить, и параметры, указывающие, какой тип действия следует выполнить при этом:

```
% myisamchk options tbl_name ...
```

Аргумент *tbl_name* может быть именем таблицы или индексного файла таблицы. Это значит, что в имени файла можно использовать шаблон, основанный на расширении индексного файла для выбора всех необходимых файлов из текущего каталога:

```
% myisamchk options *.MYI
```

Можно и вовсе избежать работы с исходными табличными файлами. Их можно скопировать в другой каталог и там работать только с копиями.

Проверка таблиц программой myisamchk

Программа myisamchk располагает различными методами проверки таблиц, отличающимися степенью глубины проверки. Для того чтобы провести обычную проверку таблиц, воспользуйтесь одной из следующих команд:

```
% myisamchk tbl_name
% myisamchk --check tbl_name
```

По умолчанию при вызове myisamchk без параметров она работает как с параметром --check, поэтому эти две команды эквивалентны.

Обычного метода проверки, как правило, вполне достаточно для решения всех возникающих проблем. Но, если он не сообщает об ошибках, а администратор по-прежнему видит наличие повреждения (вероятно, из-за того, что запросы отрабатывают не совсем правильно), можно произвести расширенную проверку, задав параметр --extend-check. Такая проверка займет больше времени, но позволит убедиться в том, что каждая запись в табличном файле данных и каждый ключ в индексном файле будут проверены на соответствие записям. (Программа myisamchk также поддерживает параметр --medium-check для выполнения проверки промежуточной степени сложности, которая будет менее тщательной, но и более быстрой, чем расширенная проверка.)

Если при проверке --extend-check ошибки не обнаружены, можно гарантировать, что с таблицами все в порядке. Если с таблицей по-прежнему возникают проблемы, то причина должна заключаться во внешних источниках. Проверьте правильность написания всех запросов, которые могут вызывать проблемные результаты. Если вы уверены в том, что проблема кроется в самом сервере MySQL, заполните отчет об ошибках или подумайте о переходе на более новую версию.

Если программа myisamchk сообщает о том, что таблица содержит ошибки, можно попробовать восстановить ее.

Восстановление таблиц с помощью программы myisamchk

Восстановление таблиц — неблагодарная работа, поскольку зависит от специфики возникшей проблемы. Тем не менее разработаны общие принципы и процедуры, которые, если им следовать, позволяют существенно увеличить вероятность восстановления таблиц. Но начинать восстановление необходимо с самого быстрого способа, чтобы посмотреть, как он помогает в восстановлении. Если оказывается, что его недостаточно, необходимо перейти к более сложным (и более медленным) способам восстановления. (На практике большинство проблем решается без необходимости перехода на углубленные и медленные режимы восстановления.) Если таблицу восстановить невозможно, ее можно воссоздать из резервных копий. Инструкции по восстановлению из резервных копий также даны в этой главе.

Для того чтобы выполнить стандартную операцию восстановления таблицы, выполните следующие действия.

1. Попробуйте восстановить таблицу с помощью параметра --recover, при этом попытайтесь произвести восстановление с использованием параметра --quick только на основе содержимого индексного файла. Это позволит восстановить таблицу, не затрагивая файлы данных:

   ```
   % myisamchk --recover --quick tbl_name
   ```

2. Если проблемы остаются, запустите команду повторно без использования параметра `--quick`, что позволит программе `myisamchk` пойти дальше и модифицировать также и файл данных:

```
% myisamchk --recover tbl_name
```

3. Если не поможет и это, попробуйте воспользоваться режимом восстановления `--safe-recover`. Этот режим медленнее, чем обычный режим восстановления, но он поможет справиться с некоторыми проблемами, с которыми режим `--recover` справиться не в состоянии:

```
% myisamchk --safe-recover tbl_name
```

Это возможно при запуске программы, когда `myisamchk` остановится с таким сообщением об ошибке: "`Can't create new temp file: file_name`". Если такое произойдет, повторите команду, добавив параметр `--force` для удаления временного файла, который мог остаться после предыдущей неудачной попытки восстановления.

Если стандартной процедуре не удается восстановить таблицу, ваш индексный файл может быть поврежден при восстановлении. Вполне возможно, хотя и маловероятно, что отсутствует файл описания таблицы. В любом из этих случаев вам потребуется заменить поврежденные файлы, а потом повторить стандартную процедуру восстановления.

Чтобы восстановить индексный файл таблицы `t`, выполните следующие действия.

1. Перейдите в каталог базы данных с поврежденной таблицей.
2. Переместите файл данных таблицы `t.MYD` в безопасное место.
3. Вызовите `mysql` и воссоздайте новую пустую таблицу, выполнив следующие операторы, в которых используются файлы описания таблицы, `t.frm`, для восстановления файла данных и индексного файла:

   ```
   mysql> TRUNCATE TABLE t;
   ```

   До MySQL 4.0 используйте оператор DELETE:

   ```
   mysql> DELETE FROM t;
   ```

4. Завершите сеанс работы с `mysql` и верните исходные файлы данных в каталог базы данных, заменив новый пустой файл данных только что созданным файлом. Файл данных и индексный файл теперь рассинхронизированы, но индексный файл теперь имеет легальную внутреннюю структуру, которую может интерпретировать сервер, перестроенную на основании файла данных и файла описания таблицы.
5. Повторите процедуру восстановления таблицы.

Чтобы восстановить файл описания таблицы, `t.frm`, восстановите его из резервной копии, после чего попробуйте повторно выполнить стандартную процедуру восстановления. Если вследствие определенных причин резервная копия у вас отсутствует, но вам хорошо известен оператор `CREATE TABLE`, предназначенный для создания таблицы, ее еще можно восстановить.

1. Перейдите в каталог базы данных, содержащий разрушенную таблицу.
2. Переместите файл данных таблицы, `t.MYD`, в безопасное место. По необходимости переместите туда же индексный файл `t.MYI`.

3. Запустите программу `mysql` и для создания таблицы задайте оператор `CREATE TABLE`.
4. Выйдите из `mysql` и перенесите исходный файл данных в каталог базы данных, заменяя новые файлы данных, созданные вами только что. Если при выполнении шага 2 переносился и индексный файл, переместите и его.
5. Попробуйте выполнить стандартную процедуру восстановления еще раз.

Ускорение работы `myisamchk`

Работа `myisamchk` может занять много времени, особенно, если вы работаете с большой таблицей или используете один из более глубоких методов проверки или восстановления.

Можно ускорить этот процесс, задав программе `myisamchk` больше памяти. Программа `myisamchk` имеет несколько устанавливаемых параметров, но наиболее важными из них являются переменные, управляющие размерами используемых буферов.

Переменная	Значение
`key_buffer_size`	Размер буфера, используемого для хранения блоков индексов
`read_buffer_size`	Размер буфера, используемого для операций чтения
`sort_buffer_size`	Размер буфера, используемого для операций сортировки
`write_buffer_size`	Размер буфера, используемого для операций записи

Для того чтобы определить стандартные значения переменных программы `myisamchk`, запустите ее с параметром `--help`. Для изменения значения переменной можно воспользоваться в командной строке параметрами `--set-variable` *`variable=value`* или `-O` *`variable=value`*. Можно пользоваться сокращенными значениями переменных: `key`, `read`, `sort` и `write`. Например, при наличии свободной памяти программа `myisamchk` может использовать буфер сортировки 512 Мбайт и буфер чтения и записи 1 Мбайт. Для этого можно воспользоваться командой

```
% myisamchk -O sort=512M -O read=1M -O write=1M other-options tbl_name
```

Параметр `sort_buffer_size` используется только с параметром `--recover` (а не `--safe_recover`), и в таком случае `key_buffer_size` не используется.

Проверка и восстановление таблицы с помощью сервера

Операторы `CHECK TABLE` и `REPAIR TABLE` обеспечивают SQL-интерфейс для тестирования возможностей проверки и восстановления таблиц. Они работают для таблиц типа MyISAM. Начиная с MySQL 3.23.39 оператор `CHECK TABLE` также работает для таблиц InnoDB.

Оператор снабжается перечнем, состоящим из одного или нескольких имен таблиц, с последующими за ними параметрами, чтобы задать один или более режимов проверки и восстановления. Например, следующий оператор выполняет проверку среднего уровня на трех таблицах, но только в том случае, если они не были закрыты правильно:

```
CHECK TABLE tbl1, tbl2, tbl3 FAST MEDIUM;
```

Этот оператор делает попытку восстановить те же таблицы в режиме быстрого восстановления:

```
REPAIR TABLE tbl1, tbl2, tbl3 QUICK;
```

Оператор CHECK TABLE имеет следующие параметры, определяющие тип выполняемой поверки.

- CHANGED. Не проверять таблицы до тех пор, пока они не будут правильно закрыты или не изменялись с момента последней проверки.
- EXTENDED. Выполнить расширенную проверку. Это режим наиболее тщательной проверки из всех имеющихся и, соответственно, самый медленный. Он делает попытку проверки целостности данных и индексов.
- FAST. Не проверяет таблицы, за исключением случая, когда они были неправильно закрыты.
- MEDIUM. Произвести проверку среднего уровня. Этот режим задан по умолчанию.
- QUICK. Режим быстрой проверки, при котором сканируются только индексы. При этом строки данных не проверяются.

Не исключено, что оператор CHECK TABLE в некоторых случаях действительно модифицирует таблицу. Например, если таблица была маркирована как разрушенная или незакрытая правильно, но проверка не обнаруживает при этом никаких проблем. Оператор CHECK TABLE пометит таблицу как нормальную. Это изменение включает только изменение внутреннего флага.

Оператор REPAIR TABLE позволяет задавать следующие параметры для определения режима восстановления.

- EXTENDED. Попытка восстановления с воссозданием индексов. (Аналогично использованию программы myisamchk с параметром `--safe-recover`.)
- QUICK. Попытка быстрого восстановления только индексов.
- USE_FRM. Попытка восстановления по файлу описания таблицы `.frm`. Восстановление воссоздает индекс на основании описания таблицы. Существенно то, что он автоматизирует процедуру, описанную выше, в которой используется файл `.frm` для перестройки индекса по описанию таблицы, когда индекса нет или он непригоден к использованию, это может пригодиться также, если индекс был утерян или разрушен. Данный параметр был добавлен в MySQL 4.0.2.

При отсутствии параметра оператор REPAIR TABLE выполняет стандартную операцию восстановления, которая производится с параметром `--recover`.

Программа mysqlcheck обеспечивает интерфейс в командной строке для операторов CHECK TABLE и REPAIR TABLE. Эта программа подключается к серверу и запускает соответствующие операторы, опираясь на заданные вами параметры.

Такая возможность появилась в MySQL 3.23.38 и заключается в проверке таблиц MyISAM (и начиная с MySQL 3.23.39 – таблиц InnoDB).

Обычно программа mysqlcheck вызывается с именем базы данных, которое может дополняться необязательными именами таблиц. Если задано только имя базы данных, программа mysqlcheck проверяет все таблицы в базе данных:

```
% mysqlcheck sampdb
```

При указанных именах таблиц, следующих за именем базы данных, программа проверяет только указанные таблицы:

```
% mysqlcheck sampdb president member
```

Если указан параметр `--databases`, все последующие имена интерпретируются как имена баз данных и программа `mysqlcheck` проверяет все таблицы из каждой указанной базы данных:

```
% mysqlcheck --databases sampdb test
```

Если задан параметр `--all-databases`, программа проверяет все таблицы во всех базах данных. В таком случае имена таблиц или баз данных не указываются:

```
% mysqlcheck --all-databases
```

По умолчанию программа `mysqlcheck` проверяет таблицы с использованием стандартной процедуры проверки, но поддерживает параметры, позволяющие детализировать тип выполняемой операции.

В следующей таблице приведены некоторые параметры программы `mysqlcheck` и соответствующие им параметры `CHECK TABLE`.

Параметры mysqlcheck	Параметры CHECK TABLE
`--check-only-changed`	`CHANGED`
`--extended`	`EXTENDED`
`--fast`	`FAST`
`--medium-check`	`MEDIUM`
`--quick`	`QUICK`

Программа `mysqlcheck` также позволяет выполнять операции восстановления. В следующей таблице приведены некоторые параметры программы `mysqlcheck` и соответствующие им параметры `REPAIR TABLE`.

Параметры mysqlcheck	Параметры REPAIR TABLE
`--repair`	Параметры отсутствуют (выполнение стандартной операции восстановления)
`--repair -quick`	`QUICK`
`--repair -extended`	`EXTENDED`

Программа `mysqlcheck` обеспечивает удобный метод задания операторов `CHECK TABLE` и `REPAIR TABLE` напрямую, так как для них требуется непосредственно указать имя восстанавливаемой или проверяемой таблицы. Программа `mysqlcheck` просматривает имена таблиц и конструирует операторы, задающие соответствующие таблицы.

Планирование превентивного обслуживания

В дополнение к активизации автоматического восстановления и разработке процедур резервного копирования, описанным выше, следует разработать план превентивной поддержки. Это поможет определить проблемы автоматически и позволит предпринять меры по их предотвращению. Организуя регулярную про-

верку таблиц, вы снижаете вероятность обращения к резервным копиям в дальнейшем. Это проще всего реализуется с помощью заданий утилиты cron, обычно вызываемой из файла crontab учетной записи, используемой для работы сервера. Например, при запуске сервера под управлением пользователя mysqladm можно задать периодическую проверку из файла crontab для этого пользователя. (О том, как программируются cron-задания, см. в главе 11.)

Для периодической автоматической проверки таблиц MyISAM и InnoDB во время работы сервера воспользуйтесь программой mysqlcheck. Предположим, что требуется запустить программу mysqlcheck из файла crontab для пользователя mysqladm. Для этого необходимо добавить в файл crontab следующую запись:

```
0 3 * * 0 /usr/local/mysql/bin/mysqlcheck --all-databases
--check-only-changed --silent
```

(Как здесь показано, команда занимает две строки, но в самом файле они записываются в одной строке.) Эта запись задает утилите cron запуск программы mysqlcheck каждую субботу в 3 часа дня. Время запуска программы можно задать по желанию.

Параметр --all-databases задает проверку всех таблиц во всех базах данных программой mysqlcheck. Это самый простой способ получения максимального эффекта от нее. Параметр --check-only-changed задает при проверке mysqlcheck режим пропуска любой таблицы, которая не была модифицирована после последней проверки, параметр --silent подавляет вывод в том случае, если в таблицах не было обнаружено ошибок. (Задание cron обычно генерирует почтовое сообщение, но только в том случае, если вообще создает какое-либо сообщение, потому что нет никакого смысла в посылке сообщений о том, что программа проверки таблиц ошибок не обнаружила.) Но будьте готовы к тому, что программа mysqlcheck может выдать сообщение о том, что в базе данных ею обнаружены таблицы таких типов, которые она не знает, как проверить.

Для проверки таблиц типа ISAM или MyISAM, но устаревшей версии MySQL, не включающей mysqlcheck, можно использовать isamchk и myisamchk. Для выполнения автоматической проверки таблицы достаточно написать простейший сценарий, который запускает эти программы на всех таблицах MyISAM и ISAM в заданном каталоге. Вот пример такого сценария:

```
#! /bin/sh
# chk_mysql_tables.sh - проверить все таблицы MyISAM/ISAM в заданном
каталоге
# Аргумент 1: каталог
if [ $# -ne 1 ]; then
   echo "Usage: $0 dirname" 1>&2
   exit 1
fi
# Перейти в каталог и проверить все таблицы в нем.
# Примечания:
# - до MySQL 3.23.22 замените параметр --check-only-changed на --fast
# - isamchk не поддерживает --check-only-changed или --fast
cd $1
if [ $? -ne 0 ]; then
   echo "Невозможно перейти в $1" 1>&2
   exit 1
fi
```

```
find . -name "*.MYI" -follow -print \
  | xargs myisamchk --silent --check-only-changed
find . -name "*.ISM" -follow -print \
  | xargs isamchk --silent
```

Для того чтобы проверить таблицы одного типа из всех проверяемых в этом сценарии, пропустите команды проверки таблиц других типов. Например, если у вас нет таблиц ISAM, можно не задавать команду `isamchk`, так как в этом случае команда `find` не снабдит ее именами файлов и программа `isamchk` отобразит сообщение "usage", показывающее, что команде `isamchk` не достает аргументов.

С помощью команды `chmod +x` сделайте сценарий исполняемым, а затем вызовите его, указав путь к вашему каталогу данных, чтобы проверить все таблицы MyISAM и ISAM:

```
% chk_mysql_tables.sh /usr/local/mysql/data
```

Для того чтобы проверить только таблицы данной базы данных, вызовите сценарий, задав путь только к каталогу соответствующей базы данных:

```
% chk_mysql_tables.sh /usr/local/mysql/data/sampdb
```

Для автоматического запуска этого сценария запрограммируйте его как задание утилиты `cron`. При увеличении количества серверов в вашей системе можно запускать сценарий `chk_mysql_tables.sh` несколько раз, каждый раз задавая в качестве аргумента различные каталоги данных.

В идеальном случае от сценария `chk_mysql_tables.sh` не должно быть никакого вывода. Однако, если внешняя блокировка отключена (это делается по умолчанию начиная с MySQL 4), вполне возможно, что сервер внесет изменения в таблицу в то время, когда будет происходить ее проверка. Сценарий всего лишь проверяет таблицы и не делает никаких попыток восстановить их, а следовательно, не может их каким-либо образом повредить, но в этом случае программа `myisamchk` или `isamchk` может сообщить о несуществующих ошибках, хотя таблицы находятся в полном порядке. (Другими словами, имеет место определенное несоответствие, но это значительно лучше, чем если отчеты программ сообщают о полном отсутствии проблем, в то время как в действительности определенные проблемы существуют.) Если ваша система поддерживает внешнюю блокировку, такая проблема не возникнет. Подробнее о проблеме внешней блокировки см. в разделе "Координация работы с сервером".

Кроме того, сценарий `chk_mysql_tables.sh` можно запустить во время загрузки. При работе с BSD-системами команда запуска сервера MySQL размещается в каталоге `/etc/rc.local` (или можно запустить сценарий `chk_mysql_tables.sh` из этого же файла перед запуском сервера). В системах типа System-V загляните в каталог `/etc/rc.d`.

Можно также запустить сценарий `chk_mysql_tables.sh`, воспользовавшись сценарием `rc.sysinit` или ему подобными сценариями, до запуска сервера MySQL.

Использование резервных копий для восстановления данных

Процедуры восстановления данных используют два источника информации — ваши файлы с резервными копиями и двоичные журналы. Файлы резервных ко-

пий восстанавливают то состояние таблиц, которое они имели во время выполнения резервного копирования. Однако содержимое таблиц обычно с момента проведения резервного копирования и до момента возникновения проблем претерпевает изменения. Запросы, которые привели к таким изменениям, содержатся в двоичных журналах, поэтому у нас есть возможность повторить их. (Вот зачем нам нужно было двоичное протоколирование. И если вы еще этого не сделали, запустите двоичное протоколирование и сделайте резервную копию, перед тем как продолжить чтение.)

Процедуры восстановления варьируются в зависимости от того, как много информации в них задействовано. В действительности может оказаться значительно проще восстановить всю базу данных, чем отдельную таблицу, так как применять журналы обновлений ко всей базе данных значительно проще, чем к одной таблице.

Какие журналы следует использовать для выполнения операций восстановления



налы обновлений и двоичные журналы обновлений. Двоичные журналы обновлений лучше всего подходят для обновления, поэтому их использование значительно предпочтительнее, чем использование журналов обновления. При изложении материала данной главы предполагается, что используются двоичные журналы. Однако двоичное резервирование появилось только в MySQL 3.23.14. Если у вас нет такой возможности, используйте журналы обновления и соответствующим образом скорректируйте инструкции.

Восстановление всей базы данных

Если восстанавливаемая база данных является базой данных `mysql`, которая содержит таблицы разрешения, сервер прежде всего должен работать с параметром `--skip-grant-tables`. В противном случае процедура восстановления будет выдавать диагностическое сообщение о том, что не может найти таблицы разрешения. Очень кстати может оказаться применение параметра `--skip-networking`, для того, чтобы отклонить все попытки подключения во время выполнения восстановления. После восстановления таблиц выключите сервер и перезапустите его таким образом, чтобы он использовал таблицы разрешения и прослушивал сетевые интерфейсы, как он обычно это делает.

Общая процедура восстановления предусматривает следующие шаги.

1. Скопируйте содержимое каталога базы данных в другое место. Они могут пригодиться позднее — например, в процессе изучения "останков" разрушенных таблиц.
2. Перезагрузите базу данных, пользуясь последней резервной копией. При использовании резервных копий, созданных программой `mysqldump` как файлы, содержащие операторы SQL, воспользуйтесь ими для ввода в `mysql`. При использовании файлов, полученных в результате непосредственного копирования из базы данных (например, с помощью утилит `mysqlhotcopy`, `tar` или `cp`), отключите сервер, скопируйте файлы в каталог базы данных и перезапустите сервер. (Причина отключения сервера заключается в том, что необходимо блокировать доступ к файлам во время выполнения операции копирования.)

3. Использование двоичных журналов обновления для повторения запросов, которые модифицируют таблицы базы данных после того, как было произведено резервное копирование. С помощью программы `mysqlbinlog` преобразуйте в формат ASCII нужный журнал и используйте полученный результат для ввода в `mysql`. Задайте параметр `--one-database` таким образом, чтобы `mysql` выполнял запросы только для той базы данных, которую вы хотите восстановить.

Если известно, что вам необходимо применить все регистрационные файлы, воспользуйтесь следующей командой, открыв каталог, в котором они размещены:

```
% ls -t -r -1 binlog.[0-9]* | xargs mysqlbinlog | mysql --one-database
db_name
```

Команда `ls` позволяет получить список файлов журналов обновлений, размещая их в столбец, отсортированных по имени. Этот порядок сортировки принят по умолчанию. Однако, если цифровые расширения имен файлов имеют не одинаковое количество цифр, это не будет работать. Например, если журналы имеют имена `binlog.998`, `binlog.999` и `binlog.1000`, команда `ls` отсортирует их таким образом, что первым будет расположен файл `binlog.1000`. Фильтрация имен посредством сортировки работать не будет, так как она сортирует имена определенным способом. Необходимо выполнить цифровую сортировку, опираясь только на значения расширений. Это делает следующий небольшой сценарий Perl:

```
#! /usr/bin/perl -w
# ext_num_sort.pl - сортирует имена, опираясь на значения цифровых
расширений.
use strict;
my @files = <>; # считывать все вводимые строки
@files = sort { # сортировать по цифровому расширению
    my $anum = $1 if $a =~ /\.(\d+)$/;
    my $bnum = $1 if $b =~ /\.(\d+)$/;
    $anum <=> $bnum;
  } @files;
print @files; # распечатать их
exit (0);
```

Использовать этот сценарий можно следующим образом:

```
% ls -1 binlog.[0-9]* | ext_num_sort.pl | xargs mysqlbinlog \
  | mysql --one-database db_name
```

В предыдущем обсуждении предполагалось, что применяются все журналы обновления, но зачастую регулярно приходится обращаться к помощи лишь некоторых из них, например тех, которые записаны после конкретного резервного копирования. Если журналы, сделанные после последнего резервного копирования, имеют имена `binlog.1392`, `binlog.1393` и т.д., эти операторы можно запустить следующим образом:

```
% mysqlbinlog binlog.1392 | mysql --one-database db_name
% mysqlbinlog binlog.1393 | mysql --one-database db_name
...
```

Если причина выполнения процедуры восстановления и использования журналов для восстановления информации заключается в том, что кто-то воспользовался злосчастными операторами типа `DROP DATABASE`, `DROP TABLE` или `DELETE`, обязательно перед запуском процедуры восстановления удалите эти опе-

раторы из регистрационного журнала! Для этого преобразуйте его в формат ASCII и сохраните его в файле. Затем отредактируйте этот файл и передайте полученный результат на вход mysql:

```
% mysqlbinlog logfile > textfile
% vi textfile
% mysql --one-database db_name < textfile
```

Если в вашем распоряжении нет двоичного файла обновлений, но имеются журналы обновлений, в использовании программы mysqlbinlog нет необходимости; этот журнал уже должен быть в формате ASCII. В этом случае, для того, чтобы применить все обновления для данной базы данных одновременно, сделайте следующее:

```
% ls -1 update.[0-9]* | ext_num_sort.pl | xargs cat \
  | mysql --one-database db_name
```

Для того чтобы воспользоваться данными из одного журнала, выполните следующее:

```
% mysql --one-database db_name < update.1392
% mysql --one-database db_name < update.1393
...
```

Проблема получения неправильной сортировки из-за различной длины расширений файлов журналов регистрации в будущем будет решена. Начиная с определенной версии (вероятно, с серии 4.1) сервер будет настроен для использования минимум шести цифр (а не только трех) в расширении файла. Это изменение сделает сценарий ext_num_sort.pl совершенно бесполезным.

Восстановление отдельных таблиц

Восстановление отдельных таблиц может оказаться более сложной задачей, чем восстановление целой базы данных. Если у вас есть файл резервной копии, созданный с помощью программы mysqldump, и он содержит резервные копии данных множества таблиц, а не только той, данные которой вас интересуют (что более вероятно), вам потребуется выбрать часть, касающуюся этого файла, и использовать ее в качестве вводного файла программы mysql. И это еще не самое сложное! Сложнее будет выбрать части, касающиеся журнала, который относится только к этой таблице. Программа mysql_find_rows может очень в этом помочь; она может помочь выбрать запросы, состоящие из нескольких строк, из журналов обновления или из двоичных журналов обновления, которые можно преобразовать в ASCII программой mysqlbinlog. Другая возможность заключается в восстановлении базы данных во вторую еще пустую базу данных.

Получите дамп этой таблицы с помощью программы mysqldump --add-drop-table, после чего загрузите ее обратно в исходную базу данных. (Параметр --add-drop-table гарантирует запуск операции восстановления с нуля.) Это позволяет упростить процедуру восстановления одной таблицы в сравнении с процедурой выборки соответствующих строк из журналов обновления. Другая возможность (она не касается таблиц типа BDB или InnoDB) заключается в копировании файлов из второй базы данных в каталог данных исходной базы данных. При копировании обеих баз данных в каталог баз данных убедитесь в том, что сервер для них отключен.

Восстановление таблиц, имеющих внешние ключи

Если у вас возникли проблемы с использованием файла дампа для восстановления таблиц, имеющих внешние ключи, поскольку эти таблицы не перечисляются в файле в порядке, в котором они необходимы для восстановления этих взаимоотношений, вы можете временно отключить проверку отношений внешних ключей с помощью оператора

```
SET FOREIGN_KEY_CHECKS = 0;
```

Параметр FOREIGN_KEY_CHECKS впервые появился в MySQL 3.23.52. После того как таблицы импортированы, проверку ключей можно восстановить:

```
SET FOREIGN_KEY_CHECKS = 1;
```

Выключение проверки позволяет создать и загрузить таблицы в любом порядке. Это также ускоряет загрузку. Отключать проверку ключей необходимо в тот же сеанс соединения, который используется для перезагрузки файла дампа, так как эта установка влияет только на соединение.

Это рекомендуется делать в диалоговом режиме работы mysql, а не в командной строке. Предположим, что у нас в распоряжении имеется файл дампа dump.sql, содержащий таблицы из базы данных mydata. Загрузить его можно следующим образом:

```
% mysql mydata
mysql> SET FOREIGN_KEY_CHECKS = 0;
mysql> source dump.sql;
mysql> SET FOREIGN_KEY_CHECKS = 1;
mysql> ...
```

Второй оператор SET необходим, только если вы намерены задавать еще операторы в пределах сеанса mysql после загрузки файла дампа. Если после загрузки файла сеанс работы mysql будет завершен, необходимость во втором операторе SET отпадает.

Восстановление таблицы с помощью команды RESTORE TABLE

Оператор RESTORE TABLE является "напарником" оператора BACKUP TABLE и позволяет восстановить таблицы MyISAM, которые были скопированы с помощью последнего. Он появился в MySQL 3.23.25, и для работы с ним требуются привилегии FILE и INSERT. Для того чтобы использовать оператор RESTORE TABLE, задайте имя таблицы (или таблиц), которую вы планируете восстановить вместе с каталогом на серверном узле, где размещаются файлы резервных копий. Предположим, что сначала с помощью оператора BACKUP TABLE следующей командой были зарезервированы три таблицы:

```
BACKUP TABLE tbl1, tbl2, tbl3 TO '/archive/sampdb';
```

Для того чтобы восстановить любую из этих таблиц или все таблицы сразу, задайте эти таблицы в операторе RESTORE TABLE. Например, для восстановления таблиц tbl1 и tbl3 необходимо:

```
RESTORE TABLE tbl1, tbl3 FROM '/archive/sampdb';
```

Оператор RESTORE TABLE перезагружает файл описания и файл данных таблицы .frm и .MYD в каталог базы данных, а затем использует их при создании индексного файла .MYI.

Восстановление табличного пространства InnoDB или таблиц BDB

Дескриптор таблиц InnoDB автоматически делает попытку выполнить все необходимое восстановление при перезапуске сервера после краха. Однако в случае, когда дескриптор InnoDB сталкивается с невосстанавливаемой ошибкой, такой запуск завершится безуспешно. В этом случае установите переменную сервера `innodb_force_recovery` в значение от 1 до 6, чтобы запустить сервер, даже если восстановление табличного пространства InnoDB после краха завершается неуспешно. Для того чтобы установить эту переменную, добавьте в параметрическую группу `[mysqld]` следующий параметр:

```
set-variable = innodb_force_recovery=level
```

Чем меньше значение `level`, тем более консервативные стратегии пытается применить дескриптор InnoDB. Рекомендуется начинать с уровня 4. После того как сервер запущен, сделайте дамп таблиц InnoDB для получения как можно более полной информации о них, удалите таблицы и восстановите их из файла вывода `mysqldump`. Эта процедура позволит восстановить файлы в виде, который является внутренне целостным и которого может оказаться достаточно для того, чтобы добиться вполне приличного восстановления. Выполнив восстановление, удалите из конфигурационного файла строку, устанавливающую параметр `innodb_force_recovery`.

Если необходимо восстановить все табличное пространство InnoDB, используемый подход зависит от того, каким образом была сделана резервная копия. (Предполагается, что одна копия уже была вами создана на основании рекомендаций из раздела "Восстановление табличного пространства InnoDB или таблиц BDB".)

- При использовании метода копирования необходимо иметь копии файлов табличного пространства, файлов регистрации и файла `.frm` для каждой таблицы и конфигурационного файла, определяющего вашу конфигурацию InnoDB. После того как вы убедились в том, что сервер отключен, удалите любой существующий файл InnoDB и замените его резервными копиями. Затем убедитесь в том, что ваш конфигурационный файл повторяет конфигурацию InnoDB так же, как и сохраненный конфигурационный файл, и перезапустите сервер.
- Для резервирования табличного пространства InnoDB запустите программу `mysqldump`. Это необходимо для создания файла SQL, содержащего операторы `CREATE TABLE` и `INSERT`, которые впоследствии потребуются для восстановления таблиц после краха. Затем проинициализируйте табличное пространство вновь и загрузите в него полученный файл дампа. При отключенном сервере уберите все существующие файлы, имеющие отношение к типу InnoDB: табличное пространство (если это не просто раздел), файлы регистрации и файлы `.frm` для всех таблиц InnoDB. Переконфигурируйте табличное пространство так же, как это было сделано в начале. (См. раздел "Настройка табличного пространства InnoDB" в главе 11. Сохраненная копия конфигурационного файла сервера может также помочь в качестве напоминания о том, какой вид должна иметь настройка. Напомним также, что при применении разделов инициализация табличного пространства представляет собой двухэтапный процесс.) Когда сервер завершит создание нового табличного пространства, перезагрузите файл резервной копии, со-

держащий операторы SQL для восстановления таблиц InnoDB, используя его в качестве ввода для `mysql`.

После восстановления табличного пространства из резервной копии повторите все обновления на основании двоичных журналов, которые происходят после резервного копирования. Задача значительно упрощается при воссоздании табличного пространства как части процедуры восстановления полного набора баз данных, так как в этом случае можно использовать сразу все обновления. При восстановлении только табличного пространства InnoDB круг применения журналов значительно сужается, поскольку необходимо использовать только обновления таблиц InnoDB.

Дескриптор BDB, как и дескриптор InnoDB, делает попытку автовосстановления при запуске сервера после краха. Если из-за невосстановимой проблемы BDB запуск сервера завершается неуспешно, перенесите все журналы регистрации BDB из каталога данных в какой-либо другой каталог (или удалите их, если не собираетесь анализировать их в дальнейшем). После этого запустите сервер с параметром `--bdb-no-recover`. Если разрушенными являются и файлы журналов, это позволяет запуститься серверу и создать новый журнал BDB. Если сервер по-прежнему не запускается, попробуйте заменить BDB-файлы из резервных копий.

- При непосредственном копировании соответствующих файлов необходимо иметь в своем распоряжении файлы таблиц BDB и файлы журналов BDB. При отключенном сервере удалите из каталога данных существующие BDB-файлы и замените их резервными копиями.
- Если для создания резервных копий, состоящих из операторов SQL, использовалась программа `mysqldump`, отключите сервер и удалите существующие таблицы и журналы BDB. Перезапустите сервер и загрузите файл резервной копии, используя его в качестве ввода в `mysql`.

Восстановив резервную копию, повторите все обновления, сделанные после резервного копирования, взяв их из журналов (памятуя при этом о тех подвохах, которые кроются в восстановлении таблиц InnoDB).

IV

Приложения

В этой части...

Приложение А. Получение и инсталляция программного обеспечения

Приложение Б. Типы столбцов

Приложение В. Операторы и функции

Приложение Г. Синтаксис SQL

Приложение Д. Программы MySQL

Приложение Е. Программный интерфейс приложений C

Приложение Ж. Программный интерфейс DBI языка написания сценариев Perl

Приложение З. Программный интерфейс языка PHP

Приложение И. Утилиты независимых разработчиков

A

Получение и инсталляция программного обеспечения

В этом приложении...

Получение и инсталляция дистрибутива базы данных `sampdb`
Получение MySQL и сопутствующего программного обеспечения
Инсталляция MySQL в системе UNIX
Инсталляция MySQL в Windows

В этом приложении рассматриваются вопросы получения и инсталляции программного обеспечения, необходимого для запуска MySQL. Здесь же рассказывается обо всем сопутствующем MySQL программном обеспечении независимых разработчиков, например поддержка Perl DBI и CGI:pm, PHP и Apache, а также о том, как можно получить файлы, связанные с базой данных `sampdb`, используемой в примерах этой книги. Цель написания этого приложения — объединение всех инструкций по инсталляции различных пакетов. Автор искренне надеется, что эти инструкции будут внимательно изучены читателями. В данном приложении собраны общие указания, подходящие к большинству инсталляций, хотя каждый программный дистрибутив, помимо всего прочего, включает дополнительные инструкции, которые могут оказаться весьма кстати, если в процессе выполнения стандартной процедуры возникают проблемы. Справочное руководство по MySQL, например, включает большую главу, посвященную процедуре инсталляции и решению множества проблем, возникающих во время работы с теми или иными системами.

Получение и инсталляция дистрибутива базы данных sampdb

Файлы, необходимые для установки и настройки доступа к базе данных sampdb, доступны по следующему адресу:

```
http://www.kitebird.com/mysql-book/
```

Дистрибутив sampdb может быть получен в виде сжатого утилитой tar файла или в виде архивного ZIP-файла. Распаковать файл tar-формата можно с помощью одной из следующих команд (вторую команду можно использовать в том случае, если ваша версия утилиты tar не "понимает" параметр z):

```
% tar zxf samp_db.tar.gz
% gunzip < samp_db.tar.gz | tar xf -
```

Для распаковки дистрибутива ZIP-формата воспользуйтесь такими утилитами, как WinZip, pkunzip или unzip.

Распаковка дистрибутива приведет к созданию каталога sampdb, содержащего несколько следующих компонентов.

- Файл README, содержащий общие инструкции по использованию дистрибутива. Этот файл следует прочитать в первую очередь. Отдельные подкаталоги дистрибутива также могут содержать файлы README с более детальной информацией.
- Файлы, необходимые для создания и загрузки базы данных sampdb. Как их использовать, рассказывается в главе 1, "Знакомство с СУБД MySQL и SQL".
- Каталог capi, содержащий C-программы, описанные в главе 6, "Программный интерфейс MySQL C API".
- Каталог perlapi, содержащий сценарии Perl DBI, описанные в главе 7, "Программный интерфейс Perl API".
- Каталог phpapi, содержащий PHP-сценарии, описанные в главе 8, "Интерфейс API для языка написания сценариев PHP".

В каталог sampdb вложено несколько других каталогов, содержащих файлы, на которые даются ссылки в данной книге.

Получение MySQL и сопутствующего программного обеспечения

В процессе инсталляции MySQL следует устанавливать только те средства независимых разработчиков, которые планируется использовать в дальнейшем.

- Если вы планируете создание Perl-сценариев для получения доступа к базам данных MySQL, необходимо инсталлировать модули DBI и DBD::mysql. Если же в ваши планы входит создание DBI-сценариев, которые обеспечивают Web-доступ к базам данных, наверняка также возникнет необходимость в инсталляции модуля CGI.pm и Web-сервера. В настоящей книге описывается использование сервера Apache, хотя с этой же целью можно применять и другие серверы.

- Если планируется написание PHP-сценариев, обеспечивающих доступ к базам данных MySQL, необходимо инсталлировать поддержку средств программирования клиентов MySQL и PHP. Модуль PHP применяется, как правило, для написания Web-сценариев, что, в свою очередь, требует наличия Web-сервера. В этой книге описывается сервер Apache, поскольку он включает модульную поддержку PHP.

Для большинства инсталляционных пакетов доступны заранее откомпилированные двоичные версии. Для инсталляции в Linux имеются дистрибутивы в виде RPM-файлов. Администраторы, которые предпочитают самостоятельно компилировать программное обеспечение из исходного кода или не могут найти двоичную версию для своей платформы, должны иметь в своем распоряжении компилятор C (C++ для MySQL).

При предоставлении MySQL-хостинга провайдером услуг Internet, вполне вероятно, что все эти пакеты уже установлены. В этом случае с ними можно работать и не читать данное приложение.

В представленной ниже таблице указываются основные адреса узлов, на которых можно найти необходимые пакеты. Некоторое программное обеспечение можно загрузить с нескольких узлов.

Пакет	Адрес
MySQL	`http://www.mysql.com/`
Модули Perl	`http://cpan.perl.org/`
PHP	`http://www.php.net/`
Apache	`http://www.apache.org/`

Выбор версии инсталлируемого пакета в первую очередь определяется задачами, стоящими перед администратором.

- Если необходимо обеспечить максимальную стабильность, возможно, следует пойти по более консервативному пути и воспользоваться самой последней (не бета-версией) версией пакета. Такой выбор позволит воспользоваться преимуществами самых новых возможностей и застраховать себя от ошибок предыдущих версий, а также избавит от исследования экспериментального кода бета-версий.
- Администраторы, желающие всегда быть на гребне обновлений или являющиеся разработчиками, конечно, захотят использовать только самую последнюю бета-версию. Это относится и к тем администраторам, которым для работы нужны возможности, имеющиеся только в последней опытной версии.
- Заранее откомпилированные двоичные и RPM-дистрибутивы MySQL зачастую создаются с помощью специальных файлов оптимизации. Эти файлы работают гораздо эффективней, чем создаваемые сценарием конфигурации из исходного дистрибутива. Кроме того, разработчики MySQL создают дистрибутивы с помощью определенных коммерческих оптимизирующих компиляторов, что еще больше ускоряет работу MySQL.

  Соответственно, программы в дистрибутивах такого типа будут работать быстрей, чем в откомпилированных администратором. Поэтому если одним из

важнейших требований является требование высокой производительности, рекомендуется остановиться на заранее откомпилированном дистрибутиве.

На перечисленных выше Web-узлах для каждого пакета указывается также, какая версия является последней официально выпущенной и какие бета-версии опытной серии имеются. Кроме того, на них можно найти также список изменения возможностей различных версий, который может оказаться весьма полезным при выборе наиболее подходящей инсталляции.

Распаковка двоичного или RPM-дистрибутива практически аналогична инсталляции, поскольку файлы распаковываются в указанные администратором каталоги. Чтобы распаковать дистрибутив в защищенные каталоги, администратор должен зарегистрироваться под именем пользователя `root`.

Исходный дистрибутив можно сначала распаковать в отдельный каталог для компиляции, а затем уже инсталлировать в соответствующие инсталляционные каталоги. Для выполнения инсталляции также необходимо зарегистрироваться в качестве пользователя `root`. Этого, однако, не требуется на этапах настройки и компиляции.

Инсталляция некоторых пакетов MySQL из исходного кода в ОС UNIX требует применения утилиты `configure`, которая упрощает установку и настройку программного обеспечения во многих системах. Если подобная установка не приводит к желаемому результату, возможно, нужно перезапустить утилиту `configure` с другими параметрами. В этом случае необходимо убедиться, что утилита `configure` не будет использовать собранную ранее информацию о конфигурации. Чтобы очистить ее, воспользуйтесь следующей командой:

```
% make distclean
```

С этой же целью можно воспользоваться и такими командами:

```
% rm config.cache
% make clean
```

Чтобы получить помощь, подпишитесь на список рассылки

После инсталляции нового программного пакета рекомендуется подписаться на общий список рассылки, посвященный данному программному обеспечению. Это позволит задавать интересующие вопросы и получать на них исчерпывающие ответы. Особенно рекомендуется подписаться на этот список пользователям бета-версий, чтобы вовремя узнавать о выявленных ошибках кода и иметь возможность их устранить. Если вы не желаете подключаться к общей дискуссии, подпишитесь хотя бы на список анонсов, чтобы получать уведомления о новых версиях. Информацию о подписке можно получить на Web-узлах соответствующих пакетов.

Инсталляция MySQL в системе UNIX

Существуют UNIX-дистрибутивы для различных версий MySQL. В настоящее время последними официальными являются версии серии 4.0, а в качестве разрабатываемых опытных уже выступают версии серии 4.1. В общем, при выборе необходимо отдавать предпочтение наибольшему номеру версии в серии. Выбрать можно также и соответствующий формат дистрибутива.

Дистрибутив MySQL может быть представлен в двоичном формате, в виде RPM-файлов и исходных текстов. Дистрибутивы первых двух форматов легче ус-

танавливать, однако при их выборе администратор получает стандартные параметры структуры и конфигурации. Дистрибутивы, представленные в виде исходного кода, инсталлировать трудней, поскольку сначала необходимо скомпилировать соответствующее программное обеспечение, а также уметь управлять параметрами конфигурации. Можно, например, скомпилировать дистрибутив только для клиентской поддержки, убрав компонент сервера, или изменить местоположение инсталляции.

В общем случае, дистрибутивы включают один или несколько следующих компонентов.

- Сервер `mysqld`.
- Клиентские программы (`mysql`, `mysqladmin` и т.п.) и программная клиентская поддержка (файлы библиотек и заголовков).
- Документация.
- Исходная база данных.
- Языковая поддержка.

Двоичные и представленные в виде исходного кода дистрибутивы содержат все эти элементы. Отдельный RPM-файл содержит только некоторые из них, поэтому для установки всех компонентов потребуется инсталлировать несколько RPM-файлов.

Если планируется подключение к серверу, уже работающему на другом компьютере, программное обеспечение сервера специально инсталлировать не нужно. Достаточно установить клиентское программное обеспечение.

- Если сервер не устанавливается, необходимо настроить клиентское программное обеспечение, чтобы иметь возможность подключаться к серверу, уже работающему на другом компьютере.
- Если сервер устанавливается, необходимо настроить средства подключения к серверу с компьютера, на котором он запущен. Это исключит необходимость регистрации на другом компьютере, использующем клиентское программное обеспечение, для тестирования работы сервера.
- Клиентская C-библиотека требуется для написания программ с использованием API-интерфейса. Например, она может потребоваться администратору для создания DBI-интерфейса, который необходим для создания сценариев Perl, работающих с MySQL.

Еще об инсталляции MySQL

Чтобы инсталлировать MySQL в системе UNIX, выполните следующие действия.

1. Создайте учетную запись UNIX для пользователя и группы, от имени которых будет работать сервер (если планируется установка сервера).
2. Получите и распакуйте дистрибутив, который собираетесь инсталлировать. Если для инсталляции применяется исходный код, скомпилируйте его и приступайте к его инсталляции.

3. Запустите сценарий `mysql_install_db` для инициализации каталога данных и таблиц разрешений (только при первой инсталляции).
4. Запустите сервер.
5. Прочитайте главу 11, "Общее администрирование MySQL", чтобы ознакомиться с общими административными функциями. В частности, настоятельно рекомендуется прочитать раздел, посвященный запуску и завершению работы сервера и запуску сервера от имени пользователя без привилегий.

Создание учетной записи для пользователя MySQL

Пользователи, которые работают с клиентским программным обеспечением MySQL и не собираются заниматься установкой сервера MySQL, могут пропустить материал этого раздела.

Сервер MySQL может работать с полномочиями любого UNIX-пользователя, однако по причине безопасности и эффективного администрирования лучше не запускать сервер от имени пользователя `root`. Рекомендуется создать отдельную учетную запись для администрирования MySQL и именно под ее управлением запускать сервер. При таком подходе администратор сможет зарегистрироваться от имени этого пользователя, получив все привилегии в каталоге данных для выполнения операций поддержки и разрешения проблем. Процедуры создания учетных записей в различных системах могут различаться, поэтому прежде чем приступить к работе, внимательно ознакомьтесь с имеющейся документацией.

В этой книге в качестве имени пользователя и группы этой учетной записи используются соответственно `mysqladm` и `mysqlgrp`. Если какой-либо пользователь планирует инсталлировать СУБД MySQL исключительно для своих потребностей, он может запускать ее от своего имени. В этом случае `mysqladm` и `mysqlgrp` следует заменить своими именами пользователя и группы. При инсталляции из RPM-файлов автоматически создается пользователь с именем `mysql`. Соответственно, при выборе именно такой инсталляции вместо `mysqladm` необходимо подставить `mysql`.

Создание и использование для запуска MySQL отдельной учетной записи пользователя без особых привилегий по сравнению с использованием записи `root` имеют следующие преимущества.

- Если MySQL не работает с полномочиями пользователя `root`, никто не сможет воспользоваться сервером для получения полного доступа к системе.
- Административные функции MySQL безопасней выполнять от имени пользователя без особых привилегий, чем как пользователь `root`.
- Владельцем созданных сервером файлов будет пользователь `mysqladm`, а не `root`. Чем меньше файлов в системе связаны с пользователем `root`, тем лучше.
- Даже исходя из соображений простоты работы лучше выделить для функционирования MySQL отдельную учетную запись. Это позволит также легче определить, какие элементы системы связаны с MySQL. Например, в каталоге с файлами `cron` будет храниться и отдельный файл для MySQL-пользователя, `mysqladm`. В любом другом случае задание процесса `cron` MySQL будет записываться в файл пользователя `root` вместе со всеми остальными операциями, периодически выполняемыми пользователем `root`.

Получение и инсталляция дистрибутива MySQL

При изложении последующего материала под *версией* будет пониматься номер версии используемого дистрибутива MySQL, а под *платформой* — название платформы, на которой выполняется инсталляция. Эти же понятия зачастую используются и в именах файлов дистрибутива, что позволяет легко определять их и различать. Номер версии представляет собой запись наподобие `3.23.52` или `4.0.5-beta`, а платформа — `sun-solaris2.8-sparc` или `pc-linux-gnu-i686`.

Файлы с двоичными дистрибутивами, имеющие в своем имени `-max`, включают в свой состав сервер с дополнительными возможностями, которых не имеет стандартный сервер. Возможности сервера периодически меняются, поэтому необходимо их периодически проверять на Web-узле MySQL. Это позволит посмотреть, насколько сейчас отличаются стандартная и максимальная версии. При компилировании из исходного текста возможности можно отключать с помощью сценария `configure`, запускаемого с соответствующими параметрами.

Инсталляция двоичного дистрибутива

Имена файлов двоичного дистрибутива записываются, как правило, в формате `mysql-`*`version-platform`*`.tar.gz`. Загрузите или получите иным путем файл дистрибутива выбранной версии и соответствующей платформы и поместите его в каталог инсталляции MySQL, например `/usr/local`.

Для распаковки дистрибутива запустите одну из следующих команд (вторую команду используйте, если рабочая версия `tar` не распознает параметр `z` при распаковке дистрибутива):

```
% tar zxf mysql-version-platform.tar.gz
% gunzip < mysql-version-platform.tar.gz | tar xf -
```

Распаковка дистрибутива сопровождается созданием каталога `mysql-`*`version-platform`*, в который и помещается ее содержимое. Чтобы упростить ссылку на этот каталог, создайте символическую связь с помощью следующей команды:

```
% ln -s mysql-version-platform mysql
```

Теперь, если MySQL инсталлируется в `/usr/local`, на каталог инсталляции можно ссылаться как на `/usr/local/mysql`.

Для того чтобы программа-клиент MySQL запускалась из командной строки без ввода полного пути, достаточно задать в переменной среды `PATH` путь к каталогу `bin`, расположенному в инсталляционном каталоге MySQL. Путь задается в одном из стартовых файлов оболочки.

Если планировалось установить только клиентскую поддержку дистрибутива без программного обеспечения сервера, работу можно считать выполненной. Если MySQL инсталлируется впервые, выполните инструкции из раздела "Инициализация каталога данных и таблиц разрешений". При обновлении уже существующей инсталляции перейдите к разделу "Запуск сервера".

Инсталляция RPM-дистрибутива

RPM-файлы используются для инсталляции MySQL в системах Linux. Они имеют следующие названия.

- MySQL-client.*version-platform*.rpm. Программы клиентов.
- MySQL-*version-platform*.rpm. Программное обеспечение сервера.
- MySQL-Max-*version-platform*.rpm. Дополнительное программное обеспечение сервера с дополнительными возможностями. Перед его установкой необходимо установить соответствующий файл MySQL-*version-platform*.rpm.
- MySQL-embedded-*version-platform*.rpm. Встроенный сервер, libmysqld.
- MySQL-devel.*version-platform*.rpm. Программная поддержка (клиентские библиотеки и файлы) для написания клиентских программ. Этот файл необходим, если планируется использовать или создавать сценарии Perl DBI для доступа к базам данных MySQL.
- MySQL-shared-*version-platform*.rpm. Разделяемые клиентские библиотеки.
- MySQL-bench.*version-platform*.rpm. Средства проверки производительности работы и тесты. Для их работы требуется поддержка DBI-интерфейса Perl. (Подробно об этом рассказывается в разделе "Инсталляция поддержки Perl DBI".)
- MySQL-*version*.src.rpm. Исходный код сервера, клиентов, средств проверки производительности и тестов.

Для выполнения инсталляции из RPM-файлов необязательно перемещаться в какой-либо определенный каталог. Эти файлы уже содержат специальную информацию о том, куда следует инсталлировать файлы MySQL. Чтобы определить, куда будут инсталлироваться файлы каждого RPM-файла *rpm_file*, воспользуйтесь следующей командой:

```
# rpm -qpl rpm_file
```

Для инсталляции RPM-файла запустите команду

```
# rm -i rpm_file
```

Поскольку различные компоненты MySQL распределены по нескольким файлам RPM, возможно, придется инсталлировать сразу несколько файлов. Для инсталляции поддержки клиентов используйте команду

```
# rpm -i MySQL-client-version-platform.rpm
```

Для инсталляции поддержки сервера применяется следующая команда (пропустите вторую команду, если дополнительные возможности max-сервера вам не нужны):

```
# rpm -i MySQL-version-platform.rpm
# rpm -i MySQL-Max-version-platform.rpm
```

Если планируется написание собственных программ при поддержке средств программирования клиентов, инсталлируйте и RPM-файл разработки:

```
# rpm -i MySQL-devel-version-platform.rpm
```

Чтобы выполнить инсталляцию из исходного RPM-файла, достаточно запустить следующую команду:

```
% rpm --recompile MySQL-version.src.rpm
```

Для того чтобы программа-клиент MySQL запускалась из командной строки без ввода полного пути, в переменной среды PATH достаточно задать путь к каталогу bin, расположенному в инсталляционном каталоге MySQL. Путь задается в одном из стартовых файлов оболочки.

Если планировалось установить только клиентскую поддержку дистрибутива без программного обеспечения сервера, работу можно считать выполненной. Если MySQL инсталлируется впервые, выполните инструкции из раздела "Инициализация каталога данных и таблиц разрешений". При обновлении уже существующей инсталляции перейдите к разделу "Запуск сервера".

Инсталляция дистрибутива в исходных текстах

Файлы дистрибутива, представленные в виде исходного текста, имеют, как правило, название `mysql-`*`version`*`.tar.gz`, где *version* — номер версии MySQL. Перед запуском команд инсталляции сначала переместитесь в каталог, в который необходимо распаковать дистрибутив. Загрузите файл дистрибутива и распакуйте его с помощью одной из двух следующих команд (используйте вторую команду, если рабочая версия tar не распознает параметр z при распаковке дистрибутива):

```
% tar zxf mysql-version.tar.gz
% gunzip < mysql-version.tar.gz | tar xf -
```

Распаковка дистрибутива приведет к созданию каталога `mysql-`*`version`*, в котором и будет размещено содержимое дистрибутива. Перейдите в этот каталог:

```
% cd mysql-version
```

Прежде чем приступить к инсталляции дистрибутива, необходимо его сконфигурировать и откомпилировать. Если эти операции не удается выполнить, обратитесь к соответствующей главе *Справочного руководства по MySQL* и поищите рекомендации относительно используемой системы и компьютера.

Для настройки конфигурации дистрибутива воспользуйтесь командой configure:

```
% ./configure
```

Возможно, имеет смысл определить параметры настройки. Чтобы получить список возможных параметров, запустите следующую команду:

```
% ./configure --help
```

Ниже приведено краткое описание наиболее полезных параметров.

- `--with-innodb`, `--without-innodb`. Включить или отключить поддержку таблиц типа InnoDB. До версии MySQL 4 для включения поддержки таблиц типа InnoDB использовался параметр `--with-innodb`. Начиная с MySQL 4 таблицы типа InnoDB поддерживаются по умолчанию, для отключения их поддержки необходимо указать параметр `--without-innodb`.
- `--with-berkeley-db`. Поддержка таблиц типа BDB.
- `--without-server`. Настройка для создания клиентской поддержки (клиентских программ или библиотек). Этот параметр можно использовать, если планируется обеспечить доступ к серверу, работающему на другом компьютере.

- `--with-embedded-server`. Создать библиотеку встроенного сервера, `libmysqld`.
- `--prefix=path_name`. По умолчанию в качестве основного каталога инсталляции задается каталог `/usr/local`. Каталог данных, каталог клиентов, каталог сервера, клиентских библиотек и файлы заголовков, страницы описания и языковые файлы инсталлируются соответственно в подкаталоги `var`, `bin`, `libexec`, `lib`, `include`, `man` и `share` этого каталога. Параметр `--prefix` используется для изменения расположения каталога инсталляции. Например, для того, чтобы установить все в каталог `/usr/local/mysql`, необходимо использовать параметр `--prefix=/usr/local/mysql`.
- `--localstatedir=path_name`. Этот параметр изменяет месторасположение каталога данных. Его могут с успехом применять администраторы, которые не намерены хранить свои базы данных в каталоге `/usr/local/var`.
- `--with-low-memory`. Для компиляции исходного файла `sql/sql_yacc.cc` требуется большой объем памяти, что иногда не позволяет успешно выполнить инсталляцию. К основным признакам этой проблемы относится появление ошибки "`Fatal Signal 11`" или переполнение виртуальной памяти. Параметр `--with-low-memory` указывает компилятору использовать меньший объем памяти.

По завершении работы утилиты `configure` откомпилируйте дистрибутив и инсталлируйте его:

```
% make
% make install
```

Если администратор не определял ранее с помощью параметра `--prefix` новый каталог инсталляции, в котором бы он обладал правами записи, то, возможно, для запуска команды инсталляции придется зарегистрироваться в качестве пользователя `root`.

Для того чтобы программа-клиент MySQL запускалась из командной строки без ввода полного пути, достаточно задать в переменной среды `PATH` путь к каталогу `bin`, расположенному в инсталляционном каталоге MySQL. Путь задается в одном из стартовых файлов оболочки.

Если планировалось установить только клиентскую поддержку дистрибутива без программного обеспечения сервера, работу можно считать выполненной. Если MySQL инсталлируется впервые, выполните инструкции из раздела "Инициализация каталога данных и таблиц разрешений". При обновлении уже существующей инсталляции перейдите к разделу "Запуск сервера".

Инициализация каталога данных и таблиц разрешений

Перед началом работы с инсталляцией MySQL необходимо обязательно проинициализировать базу данных `mysql`, которая содержит таблицы разрешений, управляющие доступом к серверу. Эту процедуру необходимо выполнять только для новой инсталляции и только при запущенном сервере. При инсталляции с исключительно клиентской поддержкой можно ее пропустить.

Итак, предположим, что *DATADIR* — это путь и имя каталога данных. Для запуска приведенных ниже команд необходимо зарегистрироваться в качестве пользователя root. Если же администратор зарегистрировался под именем mysqladm и выполнил установку MySQL под своей учетной записью, для запуска команд необязательно перерегистрироваться, а можно пропустить выполнение команд chown и chmod.

Для инициализации каталога данных, базы данных mysql и таблиц разрешения по умолчанию поменяйте размещение на инсталляционный каталог MySQL и запустите сценарий mysql_install_db. (Это не надо делать при установке RPM-файлов, так как в этом случае mysql_install_db запускается автоматически.)

Например, при установке СУБД MySQL в каталог /usr/local/mysql для этого потребуются следующие команды:

```
# cd /usr/local/mysql
# ./bin/mysql_install_db
```

Если этот сценарий работает неправильно, обратитесь к соответствующей главе *Справочного руководства по MySQL* и попробуйте поискать какую-либо информацию о возникшей проблеме. При этом не забывайте, если сценарий mysql_install_db не завершает успешно свою работу, то и таблицы разрешений будут созданы неверно. Поэтому их следует удалить перед следующим запуском сценария mysql_install_db, иначе новые созданы не будут. Для этого выполните следующую команду:

```
# rm -rf DATADIR/mysql
```

По завершении работы сервера измените права владения и режимы для всех файлов каталога данных. Если пользователем и группой являются пользователь mysqladm и группа mysqlgrp, команды будут иметь следующий вид:

```
# chown -R mysqladm.mysqlgrp DATADIR
# chmod -R go-rwx DATADIR
```

Команда chown присваивает права владения пользователю MySQL, а команда chmod изменяет режим каталога данных таким образом, что получить доступ к нему может только пользователь mysqladm.

Запуск сервера

Описанные в этом разделе действия необходимо выполнять только администраторам, которые собираются устанавливать сервер. Владельцы исключительно клиентских инсталляций могут пропустить материал этого раздела. Запускайте представленные далее команды из тех же каталогов, что и в предыдущем разделе. Как правило, для запуска приведенных ниже команд необходимо зарегистрироваться в качестве пользователя root. Если же администратор зарегистрировался под именем mysqladm и выполнил установку MySQL под своей учетной записью, для запуска команд необязательно перерегистрироваться и следует пропустить параметр --user.

Для запуска сервера перейдите в инсталляционный каталог MySQL (например, /usr/local/mysql) и выполните следующую команду:

```
# cd /usr/local/mysql
# bin/mysqld_safe --user=mysqladm &
```

Параметр `--user` указывает серверу запуститься с правами пользователя `mysqladm`. (Не забывайте, что до СУБД MySQL 4 `mysqld_safe` назывался `safe_mysqld`.)

В этом месте одновременно можно запустить и другие действия.

- Инсталляция по умолчанию позволяет пользователю MySQL `root` подключаться без пароля. Для обеспечения безопасности пароль лучше установить.
- Необходимо настроить сервер для автоматического запуска и остановки как части обычных процедур запуска и отключения.
- В конфигурационном файле можно задать параметр `--user`, чтобы не задавать его при каждом запуске сервера.
- Должны быть включены различные типы регистрации. Они полезны для управления сервером и восстановления данных.
- Для сервера, поддерживающего таблицы InnoDB, необходимо настроить обработку таблиц InnoDB.

Инструкции по выполнению этих действий приведены в приложении B, "Операторы и функции".

Инсталляция поддержки Perl DBI в ОС UNIX

Программное обеспечение DBI необходимо для доступа к базам данных MySQL с помощью сценариев Perl. Потребуется установить модуль DBI, который обеспечивает работу общего драйвера DBI, и модуль DBD::mysql, который обеспечивает работу драйвера, работающего с СУБД MySQL. Для модуля DBI необходимо также инсталлировать поддержку программирования клиентов MySQL, поскольку этот модуль использует клиентскую библиотеку C. Для написания Web-сценариев DBI можно установить модуль CGI.pm. Программное обеспечение DBI требует также наличия относительно новой версии Perl (5.005_003 и выше). Если этот пакет не установлен, посетите узел `http://www.perl.com/`, возьмите дистрибутив Perl и инсталлируйте его перед установкой поддержки DBI. Кроме того, при необходимости создавать сценарии DBI, опирающиеся на Web-технологии, следует установить модуль CGI.pm.

Чтобы убедиться в том, что заданный модуль Perl уже установлен, воспользуйтесь командой `perldoc`; в том случае, если модуль установлен, команда `perldoc` отобразит его документацию:

```
% perldoc DBI
% perldoc DBD::mysql
% perldoc CGI
```

Простейшим способом установки модулей Perl в ОС UNIX является использование оболочки CPAN. Зарегистрировавшись как пользователь `root`, выполните следующие команды:

```
# perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
cpan> install CGI
```

Кроме того, дистрибутив с исходными текстами можно загрузить с Web-узла `cpan.perl.org` в виде сжатых `tar`-файлов. Распакуйте файл дистрибутива

dist_file.tar.gz с помощью следующих команд (если рабочая версия `tar` не распознает параметр z при распаковке дистрибутива, используйте вторую команду):

```
% tar zxf dist_file.tar.gz
% gunzip < dist_file.tar.gz | tar xf -
```

Затем перейдите в каталог дистрибутива, созданный программой `tar`, и запустите следующие команды (для этого, возможно, придется зарегистрироваться в качестве пользователя `root`):

```
% perl Makefile.PL
% make
% make test
# make install
```

Независимо от выбранного метода инсталляции, Perl задаст вам ряд вопросов при установке модуля DBD::mysql.

- **Какие драйверы необходимо инсталлировать?** (Which drivers do you want to install?) Существует несколько вариантов для различных комбинаций MySQL и mSQL. Если mSQL запускать не планируется, выберите MySQL для упрощения дальнейшей работы.
- **Хотите ли вы инсталлировать эмуляцию MysqlPerl?** (Do you want to install the MysqlPerl emulation?) MysqlPerl – это старый интерфейс Perl для MySQL, который сейчас уже не нужен. Ответьте отрицательно, если, конечно, не собираетесь использовать старые сценарии MysqlPerl и активизировать для них поддержку эмуляции в модуле DBI.
- **Где расположена инсталляция MySQL?** (Where is your MySQL installed?) Необходимо указать родительский каталог подкаталога с файлами заголовков MySQL. Если место инсталляции не изменялось, то наверняка все файлы MySQL размещаются в каталоге `/usr/local` или `/usr/local/mysql`.
- **Какие базы данных следует использовать для тестирования драйверов MySQL?** (Which database should I use for testing the MySQL drivers?) В качестве такой базы данных по умолчанию выбрана база данных `test`. Эта база подходит для подобных целей, если, конечно, администратор не закрыл к ней анонимный доступ. В последнем случае необходимо определить имя базы данных, к которой имеется доступ, и определить правильное имя и пароль пользователя MySQL.
- **На каком компьютере запущена база данных?** (On which host is the database running?) Если сервер запущен на локальном компьютере, достаточно указать имя `localhost`. Если же сервер работает на другом компьютере, укажите его имя. Для успешного выполнения команды `make test` сервер MySQL на этом компьютере должен работать, иначе проверка завершена не будет.
- **Имя пользователя для подключения к базе данных? Пароль для подключения к базе данных?** (User name for connecting to the database?) (Password for connecting to the database?) Имя пользователя и пароль, используемые для подключения к проверяемому серверу MySQL. Для обоих вопросов по умолчанию устанавливается ответ `undef`, благодаря которому

драйвер подключается как анонимный пользователь. Чтобы подключиться под конкретным именем, введите собственные значения.

Если в процессе инсталляции модулей Perl возникают какие-либо проблемы, обратитесь к файлу README соответствующего дистрибутива. Ответы на большинство вопросов относительно инсталляции можно найти также в отчетах почтового списка DBI.

Инсталляция PHP и Apache

В представленных в этом разделе инструкциях предполагается, что PHP работает как DSO-модуль (dynamic shared object — динамический разделяемый объект) Web-сервера Apache (httpd). Это значит, что сначала должен быть установлен Apache, а только после этого создан и инсталлирован PHP. Если Web-сервер Apache в системе еще не установлен, можно установить двоичный дистрибутив, имеющий поддержку DSO-объектов, или откомпилировать исходный дистрибутив, который бы включал поддержку DSO-объектов. После того как Web-сервер Apache установлен, настройте ваш дистрибутив PHP с помощью следующих команд:

```
% ./configure --with-mysql --with-apxs
% ./configure --with-mysql --with-apxs=/path/to/apxs
```

Параметр --with-mysql отражает, что вы хотите построить PHP с поддержкой клиента MySQL. Этот параметр должен включаться для обеспечения правильной работы сценариев PHP, обеспечивающих работу с СУБД MySQL. Если первая команда не работает, потому что configure не "знает", где находится сценарий apxs, воспользуйтесь второй командой, которая задает его местоположение явным образом. (Сценарий apxs, или вспомогательное средство Web-сервера, представляет собой вспомогательный сценарий, который передает информацию о конфигурации Web-сервера Apache другим модулям.) После настройки PHP создайте его и установите следующим образом (для выполнения инсталляционных команд может потребоваться зарегистрироваться в качестве пользователя root):

```
% make
# make install
# cp php.ini-dist /usr/local/lib/php.ini
```

Команда cp устанавливает основной файл инициализации PHP там, где PHP может его найти. По необходимости можно заменить php.ini-recommended на php.ini-dist; посмотрите на них и выберите более предпочтительный для себя вариант. После инсталляции PHP отредактируйте конфигурационный файл Apache, httpd.conf. Необходимо дать команду Web-серверу Apache загрузить модуль PHP при запуске и распознавать сценарии PHP.

Для того чтобы Web-сервер Apache знал о загрузке модуля PHP, в соответствующих разделах конфигурационного файла httpd.conf должны иметься директивы LoadModule и AddModule (посмотрите на аналогичные директивы конфигурационного файла). Эти директивы могут добавляться во время выполнения процедуры инсталляции, и, если этого не было сделано, они должны иметь следующий вид:

```
LoadModule php4_module libexec/libphp4.so
AddModule mod_php4.c
```

Конфигурационный файл Apache `httpd.conf` следует также отредактировать, чтобы модули Apache распознавали сценарии PHP. Это распознавание основывается на определении расширений имен файлов, используемых для сценариев PHP.

Эти команды изменяют конфигурацию Apache таким образом, чтобы дать возможность обнаружить файл модуля PHP, после чего выполняется собственно построение и инсталляция. Следует также отредактировать файл конфигурации Apache `httpd.conf`, чтобы модули Apache распознавали сценарии PHP. Это распознавание основывается на сравнении расширений имен файлов, используемых для сценариев PHP.

Расширение `.php` распространено наиболее широко и используется в примерах, приведенных в этой книге. Другим популярным расширением является `.phtml`. В конце концов, можно задать распознавание сразу всех этих расширений. Это будет наилучшим вариантом, если в будущем планируется инсталляция страниц, получаемых от других пользователей. Ведь нельзя наверняка сказать, какие расширения они будут использовать. Так, например, чтобы Apache смог распознавать файлы расширений `.php` и `.phtml`, добавьте в конфигурационный файл следующие строки:

```
AddType application/x-httpd-php3 .php
AddType application/x-httpd-php3 .phtml
```

Можно также задать, чтобы Apache по умолчанию считывал файл `index.php` или `index.phtml`, если в конце URL-адреса не указывается конкретное имя файла. Для этого отыщите в файле конфигурации строку

```
DirectoryIndex index.html
```

и измените ее на

```
DirectoryIndex index.php index.phtml index.html
```

Завершив редактирование конфигурационного файла Apache, остановите работу сервера `httpd`, если он еще работает, и перезапустите его. В большинстве систем это можно осуществить с помощью следующих команд (запущенных от имени пользователя `root`):

```
# /usr/local/apache/bin/apachectl stop
# /usr/local/apache/bin/apachectl start
```

Необходимо также настроить автоматический запуск и остановку Apache в моменты загрузки и завершения работы системы. Как это сделать, рассказывается в документации по Apache. Для этого в момент загрузки необходимо запустить команду `apachectl start`, а перед завершением работы — `apachectl stop`.

Если в процессе установки возникают какие-либо проблемы, обратитесь к разделу "VERBOSE INSTALL" файла `INSTALL`, входящего в состав дистрибутива PHP. (Нелишним будет также прочитать весь этот файл, содержащий много полезной информации.)

Инсталляция MySQL в Windows

СУБД MySQL может работать под управлением ОС, опирающихся на NT-технологии (Windows NT, 2000 и XP), а также на ОС, к числу которых можно отнести

Windows 95, Windows 98 и Windows Me. Для успешного запуска MySQL необходимо инсталлировать поддержку протокола TCP/IP. На ОС Windows NT 4 убедитесь в том, что в вашей системе используется Service pack 3 или выше. На Windows 95 должно использоваться программное обеспечение Winsock версии 2 и выше.

По возможности рекомендуется использовать NT-версию ОС Windows. В этом случае появляется возможность запуска MySQL в качестве службы, которую Windows NT запускает и останавливает автоматически во время запуска и остановки системы. Кроме того, NT-серверы также позволяют осуществлять подключение клиентов по именованным каналам.

Кроме MySQL-сервера и клиентских приложений, также можно установить драйвер MyODBC — это драйвер MySQL, соответствующий стандарту ODBC (Open Database Connectivity), разработанному Microsoft. Драйвер MyODBC позволяет совместимым с ODBC программам осуществлять доступ к базам данных MySQL. Например, можно установить MyODBC, а потом для подключения к MySQL-серверу использовать такие ODBC-программы, как Microsoft Access.

Дистрибутивы СУБД MySQL для ОС Windows доступны на Web-узле MySQL в виде zip-файлов, имеющих имена `mysql-version-win.zip`. Для распаковки такого файла достаточно дважды щелкнуть на нем. Если двойной щелчок к распаковке не приводит, воспользуйтесь соответствующей программой, например Winzip, pkunzip или unzip. В результате вы получите каталог, содержащий приложение `Setup.exe`, которое необходимо запустить для того, чтобы установить СУБД MySQL. По умолчанию СУБД MySQL инсталлируется в каталоге `C:\mysql`, но можно задать и другой каталог.

После инсталляции СУБД MySQL в ОС Windows нет необходимости инициализировать каталог данных или таблицы разрешений, так как они автоматически устанавливаются в дистрибутиве. Однако при установке СУБД MySQL в любом другом месте, а не в `C:\mysql`, необходимо в конфигурационном файле задать специальную параметрическую группу `[mysqld]`, которую сервер считывает во время запуска, таким образом, он может определить, где размещаются основной установочный каталог и каталог данных. (Конфигурационным файлом может быть файл `C:\my.cnf` или `my.ini` в системном каталоге Windows.) Например, при установке MySQL в `E:\mysql` конфигурационный файл должен иметь такой вид (обратите внимание на использование в имени обычных, а не обратных косых черт):

```
[mysqld]
basedir=E:/mysql
datadir=E:/mysql/data
```

Имена путей также необходимо поменять, если на практике будут использоваться каталоги, отличные от каталогов, принимаемых по умолчанию.

Выберите один из предложенных серверов.

Сервер	Описание
`mysqld`	Стандартный сервер
`mysql-nt`	Оптимизированный сервер с поддержкой именованных каналов
`mysqld-max`	Оптимизированный сервер с поддержкой транзакций и символических ссылок
`mysqld-max-nt`	Оптимизированный сервер с поддержкой транзакций, символических ссылок и именованных каналов

Серверы, имеющие в своем имени -nt, при работе в NT-системах поддерживают работу с именованными каналами. Однако поддержка именованных каналов по умолчанию работает только до версии MySQL 3.23.50. Для того чтобы эта возможность работала после версии 3.23.50, в группу [mysqld] конфигурационного файла необходимо добавить такую строку:

```
[mysqld]
enable-named-pipe
```

В Windows NT-системах сервер mysqld-nt может быть настроен для работы как служба, которая запускается автоматически при запуске Windows (mysqld-nt можно заменить на mysqld-max-nt в следующей команде):

```
C:\> C:\mysql\bin\mysqld-nt --install
```

Задание в этой строке параметра --install-manual вместо --install означает, что сервер устанавливается как служба, запускаемая вручную.

При установке mysqld-nt для работы в качестве службы в группе [mysqld] конфигурационного файла можно задать и другие параметры.

Сервер, установленный как сервис, можно запустить вручную с помощью диспетчера служб Windows. Он должен присутствовать как элемент Services панели управления Windows или в элементе Administrative Tools панели управления (Control Panel). Эта же служба может быть запущена с помощью команды

```
C:\> net start MySQL
```

Для остановки сервера воспользуйтесь диспетчером служб или одной из следующих команд:

```
C:\> net stop MySQL
C:\> C:\mysql\bin\mysqladmin -u root shutdown
```

Для того чтобы удалить MySQL из списка служб, отключите его, а потом вызовите следующую команду:

```
C:\> C:\mysql\bin\mysqld-nt --remove
```

Во избежание взаимодействия диспетчера служб и команд DOS, диспетчер служб при работе с командами, затрагивающими службы, лучше всего закрыть.

В системах, не опирающихся на NT-технологии (или в NT-системах, когда сервер не установлен как служба), сервер необходимо запускать и останавливать из командной строки. Для запуска mysqld запустите его следующим образом (mysqld-max в этой команде можно заменить):

```
C:\> C:\mysql\bin\mysqld-max
```

По необходимости из командной строки можно задать другие команды. Для отключения сервера можно воспользоваться программой mysqladmin:

```
C:\> C:\mysql\bin\mysqladmin -u root shutdown
```

Запустить сервер в режиме консоли (при котором он отображает диагностические сообщения в окне консоли) можно следующим образом:

```
C:\> C:\mysql\bin\mysqld --console
```

В консольном режиме может быть запущен любой сервер. В этом случае в командной строке после параметра --console или в конфигурационном файле

можно задать и другие параметры. Для отключения сервера используйте программу `mysqladmin`.

При возникновении проблем при запуске сервера можно просмотреть заметки, касающиеся ОС Windows, в соответствующей главе *Справочного руководства по MySQL*, посвященной инсталляции.

Чтобы вызвать клиентские программы MySQL из командной строки, не указывая при этом их полные пути, задайте путь к каталогу `C:\mysql\bin` в переменной среды `PATH`. (Если MySQL установлен в каком-то другом каталоге, необходимо изменить имя пути соответствующим образом.) Кроме того, путь можно задать и в файле `AUTOEXEC.BAT` или (на NT-системах) использовать элемент System панели управления.

Стандартная инсталляция позволяет всем подключаться к MySQL как пользователю `root` без указания пароля, что позволяет задавать пароли в соответствии с инструкциями из приложения B, "Операторы и функции". В этом приложении также содержатся инструкции по выполнению других действий, таких как включение протоколирования и настройка дескриптора таблиц типа InnoDB.

Инсталляция поддержки Perl DBI

Самый простой способ инсталлировать эту поддержку — загрузить дистрибутив ActiveState Perl с Web-узла ActiveState (`http://www.activestate.com/`) и установить его. Затем инсталлируйте все модули Perl, необходимые для работы. Для этих целей используется программа `ppm` (Perl Package Manager). Чтобы определить, какие модули уже инсталлированы, запустите следующую команду:

```
C:\> ppm info
```

Затем инсталлируйте необходимые модули, используя соответствующие команды из приведенного ниже списка. Наверняка модуль CGI.pm уже установлен, и необходимо добавить лишь DBI-пакеты:

```
C:\> ppm
ppm> install DBI
ppm> install DBD-mysql
ppm> install CGI
```

Инсталляция PHP и Apache в ОС Windows

Web-сервер Apache и PHP можно получить как исполняемые модули на Web-узлах Apache и PHP, адреса которых были указаны в самом начале этого приложения. Под управлением Apache 1.3.x PHP может работать только как автономная CGI-программа. Под управлением Apache 2.x PHP может работать как автономная программа или как модуль Apache.

Инсталляция MyODBC

Дистрибутивы MyODBC 3 запакованы в саморазворачивающихся архивах с именами типа `MyODBC-`*`version`*`.exe`. Загрузите файл дистрибутива, а затем запустите его для инсталляции. Для версий до MyODBC 3 дистрибутивы запакованы в ZIP-файлы. Есть разные дистрибутивы для NT- и не NT-систем (обозначаются

соответственно `-nt` и `-win95` в именах файлов). Для инсталляции дистрибутива ZIP-формата распакуйте его двойным щелчком, а затем запустите приложение `Setup.exe`, расположенное внутри полученного каталога. Если в процессе инсталляции MyODBC возникает ошибка "Problems while copying MFC30.DLL", это значит, что файл `MFC30.DLL` уже используется каким-то приложением. В таком случае попробуйте перезагрузить Windows в безопасном режиме и снова запустите программу `Setup`.

После инсталляции дистрибутива MyODBC драйвер необходимо настроить с помощью элемента ODBC панели управления. (Она должна быть размещена в панели управления или в элементе Administrative Tools панели управления с именем наподобие Data Sources (ODBC) или ODBC Data Sources.) После щелчка на элементе ODBC отобразится окно, в котором можно установить имя источника данных (Data Source Name — DSN). Активизируйте вкладку User DSN и щелкните на кнопке Add, чтобы открыть окно со списком доступных драйверов источников данных. Выберите в списке драйвер MySQL и щелкните на кнопке Finish. В появившемся окне введите параметры соединения для источника данных и щелкните на кнопке OK.

Так, например, для настройки источника данных для базы данных `sampdb`, использующей описанные в этой книге сервер и учетную запись, можно определить следующие параметры.

Имя поля	Значение поля
DSN name:	`sampdb-dsn`
Host Name (или IP):	`localhost`
database name:	`sampdb`
User:	`sampadm`
Password:	`secret`
Port:	`3306`

Сразу после установки этих параметров можно будет получить доступ к базам данных MySQL из программ, поддерживающих работу с ODBC. Например, одним из основных применений MyODBC является подключение к MySQL-серверу из Microsoft Access. После установки ODBC для подключения к MySQL-серверу из MS Access выполните следующее.

1. Запустите программу MS Access.
2. Откройте базу данных или создайте новую.
3. В меню File выберите Get External Data, а затем Link Tables.
4. В появившемся окне щелкните на элементе Files всплывающего меню Type, а затем на элементе ODBC Databases.
5. Выберите DSN, настроенный вами в элементе ODBC панели управления, для подключения к MySQL.
6. Выберите нужные таблицы MySQL.

После выполнения этой процедуры выбранные таблицы станут доступными через MS Access.

Б

Типы данных

В этом приложении...

Числовые типы

Строковые типы

Типы даты и времени

В этом приложении описываются все используемые в MySQL типы данных. Более детально о применении каждого типа рассказывалось в главе 1, "Знакомство с СУБД MySQL и SQL". Все описанные здесь типы могут использоваться в MySQL с версии 3.22.0 (если не указано другое). С тех пор некоторые типы претерпели существенные изменения, и это отражено в описаниях типов.

В именах типов применяются следующие условные обозначения.

- **Квадратные скобки** ([ ]). Необязательная информация.
- *M*. Максимально возможная ширина отображения. Если иное не указано, *M* представляет собой целое число в диапазоне от 1 до 255.
- *D*. Число цифр после запятой в десятичных типах. *D* может быть только целым числом в диапазоне от 0 до 30 и должно быть не больше, чем *M*-2 (иначе значение *M* автоматически увеличится до *D*+2).

Согласно терминологии ODBC, значения *M* и *D* соответствуют понятиям "точность" и "масштаб".

Для каждого рассматриваемого типа будет предоставлена следующая информация.

- **Описание.** Краткое описание типа.
- **Допустимые атрибуты.** Ключевые слова необязательных атрибутов, которые могут связываться с типом столбца в операторах `CREATE TABLE` и `ALTER TABLE`. Атрибуты в этом приложении указываются в алфавитном

порядке, который никак не отражает порядок применения согласно синтаксису операторов CREATE TABLE и ALTER TABLE. Их синтаксис представлен в приложении Г, "Синтаксис SQL". Помимо указанных в описании каждого типа дополнительных атрибутов, могут также использоваться и приведенные ниже глобальные атрибуты.

- **Допустимая длина.** Максимально возможная длина значений столбца для строковых типов.
- **Диапазон.** Диапазон значений может указываться для числовых типов, а также для типов даты и времени. Для целочисленных типов указываются два диапазона, поскольку значения столбцов могут быть как со знаком, так и без него. Диапазоны в таких случаях будут другими.
- **Нулевое значение.** Для типов даты и времени нулевым является значение, которое сохраняется системой при вводе недопустимого значения.
- **Значение по умолчанию.** Значение, принимаемое по умолчанию, если в описании типа атрибут DEFAULT не определяется явным образом.
- **Объем.** Число байт, необходимых для хранения значений данного типа. Для некоторых типов этот параметр строго фиксирован, для остальных изменяется в зависимости от длины хранимого значения.
- **Сравнение.** Определяет, учитывается ли регистр символов в значениях данного типа. Эта информация используется при сортировке и индексации, поскольку эти операции основаны на сравнении.
- **Синонимы.** Все синонимы имени типа.
- **Замечание.** Отдельные замечания относительно типа.

Некоторые глобальные атрибуты применяются ко всем или почти всем типам данных, поэтому во избежание многочисленных повторений они приводятся только здесь.

- Для всех типов можно задавать атрибуты NULL или NOT NULL.
- Для большинства типов можно определять атрибут DEFAULT *default_value*. Значения по умолчанию задаются константами. Например, нельзя задавать значение по умолчанию выражением DEFAULT NOW() для столбца типа DATETIME. Значение по умолчанию неэффективно для столбцов типа TIMESTAMP, столбцов, объявленных с атрибутом AUTO_INCREMENT, и не применяется для столбцов BLOB и TEXT. (Добавление возможности задания динамических значений по умолчанию и для столбцов типа BLOB и TEXT запланированы в версии 4.1, но на момент написания этой книги они отсутствовали.)

Числовые типы

СУБД MySQL имеет числовые типы как для целых, так и для десятичных значений. Их выбор зависит в основном от диапазона значений, который необходимо представить данным типом. При определении атрибута ZEROFILL числовые значения дополняются нулями для достижения установленной ширины отображения столбца.

Если для целочисленных типов определен атрибут AUTO_INCREMENT, столбец должен иметь индекс PRIMARY KEY или UNIQUE. Вставка значения NULL в столбец AUTO_INCREMENT приводит к получению значения, на единицу большего максимального текущего значения столбца. Более подробная информация о поведении столбцов, объявленных с атрибутом AUTO_INCREMENT, приводится в приложении В, "Операторы и функции".

Определение атрибута UNSIGNED запрещает ввод для числовых типов отрицательных значений. Однако до версии MySQL 4.0.2 для столбцов типа FLOAT или DOUBLE атрибут UNSIGNED не поддерживается. (Для столбцов такого типа атрибут UNSIGNED может задаваться, но он все равно не будет обеспечивать несохранение отрицательных значений, что может привести к неправильному отображению отрицательных значений при объявлении их с атрибутом ZEROFILL.)

В некоторых случаях объявление с одним атрибутом приводит к активизации других атрибутов. Так, объявление атрибута ZEROFILL для численных типов вызывает объявление такого столбца, как UNSIGNED. С версии MySQL 3.23 определение атрибута AUTO_INCREMENT автоматически вызывает объявление столбца как NOT NULL.

Запомните, что операторы DESCRIBE и SHOW COLUMNS отображают столбцы с атрибутом AUTO_INCREMENT как NULL, хотя литеральное значение NULL присвоить такому столбцу невозможно. Это отражает тот факт, что при добавлении новой записи присвоением значения NULL значение столбца присваивается по умолчанию (следующий порядковый номер).

TINYINT [(*M*)]

Описание: очень малое целое.

Допустимые атрибуты: AUTO_INCREMENT, UNSIGNED, ZEROFILL.

Диапазон: от -128 до 127 (от -2^7 до 2^7-1) или от 0 до 255 (от 0 до 2^8-1), если UNSIGNED.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; 0 — если NOT NULL.

Объем: 1 байт.

Синонимы: типы INT1[(*M*)], BIT и BOOL являются синонимами TINYINT(1).

SMALLINT [(*M*)]

Описание: малое целое.

Допустимые атрибуты: AUTO_INCREMENT, UNSIGNED, ZEROFILL.

Диапазон: от -32768 до 32767 (от -2^{15} до $2^{15}-1$) или от 0 до 65535 (от 0 до $2^{16}-1$), если UNSIGNED.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; 0 — если NOT NULL.

Объем: 2 байта.

Синоним: INT2 [(*M*)].

MEDIUMINT [(*M*)]

Описание: целое среднего размера.

Допустимые атрибуты: AUTO_INCREMENT, UNSIGNED, ZEROFILL.

Диапазон: от -8388608 до 8388607 (от -2^{23} до 2^{23}-1) или от 0 до 16777215 (от 0 до 2^{24}-1), если UNSIGNED.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; 0 — если NOT NULL.

Объем: 3 байта.

Синонимы: INT3 [(*M*)] и MIDDLEINT [(*M*)].

INT [(*M*)]

Описание: целое нормального размера.

Допустимые атрибуты: AUTO_INCREMENT, UNSIGNED, ZEROFILL.

Диапазон: от -2147483648 до 2147483647 (от -2^{31} до 2^{31}-1) или от 0 до 4294967295 (от 0 до 2^{32}-1), если UNSIGNED.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; 0 — если NOT NULL.

Объем: 4 байта.

Синонимы: INTEGER [(*M*)] и INT4 [(*M*)].

BIGINT [(*M*)]

Описание: большое целое.

Допустимые атрибуты: AUTO_INCREMENT, UNSIGNED, ZEROFILL.

Диапазон: от -9223372036854775808 до 9223372036854775807 (от -2^{63} до 2^{63}-1) или от 0 до 18446744073709551615 (от 0 до 2^{64}-1), если UNSIGNED.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; 0 — если NOT NULL.

Объем: 8 байт.

Синоним: INT8 [(*M*)].

FLOAT [(*p*)]

Описание: число с плавающей запятой; при минимальном количестве требующихся битов точности, заданных значением *p*. Для значений *p* в диапазоне от 0 до 24 столбец такого типа считается значением одинарной точности, что равносильно типу FLOAT с атрибутами *M* и *D*. Для значений *p* в диапазоне от 25 до 53 столбец такого типа равносилен столбцу двойной точности, что равносильно типу DOUBLE с атрибутами *M* и *D*.

Допустимые атрибуты: UNSIGNED (начиная с MySQL 4.0.2), ZEROFILL.

Диапазон: см. описание типов FLOAT и DOUBLE.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; 0 — если NOT NULL.

Объем: 4 байта при одинарной точности и 8 байт при двойной.

Замечание: до версии MySQL 3.23.6 допустимыми значениями *p* были 4 и 8. В версиях MySQL 3.23.0–3.23.5 FLOAT(4) и FLOAT(8) рассматриваются как зна-

чения с одинарной и двойной точностью, сохраненные с полной точностью, которую позволяет данная аппаратная платформа.

До MySQL 3.23 `FLOAT(4)` и `FLOAT(8)` равносильны `FLOAT(10,2)` и `DOUBLE(16,4)` со значениями, округленными до 2 и 4 знаков после запятой, сохраненных с полной точностью, которую позволяет используемая аппаратная платформа.

FLOAT [(*M*, *D*)]

Описание: малое число с плавающей запятой; одинарная точность (менее точный тип, чем DOUBLE). Значением *M* задается ширина отображения, *D* — минимальное необходимое количество знаков после запятой. Если *D* равен 0, значения не имеют десятичной точки или дробной части. При отсутствии *M* и *D* ширина отображения и количество знаков после запятой не заданы.

Допустимые атрибуты: UNSIGNED (начиная с MySQL 4.0.2), ZEROFILL.

Диапазон: минимальные ненулевые значения — ±1,175494351E-38; максимальные ненулевые значения — ±3,402823466E+38. Отрицательные значения при объявлении атрибута UNSIGNED недопустимы.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; 0 — если NOT NULL.

Объем: 4 байта.

Синонимы: в версиях, предшествующих MySQL 3.23.6, FLOAT и FLOAT4 — синонимы `FLOAT(10,2)`.

Замечание: до версии MySQL 3.23.6 значения столбцов FLOAT (*M*, *D*) округляются до *D* знаков после запятой, а не хранятся с полной точностью, которую позволяет аппаратная платформа.

DOUBLE [(*M*, *D*)]

Описание: большое число с плавающей запятой; двойная точность (более точный тип, чем FLOAT). Значением *M* задается ширина отображения, *D* — минимальное необходимое количество знаков после запятой. Если *D* равно 0, значения не имеют десятичной точки или дробной части. При отсутствии *M* и *D* ширина отображения и количество знаков после запятой не заданы.

Допустимые атрибуты: UNSIGNED (начиная с MySQL 4.0.2), ZEROFILL.

Диапазон: минимальные ненулевые значения — ±2,2250738585072014E-308; максимальные ненулевые значения — ±1,7976931348623157E+308. Отрицательные значения недопустимы при объявлении атрибута UNSIGNED.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; 0 — если NOT NULL.

Объем: 8 байт.

Синоним: DOUBLE PRECISION [(*M*, *D*)] и REAL [(*M*, *D*)] — синонимы для DOUBLE [(*M*, *D*)]. В версиях, предшествующих MySQL 3.23.6, FLOAT8 и DOUBLE являются значениями для `DOUBLE(16,4)`.

Замечание: до версии MySQL 3.23.6 значения столбцов DOUBLE (*M*, *D*) округляются до *D* знаков после запятой, а не хранятся с полной точностью, которую позволяет аппаратное обеспечение.

DECIMAL [(*M*, *D*)]

Описание: число с плавающей запятой, хранимое как строка (1 байт на цифру, десятичную точку или знак "минус"). Значением *M* задается ширина отображения, *D* — минимальное необходимое количество знаков после запятой. Если значение *D* равно 0, значения не имеют десятичной точки или дробной части. По умолчанию значения *M* и *D* равны 10 и 0. (До версии MySQL 3.23.6 было необходимо указывать значения *M* и *D*.)

Допустимые атрибуты: UNSIGNED (начиная с MySQL 4.0.2), ZEROFILL.

Диапазон: максимальный диапазон этого типа соответствует диапазону типа DOUBLE. Эффективный диапазон для типа DECIMAL определяется значениями *M* и *D*.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; 0 — если NOT NULL.

Объем: *M* байт в версиях, предшествующих MySQL 3.23; *M+2* байта в MySQL 3.23 и выше. Если столбец задан с атрибутом UNSIGNED, символ знака не сохраняется, что уменьшает количество хранимых символов на один байт. До версии 3.23 тип DECIMAL требовал *M* байт. Символ знака и символы после запятой тоже входили в эти *M* байт, выделенные для значений столбца. Это значит, что тип DECIMAL (*M*, *D*) с версии MySQL 3.23 имеет больший диапазон, поэтому для его хранения требуется больше памяти.

Синоним: NUMERIC (*M*, *D*) и DEC [(*M*, [*D*])].

Замечание: в соответствии со стандартом ANSI SQL, начиная с версии MySQL 3.23, значение *M* не включает байты, необходимые для хранения символа знака или десятичной точки.

Строковые типы

Строковые типы MySQL наиболее часто используются для хранения текста, хотя по сути являются общими и могут содержать и произвольные данные. Пользователям MySQL доступны типы, предназначенные для хранения значений переменной длины. Выбор их определяется тем, желает ли пользователь, чтобы в записанных значениях учитывался регистр символов.

С версии MySQL 4.1 можно задавать кодировки для отдельных столбцов для типов CHAR, VARCHAR и TEXT. Это можно сделать с применением синтаксиса CHARACTER SET *charset*, где *charset* является идентификатором набора символов наподобие latin1, greek или utf8. Допустимый набор символов, поддерживаемый сервером, можно объявить с помощью оператора SHOW CHARACTER SET. Заметим, что задание кодировки для столбцов типа CHAR или VARCHAR исключает применение атрибута BINARY, применимого к этим типам.

CHAR (*M*)

Описание: строка фиксированной длины от 0 до *M* символов. Аргумент *M* должен быть целочисленным значением от 1 до 255 (в версиях до MySQL 3.23) или от 0 до 255 (в версиях MySQL 3.23 и выше). Если *M* не задана, по умолчанию присваивается 1. Строки, длина которых составляет меньше заданного значения *M*, при записи справа дополняются пробелами. Если длина строки превышает *M*

символов, лишние символы при записи удаляются. Замыкающие пробелы при считывании строки удаляются.

Допустимые атрибуты: BINARY, CHARACTER SET (начиная с MySQL 4.1).

Допустимая длина: от 0 до *M* байт.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; " " (пустая строка) – если NOT NULL.

Объем: *M* байт.

Сравнение: регистр символов не учитывается (при указании атрибута BINARY учитывается).

Синонимы: тип CHAR без аргументов является синонимом CHAR(1). Тип BINARY(*M*) является синонимом CHAR(*M*) BINARY. Начиная с MySQL 3.23.5 NCHAR(*M*) и NATIONAL CHAR(*M*) являются синонимами CHAR(*M*).

VARCHAR(*M*)

Описание: строка переменной длины от 0 до *M* символов. Аргумент *M* должен быть целочисленным значением от 1 до 255 (в версиях до MySQL 4.0.2) и от 0 до 255 (в версиях MySQL 3.23 и выше). Завершающие пробелы при считывании строки удаляются. Если длина строки превышает *M* символов, лишние символы при записи удаляются. (Удаление замыкающих пробелов должно стать необязательным в MySQL 4.1, но на момент написания этой книги этого еще не было.)

Допустимые атрибуты: BINARY, CHARACTER SET (начиная с MySQL 4.1).

Допустимая длина: от 0 до *M* байт.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; " " – если NOT NULL.

Объем: длина значения плюс 1 байт для записи длины.

Сравнение: регистр символов не учитывается (при указании атрибута BINARY учитывается).

Синонимы: CHAR VARYING(*M*). Начиная с MySQL 3.23.5 NCHAR(*M*) и NATIONAL CHAR(*M*) являются синонимами VARCHAR(*M*).

TINYBLOB

Описание: малое значение BLOB.

Допустимые атрибуты: только глобальные атрибуты.

Допустимая длина: от 0 до 255 (от 0 до 2^8-1) байт.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; " " (пустая строка) – если NOT NULL.

Объем: длина значения плюс 1 байт для записи длины.

Сравнение: регистр символов учитывается.

BLOB

Описание: значение BLOB нормального размера.

Допустимые атрибуты: только глобальные атрибуты.

Допустимая длина: от 0 до 65535 (от 0 до 2^{16}-1) байт.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; " " (пустая строка) – если NOT NULL.

Объем: длина значения плюс 2 байта для записи длины.
Сравнение: регистр символов учитывается.

MEDIUMBLOB

Описание: значение BLOB среднего размера.
Допустимые атрибуты: только глобальные атрибуты.
Допустимая длина: от 0 до 16777215 (от 0 до 2^{24}-1) байт.
Значение по умолчанию: NULL, если столбец может иметь значение NULL; " " (пустая строка) — если NOT NULL.
Объем: длина значения плюс 3 байта для записи длины.
Сравнение: регистр символов учитывается.
Синонимы: LONG VARBINARY.

LONGBLOB

Описание: большое значение BLOB.
Допустимые атрибуты: только глобальные атрибуты.
Допустимая длина: от 0 до 4294967295 (от 0 до 2^{32}-1) байт.
Значение по умолчанию: NULL, если столбец может иметь значение NULL; " " (пустая строка) — если NOT NULL.
Объем: длина значения плюс 4 байта для записи длины.
Сравнение: регистр символов учитывается.

TINYTEXT

Описание: малое значение TEXT.
Допустимые атрибуты: CHARACTER SET (начиная с MySQL 4.1).
Допустимая длина: от 0 до 255 (от 0 до 2^{8}-1) байт.
Значение по умолчанию: NULL, если столбец может иметь значение NULL; " " (пустая строка) — если NOT NULL.
Объем: длина значения плюс 1 байт для записи длины.
Сравнение: регистр символов не учитывается.

TEXT

Описание: значение TEXT нормального размера.
Допустимые атрибуты: CHARACTER SET (начиная с MySQL 4.1).
Допустимая длина: от 0 до 65535 (от 0 до 2^{16}-1) байт.
Значение по умолчанию: NULL, если столбец может иметь значение NULL; " " (пустая строка) — если NOT NULL.
Объем: длина значения плюс 2 байта для записи длины.
Сравнение: регистр символов не учитывается.

MEDIUMTEXT

Описание: значение TEXT среднего размера.
Допустимые атрибуты: CHARACTER SET (начиная с MySQL 4.1).
Допустимая длина: от 0 до 16777215 (от 0 до 2^{24}-1) байт.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; "" (пустая строка) – если NOT NULL.

Объем: длина значения плюс 3 байта для записи длины.

Сравнение: регистр символов не учитывается.

Синоним: LONG VARCHAR (*M*).

LONGTEXT

Описание: большое значение TEXT.

Допустимые атрибуты: CHARACTER SET (начиная с MySQL 4.1).

Допустимая длина: от 0 до 4294967295 (от 0 до 2^{32}-1) байт.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; "" (пустая строка) – если NOT NULL.

Объем: длина значения плюс 4 байта для записи длины.

Сравнение: регистр символов не учитывается.

ENUM("value1","value2",...)

Описание: перечень; значениям столбцов может присваиваться одно из значений списка.

Допустимые атрибуты: только глобальные атрибуты.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; первое значение списка – если NOT NULL.

Объем: 1 байт для перечня с числом членов от 1 до 255, 2 байта для перечня с числом членов от 256 до 65535.

Сравнение: регистр символов не учитывается (учитывался в версиях до MySQL 3.22.1).

SET("value1","value2",...)

Описание: набор; значениям столбцов может присваиваться одно или больше значений списка или не присваивается ни одного.

Допустимые атрибуты: только глобальные атрибуты.

Значение по умолчанию: NULL, если столбец может иметь значение NULL; "" (пустой набор) – если NOT NULL.

Объем: 1 байт (для наборов с числом членов от 1 до 8), 2 байта (от 9 до 16 членов), 3 байта (от 17 до 24 членов), 4 байта (от 25 до 32 членов) или 8 байт (от 33 до 64 членов).

Сравнение: регистр символов не учитывается (учитывался в версиях до MySQL 3.22.1).

Типы даты и времени

Для представления временных данных в различных форматах в MySQL используются типы даты и времени. Пользователям доступны как типы, предназначенные для отдельного хранения дат или времени, так и их комбинации. Существует даже специальный тип временной метки, который автоматически обновляется при записи изменений. Если нужно выводить полную дату, можно воспользоваться типом года.

Обозначениями *CC*, *YY*, *MM* и *DD* в формате даты представлены столетие, год, месяц и день. Обозначениями *hh*, *mm* и *ss* в форматах представлены часы, минуты и секунды.

DATE

Описание: дата, представленная в формате "*CCYY-MM-DD*".

Допустимые атрибуты: только глобальные атрибуты.

Диапазон: от "`1000-01-01`" до "`9999-12-31`".

Нулевое значение: "`0000-00-00`".

Значение по умолчанию: NULL, если столбец может иметь значение NULL; "`0000-00-00`" – если NOT NULL.

Объем: 3 байта (4 байта в версиях, предшествующих MySQL 3.22).

TIME

Описание: значение времени, представленное в формате "*hh:mm:ss*" ("*-hh:mm:ss*" – для отрицательных значений). Это значение отражает истекшее время, однако может устанавливаться и как время суток.

Допустимые атрибуты: только глобальные атрибуты.

Диапазон: от "`-838:59:59`" до "`838-59-59`".

Нулевое значение: "`00:00:00`".

Значение по умолчанию: NULL, если столбец может иметь значение NULL; "00:00:00" – если NOT NULL.

Объем: 3 байта.

Замечание: хотя значение "`00:00:00`" используется как нулевое при вставке недопустимых значений в столбец TIME, оно также является допустимым и лежит в пределах нормального диапазона.

DATETIME

Описание: значение даты и времени (обязательны обе части), представленное в формате "*CCYY-MM_DD hh:mm:ss*".

Допустимые атрибуты: только глобальные атрибуты.

Диапазон: от "`1000-01-01 00:00:00`" до "`9999-12-31 23:59:59`".

Нулевое значение: "`0000-00-00 00:00:00`".

Значение по умолчанию: NULL, если столбец может иметь значение NULL; "`0000-00-00 00:00:00`" – если NOT NULL.

Объем: 8 байт.

TIMESTAMP

Описание: временная метка, представленная в формате "*CCYYMMDDhhmmss*". Вставка значения NULL в первый столбец TIMESTAMP таблицы приводит к вставке текущего значения даты и времени. Изменение значения любого другого столбца в строке сопровождается обновлением первого столбца TIMESTAMP в соответствии с изменением даты и времени. Эти значения хранятся и рассчитываются с точностью до 14 символов, независимо от точности отображения.

Допустимые атрибуты: только глобальные атрибуты.

Диапазон: от 19700101000000 до определенного времени года 2037.

Нулевое значение: 00000000000000.

Значение по умолчанию: текущее значение даты и времени. Обратите внимание, операторы `DESRIBE` и `SHOW COLUMNS` выводят значение по умолчанию как `NULL`. Это показывает, что вы установили значение столбца `TIMESTAMP` равным текущей дате и времени, а затем создали новую запись, установив его равным `NULL`.

Объем: 4 байта.

Замечание: если в качестве атрибута определено значение `NOT NULL`, оно пропускается.

YEAR

Описание: значение года. *M* может быть равным 2 или 4 в формате *YY* или *CCYY*. Если *M* не задано, это значение по умолчанию равно 4.

Допустимые атрибуты: только глобальные атрибуты.

Диапазон: от 1900 до 2155; 0000 для `YEAR(4)`; от 1970 до 2069 для `YEAR(2)`, но отображаются только две последние цифры.

Нулевое значение: 0000 для `YEAR(4)`, 00 для `YEAR(2)`.

Значение по умолчанию: `NULL`, если столбец может иметь значение `NULL`; 0000 или 00 — если `NOT NULL`.

Объем: 1 байт.

Замечание: тип `YEAR` впервые появился в MySQL 3.22.

B

Операторы и функции

В этом приложении...

Операторы
Функции

Это приложение посвящено описанию всех используемых в операторах SQL функций MySQL. Здесь же представлен список применяемых в выражениях операторов. Описанные в этом приложении операторы и функции могут использоваться в MySQL 3.22.0 и более поздних версиях (если иное не указано). Если оператор или функция ведет себя не так, как описано в этой главе, ознакомьтесь с разделом описания изменений в *Справочном руководстве по MySQL*. Вполне вероятно, что старые версии MySQL включают программные ошибки, которые были устранены в новых версиях.

Примеры операторов и функций в этом приложении представлены в следующем формате:

expression ⇨*result*

Выражение *expression* показывает, как используется оператор или функция, а значение *result* отражает результат вычисления выражения. Например, запись

```
LOWER('ABC')    ⇨'abc'
```

показывает, что результатом выполнения функции `LOWER('ABC')` является строка "`abc`".

Все представленные далее примеры можно проверить на своем компьютере в программе `mysql`. Чтобы убедиться в достоверности предыдущего примера, вызовите `mysql`, введите оператор `SELECT` с выражением, затем точку с запятой и нажмите клавишу <Enter>.

```
mysql> SELECT LOWER('ABC');
+--------------+
| LOWER('ABC') |
+--------------+
| abc          |
+--------------+
```

MySQL не требует обязательного наличия предложения FROM в операторе SELECT, что позволяет легко экспериментировать с операторами и функциями, вводя их подобным образом. (Некоторые системы баз данных не позволяют использовать оператор SELECT без предложения FROM, что весьма неудобно.)

Примеры с полными операторами SELECT приведены лишь для функций, которые нельзя представить другим способом. Так, в разделе "Суммирующие функции" примеры написаны именно таким образом потому, что описываемые в нем функции обязательно должны задаваться по отношению к какой-нибудь таблице.

Имена функций, а также задаваемые в виде слов операторы, например BETWEEN, могут записываться как строчными, так и прописными буквами.

Некоторые типы аргументов функций встречаются постоянно, поэтому для простоты заменяются фиксированными названиями.

- *expr*. Представляет выражение, которое в зависимости от контекста может быть цифровым, строковым или выражением даты или времени. Оно также может включать константы, ссылки на столбцы таблицы и другие выражения.
- *str*. Представляет строку, которая может быть буквенной, являться ссылкой на столбец таблицы со строковыми значениями или выражением, результатом вычисления которого является строка.
- *n*. Представляет целое значение (как, впрочем, и близкие к n буквы алфавита).
- *x*. Представляет число с плавающей запятой (как, впрочем, и близкие к *x* буквы алфавита).

Имена других аргументов используются реже и объясняются в местах использования.

Необязательные части операторов и последовательностей вызовов функции заключены в квадратные скобки ([]).

Вычисление выражения зачастую сопровождается преобразованием типов значений. Более детальное описание условий, требующих подобного преобразования, а также правил преобразования значений из одного типа в другой в MySQL представлено в главе 2, "Работа с данными в MySQL".

Операторы

Операторы используются для объединения условий в выражениях и позволяют выполнять арифметические, логические и побитовые операции, а также сравнивать значения между собой и заданным образцом.

Приоритет операторов

Операторы обладают разными уровнями приоритета. Эти уровни отражены в приведенном ниже списке в порядке уменьшения. Записанные в одной строке

операторы обладают одним уровнем приоритета. В выражениях операторы с одинаковым уровнем приоритета вычисляются слева направо.

Операторы с более высоким приоритетом вычисляются до операторов с более низким приоритетом.

```
BINARY COLLATE
NOT !
^
XOR
- (унарный минус) ~ (унарное побитовое отрицание)
* / %
+ -
<< >>
&
|
< <= = <=> != <> >= > IN IS LIKE REGEXP RLIKE
BETWEEN CASE WHEN THEN ELSE
AND &&
OR ||
:=
```

Унарные операторы (унарный минус, унарное побитовое отрицание, NOT и BINARY) обладают более высоким приоритетом по сравнению с двойными. Это объясняется тем, что в выражении они группируются с непосредственно следующим за ними элементом, а не со всей оставшейся частью выражения. Это видно в следующем примере:

```
-2+3                    ⇨1
-(2+3)                  ⇨-5
```

Операторы группировки

Для группирования частей выражения могут использоваться круглые скобки, "(" и ")". Они позволяют изменить установленный по умолчанию порядок приоритетности операторов, который, в свою очередь, определяет порядок вычисления выражения (см. раздел "Приоритет операторов"). Круглые скобки могут также использоваться для внесения большей ясности, например, чтобы сделать выражение более "читабельным":

```
1 + 2 * 3 / 4                   ⇨2.50
(((1 + 2) * 3) / 4)             ⇨2.25
```

Арифметические операторы

Эти операторы выполняют стандартные арифметические действия. Они применяются к числам, но не к строкам (хотя записанные в виде чисел строки автоматически преобразуются в соответствующее числовое значение). Результатом арифметических выражений, содержащих значение NULL, является также значение NULL.

- +.

  Сложение; вычисляет сумму аргументов.

  ```
  2 + 2                   ⇨4
  3.2 + 4.7               ⇨7.9
  ```

```
'43bc' + '21d'                    ⇨64
'abc' + 'def'                     ⇨0
```

Из последнего примера видно, что оператор "+" не используется в качестве оператора конкатенации строк, как в некоторых других языках. При подобной записи строки перед выполнением арифметической операции преобразуются в числовые значения. Строки, которые не являются символьными записями чисел, преобразуются в 0. Для конкатенации строк в MySQL используется функция `CONCAT()`.

- `-`.

  Вычитание; вычисляет разность операндов при размещении между двумя элементами выражения. Изменяет знак операнда на противоположный, если указывается перед одиночным элементом.

```
10 - 7                            ⇨3
-(10 - 7)                         ⇨-3
```

- `*`.

  Умножение; вычисляет произведение операндов.

```
2 * 3                             ⇨6
2.3 * -4.5                        ⇨-10.3
```

- `/`.

  Деление; делит один операнд на другой. Деление на нуль приводит к возврату результата NULL.

```
3 / 1                             ⇨3.00
1 / 3                             ⇨0.33
1 / 0                             ⇨NULL
```

- `%`.

  Оператор модуля; вычисляет остаток от деления значения *m* на значение *n*. Запись *m* % *n* аналогична записи функции `MOD(`*m*, *n*`)`. Как и в случае с делением, этот оператор возвращает результат NULL, если делителем является нуль.

```
12 % 4                            ⇨0
12 % 5                            ⇨2
12 % 0                            ⇨NULL
```

Арифметические операции с операторами "+", "-" и "*" выполняются со значениями типа `BIGINT` (64-разрядными целыми значениями), если оба аргумента являются целыми числами. Это значит, что при использовании в выражениях больших значений диапазон вычислений 64-разрядных целых чисел может быть превышен, что приведет к получению неожиданного результата:

```
9999999999999999 * 9999999999999999   ⇨-7527149226598858751
99999999999 * 99999999999 * 99999999999   ⇨-1504485813132150785
18014398509481984 * 18014398509481984     ⇨0
```

В выражениях с операторами "/" и "%" значения типа `BIGINT` используются только в тех случаях, когда в контексте деления результат преобразуется в целое число.

Операторы сравнения

Операторы сравнения возвращают значение 1, если сравнение истинно, или значение 0, если сравнение ложно. Сравнивать можно как числа, так и строки. По необходимости операнды преобразуются в соответствии со следующими правилами.

- При сравнении со значением NULL результат возвращается равным NULL. Единственным исключением из этого правила является оператор "<=>". (Этот оператор аналогичен оператору "=", однако используется в выражениях типа NULL <=> NULL. Последнее выражение, кстати, истинно.)
- Если оба операнда являются строками, они сравниваются лексикографическим методом. При сравнении двоичных строк сравниваются цифровые значения каждого байта (байт за байтом). Сравнение недвоичных строк выполняется посимвольно с использованием того порядка сортировки, который используется для данной кодировки. Если строки имеют различные кодировки (а это стало возможно начиная с MySQL 4.1), сравнение между двоичными и недвоичными строками считается сравнением двоичных строк.
- Если оба операнда являются целыми числами, они сравниваются как целые числа.
- Начиная с версии 3.23.22 шестнадцатеричные константы сравниваются как числа. До этого шестнадцатеричные константы, не сравнимые с числом, сравнивались как двоичные строки.
- Если один из операндов имеет значение типа TIMESTAMP или DATETIME, а другой является константой, операнды сравниваются как значения типа TIMESTAMP. Это правило добавлено для повышения эффективности работы ODBC-приложений.
- Если определенное выражение не отвечает ни одному из описанных выше правил, операнды сравниваются как числовые значения с плавающей точкой. Заметьте, это относится также и к операции сравнения строковых и числовых аргументов. Строка перед сравнением преобразуется в число. Результатом этого преобразования является значение 0 (если строка не является записью числа). Строка '14.3', например, преобразуется в 14.3, а 'L4.3' – в 0.

Представленные ниже примеры иллюстрируют данные правила.

```
2 < 12                  ⇨1
'2' < '12'              ⇨0
'2' < 12                ⇨1
```

Первый пример включает в себя два целых числа, которые сравниваются как числа. Во втором примере заданы две строки, которые сравниваются лексикографически. В третьем примере выполняется сравнение строки и числа, которые преобразуются в значения с плавающей точкой.

При сравнении строк регистр символов не учитывается, если только один из сравниваемых операндов не является двоичной строкой. Следовательно, для выполнения сравнения с учетом регистра необходимо добавить ключевое слово BINARY или задать значения столбцов типов CHAR BINARY, VARCHAR BINARY или BLOB.

- =.

  Равен 1, если операнды равны, в противном случае равен 0.

```
1 = 1                          ⇨1
1 = 2                          ⇨0
'abc' = 'abc'                  ⇨1
'abc' = 'def'                  ⇨0
'abc' = 'ABC'                  ⇨1
BINARY 'abc' = 'ABC'           ⇨0
BINARY 'abc' = 'abc'           ⇨1
'abc' = 0                      ⇨1
```

  Строку 'abc' можно записать и в виде 'ABC', поскольку при сравнении строк по умолчанию регистр символов не учитывается. Использование оператора BINARY позволит выполнить сравнение с учетом регистра символов. В соответствии с правилами сравнения строка 'abc' при преобразовании в числовое значение становится равной 0, поскольку не является записью числа в виде строки.

  Текущий набор символов определяет сравнительное значение подобных символов, различаются они только диакритическими знаками.

- <=>. Оператор равенства с учетом значения NULL; подобен оператору "=" с той лишь разницей, что он равен 1, когда операнды равны, даже если они имеют значение NULL.

```
1 <=> 1                        ⇨1
1 <=>2                         ⇨0
NULL <=> NULL                  ⇨1
NULL = NULL                    ⇨NULL
```

  Из последних двух примеров видно, как отличается сравнение со значениями NULL у операторов "=" и "<=>".

  Оператор "<=>" впервые появился в MySQL 3.23.0.

- != или <>.

  Равен 1, если операнды равны, в противном случае равен 0.

```
3.4 != 3.4                     ⇨0
'abc' <> 'ABC'                 ⇨0
BINARY 'abc' <> 'ABC'          ⇨1
'abc' != 'def'                 ⇨1
```

- <.

  Равен 1, если левый операнд меньше правого операнда, в противном случае равен 0.

```
3 < 10                         ⇨1
105.4 < 10e+1                  ⇨0
'abc' < 'ABC'                  ⇨0
'abc' < 'def'                  ⇨1
```

- <=.

  Равен 1, если левый операнд меньше правого операнда либо равен ему, в противном случае равен 0.

```
'abc' <= 'a'                   ⇨0
'a' <= 'abc'                   ⇨1
```

```
13.5 <= 14                ⇨1
(3 * 4) - (6 * 2) <= 0    ⇨1
```

- >=.

  Равен 1, если левый операнд больше правого операнда либо равен ему, в противном случае равен 0.

```
'abc' >= 'a'              ⇨1
'a' >= 'abc'              ⇨0
13.5 >= 14                ⇨0
(3*4) - (6*2) >= 0        ⇨1
```

- >. Равен 1, если левый операнд больше правого, в противном случае равен 0.

```
PI () > 3                 ⇨1
'abc' > 'a'               ⇨1
SIN(0) > COS(0)           ⇨0
```

- *expr* BETWEEN *min* AND *max*.
  expr NOT BETWEEN *min* AND *max*.

  BETWEEN равен 1, если значение *min* меньше либо равно выражению *expr* и значение *max* больше либо равно *expr*. В противном случае равен 0. Для NOT BETWEEN верно обратное. Если операнды *expr*, *min* и *max* относятся к одному типу, следующие две строки будут эквивалентны:

```
expr BETWEEN min AND max
(min <= expr AND expr <= max)
```

  Если сравниваются операнды разных типов, выполняется преобразование типа и эти два выражения могут не быть эквивалентными. Сравнение оператора BETWEEN выполняется в соответствии с типом выражения *expr*.

  - Если *expr* является строкой, операнды сравниваются лексикографически как строки. Регистр символов учитывается в зависимости от того, является или не является *expr* двоичной строкой.
  - Если *expr* является целым значением, операнды сравниваются как целые числа.
  - Если ни одно из перечисленных выше правил не выполняется, операнды сравниваются как числовые значения с плавающей точкой.

```
'def' BETWEEN 'abc' and 'ghi'        ⇨1
'def' BETWEEN 'abc' and 'def'        ⇨1
13.3 BETWEEN 10 and 20               ⇨1
13.3 BETWEEN 10 and 13               ⇨0
2 BETWEEN 2 and 2                    ⇨1
'B' BETWEEN 'A' and 'a'              ⇨0
BINARY 'B' BETWEEN 'A' and 'a'       ⇨1
```

- CASE *expr* WHEN *expr1* THEN *result1* ...
  [ELSE *default*] END
  CASE WHEN *expr1* THEN *result1* ...
  [ELSE *default*] END

  Первая форма CASE сравнивает первоначальное выражение *expr* с выражениями, которые следуют за предложениями WHEN. Если первое выражение дает равенство, то результатом выполнения оператора становится со-

ответствующее значение, указанное в предложении THEN. Это удобно при сравнении заданного значения с набором значений.

```
CASE 0 WHEN 1 THEN 'T' WHEN 0 THEN 'F' END        ⇨'F'
CASE 'F' WHEN 'T' THEN 1 WHEN 'F' THEN 0 END      ⇨0
```

Вторая форма CASE оценивает выражения WHEN до тех пор, пока не найдется истинное (не равно 0 или NULL). Результатом выражения становится соответствующее значение THEN. Это очень удобно при выполнении проверок на неравенство или произвольных условий.

```
CASE WHEN 1=0 THEN 'absurd' WHEN 1=1 THEN 'obvious' END ⇨'obvious'
```

При отсутствии совпадающих выражений WHEN значение будет взято из выражения ELSE. При отсутствии выражения ELSE оператор CASE возвращает значение "NULL".

```
CASE 0 WHEN 1 THEN 'true' ELSE 'false' END        ⇨'false'
CASE 0 WHEN 1 THEN 'true' END                     ⇨NULL
CASE WHEN 1=0 THEN 'true' ELSE 'false' END        ⇨'false'
CASE WHEN 1/0 THEN 'true' END                     ⇨NULL
```

Типом значения, следующим за первым выражением THEN, определяется тип всего выражения CASE.

```
CASE 1 WHEN 0 THEN 0 ELSE 1 END                   ⇨1
CASE 1 WHEN 0 THEN '0' ELSE 1 END                 ⇨'1'
```

Конструкция CASE была добавлена в MySQL 3.23.3.

- *expr* IN (*value1*,*value2*,...);
  expr NOT IN (*value1*,*value2*,...).

Оператор IN равен 1, если выражение *expr* является одним из значений, представленных в скобках списка. В противном случае этот оператор равен 0. Оператор NOT IN работает с точностью до наоборот. В этой связи аналогичны следующие выражения:

```
expr NOT IN (value1,value2,...)
NOT (expr IN (value1,value2,...))
```

Если все значения списка являются константами, MySQL упорядочивает их и выполняет проверку IN с помощью очень быстрого двоичного поиска.

```
3 IN (1,2,3,4,5)                                  ⇨1
'd' IN ('a','b','c','d','e')                      ⇨1
'f' IN ('a','b','c','d','e')                      ⇨0
3 NOT IN (1,2,3,4,5)                              ⇨0
'd' NOT IN ('a','b','c','d','e')                  ⇨0
'f' NOT IN ('a','b','c','d','e')                  ⇨1
```

Оператор IN впервые появился в MySQL 3.21.0.

- *expr* IS NULL;
  expr IS NOT NULL.

Оператор IS NULL равен 1, если результатом выражения *expr* является значение NULL. В противном случае этот оператор равен 0. Оператор IS NOT NULL работает с точностью до наоборот. В этой связи следующие выражения эквивалентны:

```
expr IS NOT NULL
```

NOT (*expr* IS NULL)

Операторы IS NULL и IS NOT NULL следует использовать для определения, равно ли выражение *expr* значению NULL. Для подобной цели нельзя использовать формальные операторы сравнения "=" и "!=". (Начиная с MySQL 3.23 для проверки равенства пустому значению используется оператор "<=>".)

```
NULL IS NULL                    ⇨1
0 IS NULL                       ⇨0
NULL IS NOT NULL                ⇨0
0 IS NOT NULL                   ⇨1
NOT (0 IS NULL)                 ⇨1
NOT (NULL IS NULL)              ⇨0
NOT NULL IS NULL                ⇨1
```

Результат последнего примера объясняется более высоким приоритетом оператора NOT по сравнению с IS (см. раздел "Приоритет операторов").

Битовые операторы

Битовые операции выполняются над значениями типа BIGINT (64-разрядные целые), что накладывает ограничение на максимальный диапазон значений для этих операций. Результатом битовых операций, связанных с обработкой значений NULL, является также значение NULL.

- &. Соответствует побитовой операции И (пересечение) .

```
1 & 1                           ⇨1
1 & 2                           ⇨0
7 & 5                           ⇨5
```

- |. Соответствует побитовой операции ИЛИ (объединение).

```
1 | 1                           ⇨1
1 | 2                           ⇨3
1 | 2 | 4 | 8                   ⇨15
1 | 2 | 4 | 8 | 15              ⇨15
```

- ^

  Соответствует побитовой операции XOR (исключающее "ИЛИ") операндов.

```
1 ^ 1                           ⇨0
1 ^ 0                           ⇨1
255 ^ 127                       ⇨128
```

  Этот оператор был добавлен в MySQL 4.0.2. До этой версии эквивалентный результат можно было получить с помощью следующей операции над значениями *m* и *n*:

```
(m & (~n)) | ((~m) & n)
```

- <<.

  Сдвигает левый крайний операнд влево на число позиций битов, указываемых правым операндом. Сдвиг на отрицательное значение приводит к возврату нулевого значения.

```
1 << 2                          ⇨4
2 << 2                          ⇨8
```

```
1 << 62                    ⇨4611686018427387904
1 << 63                    ⇨-9223372036854775808
1 << 64                    ⇨0
```

Два последних примера наглядно иллюстрируют ограничения 64-разрядных вычислений.

Оператор "<<" впервые появился в MySQL 3.22.2.

- >>.

  Сдвигает левый крайний операнд вправо на число позиций битов, указываемых правым операндом. Сдвиг на отрицательное значение приводит к возврату нулевого значения.

```
16 >> 3                    ⇨2
16 >> 4                    ⇨1
16 >> 5                    ⇨0
```

Оператор ">>" впервые появился в MySQL 3.22.2.

- ~

  Выполняет побитовое отрицание следующего выражения. Это значит, что все биты 0 инвертируются в 1 и наоборот.

```
~0                         ⇨-1
~(-1)                      ⇨0
~~(-1)                     ⇨-1
```

Оператор "~" впервые появился в MySQL 3.23.5.

Логические операторы

Логические операторы (или булевы операторы, названные в честь математика Джорджа Буля, который формализовал их использование) проверяют истинность или ложность выражений. Все логические операторы возвращают значение 1 в случае истинности выражений или значение 0, если выражение ложно. Логические операторы интерпретируют ненулевые операнды как истину и нулевые — как ложь. Обработка значения NULL определяется в описании операторов.

Логические операторы ожидают получить числовые операнды, поэтому задаваемые строковые операнды перед вычислением преобразуются в числа.

- NOT или !.

  Логическое отрицание; равен 1, если следующий операнд ложный, или 0, если операнд истинен. Исключение составляет операнд NOT NULL, результатом которого будет значение NULL.

```
NOT 0                      ⇨1
NOT 1                      ⇨0
NOT NULL                   ⇨NULL
NOT 3                      ⇨0
NOT NOT 1                  ⇨1
NOT '1'                    ⇨0
NOT '0'                    ⇨1
NOT ''                     ⇨1
NOT 'abc'                  ⇨1
```

- OR или ||. Логическое ИЛИ; равен 1, если хотя бы один из операндов истинен (не нуль или NULL). В противном случае равен 0.

```
0 OR 0                    ⇨0
0 OR 3                    ⇨1
4 OR 2                    ⇨1
```

Поведение OR в отношении к операндам NULL (т.е. операндам с неизвестным значением) зависит от версии. В MySQL 3.23.9 OR дает 1, если результат истинный, и NULL, если результат неопределим.

```
1 OR NULL                 ⇨1
0 OR NULL                 ⇨NULL
NULL OR NULL              ⇨NULL
```

До MySQL 3.23.9 OR дает 1, если результат истинный, в противном случае — 0. В действительности NULL обрабатывается как 0.

```
1 OR NULL                 ⇨1
0 OR NULL                 ⇨0
NULL OR NULL              ⇨0
```

- AND или &&. Логическое И; равен 1, если оба операнда истинны (не нуль или NULL). В противном случае равен 0.

```
0 AND 0                   ⇨0
0 AND 3                   ⇨0
4 AND 2                   ⇨1
```

Поведение AND в отношении к операндам NULL (т.е. операндам с неизвестным значением) зависит от версии. В MySQL 3.23.9 AND дает 0, если результат ложный, и NULL, если результат неопределим.

```
1 AND NULL                ⇨NULL
0 AND NULL                ⇨0
NULL AND NULL             ⇨NULL
```

До MySQL 3.23.9 AND дает 0, если операнды NULL. В действительности NULL обрабатывается как 0.

```
1 AND NULL                ⇨0
0 AND NULL                ⇨0
NULL AND NULL             ⇨0
```

- XOR

Логическое исключающее "ИЛИ" (XOR); дает 1, если точно один из операндов истинен (не нуль и не NULL), в противном случае — 0. Дает NULL, если один из операндов равен NULL.

```
0 XOR 0                   ⇨0
0 XOR 9                   ⇨1
7 XOR 0                   ⇨1
5 XOR 2                   ⇨0
```

Этот оператор был добавлен в MySQL 4.0.2. До этой версии аналогичного результата можно добиться с помощью следующей операции над значениями *m* и *n*:

```
(m AND (NOT n)) OR ((NOT m) AND n)
```

В MySQL логические операции, как и в языке C, можно задавать с помощью символов "!", "||" и "&&". Необходимо помнить, однако, что оператор "||" не выполняет объединение строк, как в некоторых других версиях SQL. Для объединения строк можно использовать только функцию CONCAT(). (Для того чтобы оператор "||" рассматривался как оператор объединения строк, сервер необходимо запустить в режиме ANSI, задав параметр --ansi.)

Операторы преобразования

Операторы преобразования преобразуют данные одного типа в данные другого типа.

- *_charset str*

  Для того чтобы обрабатывать строковую константу (или столбец) как такую, у которой была задана определенная кодировка, предварите ее оператором *_charset* (где *charset* — имя кодировки, поддерживаемой сервером). Например, следующие выражения рассматривают строку 'abc' как имеющую соответственно кодировки latin1_de, utf8 или ucs2:

  ```
  _latin1_de 'abc'
  _utf8 'abc'
  _ucs2 'abc'
  ```

  Оператор *_charset* впервые был введен в MySQL 4.1.0.

- BINARY *str*

  Оператор BINARY заставляет программу обрабатывать следующий операнд как двоичную строку, в результате чего сравнение с этой строкой выполняется с учетом регистра символов. Если следующий операнд является числом, то он преобразуется в строковую форму:

  ```
  'abc' = 'ABC'                ⇨1
  'abc' = BINARY 'ABC'         ⇨0
  BINARY 'abc' = 'ABC'         ⇨0
  '2' < 12                     ⇨1
  '2' < BINARY 12              ⇨0
  ```

  Оператор BINARY вызывает преобразование числа в строку, после чего выполняется лексикографическое сравнение, поскольку оба операнда являются строками.

  Оператор BINARY впервые появился в MySQL 3.23.0.

- *str* COLLATE *charset*

  Оператор COLLATE производит сравнение строки *str* с использованием порядка сортировки кодировки *charset*. Это влияет на такие операции, как сравнение, сортировка, группировка и DISTINCT.

  ```
  SELECT ... WHERE col_name COLLATE utf8 > 'M';
  SELECT MAX(col_name COLLATE greek) FROM ... ;
  SELECT ... GROUP BY col_name COLLATE latin1;
  SELECT ... ORDER BY col_name COLLATE czech;
  SELECT DISTINCT col_name COLLATE latin1_de FROM ...;
  ```

  Оператор COLLATE был добавлен в MySQL 4.1.0.

Операторы сравнения с образцом

MySQL включает SQL-оператор LIKE для сравнения с образцом значений и SQL-оператор REGEXP для сравнения с образцом расширенных регулярных выражений. При сравнении с образцом SQL регистр символа не учитывается. Исключение составляют лишь случаи, когда задаваемая строка или образец сравнения представляет собой двоичные строки. При сравнении с образцом расширенных регулярных выражений регистр символов всегда учитывался до MySQL 3.23.4. Операция сравнения SQL считается успешной, если задаваемому образцу соответствует вся строка. Сравнение с образцом расширенных регулярных выражений считается успешно завершенным даже в том случае, если образец найден в любом месте строки.

- *str* LIKE *pat* [ESCAPE *'c'*];
  str NOT LIKE *pat* [ESCAPE *'c'*].

  Оператор LIKE выполняет простое SQL-сравнение с образцом и равен 1, если строка образца *pat* совпадает со всем строковым выражением *str*. Если же строка не совпадает, этот оператор равен 0. Оператор NOT LIKE функционирует с точностью до наоборот. В этой связи следующие два выражения эквивалентны:

```
str NOT LIKE pat [ESCAPE 'c']
NOT (str LIKE pat [ESCAPE 'c'])
```

  Результатом этого оператора является значение NULL, если хотя бы одна из строк имеет значение NULL.

  В SQL-образцах два символа имеют специальное значение.

  - '%'. Любая последовательность символов, кроме значения NULL (включая пустую строку).
  - '_' (подчеркивание). Соответствует любому одинарному символу.

  Задаваемые для сравнения образцы могут содержать как один такой символ, так и оба сразу.

```
'catnip' LIKE 'cat%'                ⇨1
'dogwood' LIKE '%wood'              ⇨1
'bird' LIKE '____'                  ⇨1
'bird' LIKE '___'                   ⇨0
'dogwood' LIKE '%wo__'              ⇨1
```

  Учитывать ли регистр символов при сравнении с образцом с помощью оператора LIKE, определяется сравниваемыми строками. Обычно в таких случаях регистр символов при сравнении не учитывается. Ситуация изменяется, если одна из строк является двоичной строкой.

```
'abc' LIKE 'ABC'                    ⇨1
BINARY 'abc' LIKE 'ABC'             ⇨0
'abc' LIKE BINARY 'ABC'             ⇨0
```

  Поскольку символу "%" можно поставить в соответствие любую последовательность символов, то ему же может соответствовать и полное их отсутствие.

```
'' LIKE '%'                         ⇨1
'cat' LIKE 'cat%'                   ⇨1
```

В MySQL оператор LIKE можно использовать с числовыми выражениями:

```
50 + 50 LIKE '1%'                    ⇨1
200 LIKE '2__'                       ⇨1
```

Если в строке необходимо ввести символы "%" или "_", отключив их специальное значение, достаточно впереди установить символ "\":

```
'100% pure' LIKE '100%'              ⇨1
'100% pure' LIKE '100\%'             ⇨0
'100% pure' LIKE '100\% pure'        ⇨1
```

Чтобы использовать другой символ перехода (т.е. не "\"), достаточно определить его с помощью оператора ESCAPE:

```
'100% pure' LIKE '100^%' ESCAPE ^    ⇨0
'100% pure' LIKE '100^% pure' ESCAPE ^   ⇨1
```

- *str* REGEXP *pat*
  str NOT REGEXP *pat*.

  Оператор REGEXP выполняет сравнение с образцом расширенных регулярных выражений. Оператор REGEXP равен 1, если строка образца *pat* соответствует строке выражения *expr*. В противном случае этот оператор равен 0.

  Оператор NOT REGEXP работает с точностью до наоборот. В этой связи следующие выражения эквивалентны:

  ```
  str NOT REGEXP pat
  NOT (str REGEXP pat)
  ```

  Если хотя бы одна из строк имеет значение NULL, результатом выполнения оператора также станет значение NULL.

  Образец необязательно должен соответствовать полной строке, достаточно соответствия какой-либо ее части.

  Расширенные регулярные выражения подобны образцам, используемым UNIX-утилитами grep и sed. В табл. В.1 представлены последовательности образцов, которые можно применять в MySQL.

Таблица В.1. Последовательности расширенных регулярных выражений

Последовательность	Описание
^	Соответствует началу строки
$	Соответствует концу строки
.	Соответствует любому одинарному символу, включая символ перехода на новую строку
[...]	Соответствует любому символу, расположенному между скобками
[^...]	Соответствует любому символу, не расположенному между скобками
e*	Соответствует нескольким или определенному количеству экземпляров элемента образца *e*
e+	Соответствует одному или нескольким экземплярам элемента образца *e*
e?	Соответствует нескольким или определенному количеству экземпляров элемента образца *e*
e1|e2	Соответствует элементу образца *e1* или *e2*

Последовательность	Описание
`e{m}`	Соответствует *m* экземплярам элемента образца *e*
`e{m,}`	Соответствует *m* или более экземплярам элемента образца *e*
`e{,n}`	Соответствует 0–*n* экземплярам элемента образца *e*
`e{m,n}`	Соответствует *m*–*n* экземплярам элемента образца *e*
`(...)`	Групповые элементы образца в одном элементе
Другая	Неспециальные символы, которые соответствуют самим себе

Результатом сравнения регулярного выражения является NULL, если какая-либо из строк равна NULL.

Шаблон регулярного выражения не должен соответствовать целой строке, соответствие ему может быть обнаружено где-то в строке.

```
'cats and dogs' REGEXP 'dogs'        ⇨1
'cats and dogs' REGEXP 'cats'        ⇨1
'cats and dogs' REGEXP 'c.*a.*d'     ⇨1
'cats and dogs' REGEXP 'o'           ⇨1
'cats and dogs' REGEXP 'x'           ⇨0
```

Для определения соответствия шаблону в начале или в конце строки можно использовать символы '^' или '$' соответственно.

```
'abcde' REGEXP 'b'                   ⇨1
'abcde' REGEXP '^b'                  ⇨0
'abcde' REGEXP 'b$'                  ⇨0
'abcde' REGEXP '^a'                  ⇨1
'abcde' REGEXP 'e$'                  ⇨1
'abcde' REGEXP '^a.*e$'              ⇨1
```

Конструкции `[...]` и `[^...]` определяют классы символов. Внутри класса можно определить диапазон символов, указав разделенные дефисом крайние значения диапазона. Конструкция `[a-z]`, например, соответствует любой букве нижнего регистра, а `[0-9]` соответствует любой цифре.

```
'bin' REGEXP '^b[aeiou]n$'           ⇨1
'bxn' REGEXP '^b[aeiou]n$'           ⇨0
'oboeist' REGEXP '^ob[aeiou]+st$'    ⇨1
'wolf359' REGEXP '[a-z]+[0-9]+'      ⇨1
'wolf359' REGEXP '[0-9a-z]+'         ⇨1
'wolf359' REGEXP '[0-9]+[a-z]+'      ⇨0
```

Чтобы определить в диапазоне символ "]", его нужно ввести в качестве первого символа класса. Для определения символа "-" внутри диапазона необходимо ввести его первым или последним в классе. Наконец, чтобы ввести символ "^", не нужно устанавливать его первым символом после "[".

Имеется также несколько специальных конструкций классов символов POSIX, которые могут применяться для работы с последовательностями сортировки и классами равенства (табл. B.2).

Таблица В.2. Классы символов POSIX регулярных выражений

Последовательность	Описание
`[:alnum:]`	Алфавитно-цифровые символы
`[:alpha:]`	Алфавитные символы
`[:blank:]`	Символы пробела или табуляции
`[:cntrl:]`	Управляющие символы
`[:digit:]`	Десятичные цифры (0–9)
`[:graph:]`	Графические (непустые) символы
`[:lower:]`	Символы алфавита в нижнем регистре
`[:print:]`	Графические или пустые символы
`[:punct:]`	Знаки препинания
`[:space:]`	Символы пробела, табуляции, новой строки или возврата каретки
`[:upper:]`	Символы алфавита в верхнем регистре
`[:xdigit:]`	Шестнадцатеричные цифры (0–9, a–f, A–F)

Конструкторы POSIX используются в пределах классов символов.

```
'abc' REGEXP '[[:space:]]'            ⇨0
'a c' REGEXP '[[:space:]]'            ⇨1
'abc' REGEXP '[[:digit:][:punct:]]'   ⇨0
'a0c' REGEXP '[[:digit:][:punct:]]'   ⇨1
'a,c' REGEXP '[[:digit:][:punct:]]'   ⇨1
```

Для определения специальных последовательностей внутри строк MySQL используется такой же синтаксис, как и в языке C. Последовательности "\n", "\t" и "\\", например, интерпретируются соответственно как символы перехода на новую строку, табуляции и обратной косой черты. Чтобы определить эти символы в образце, достаточно добавить впереди еще одну обратную косую черту ("\\n", "\\t", "\\\\"). Одна обратная косая черта в процессе синтаксического анализа запроса обязательно удаляется, после чего следующая за ним последовательность соответствующим образом интерпретируется при сравнении строки с образцом.

- *str* `RLIKE` *pat*
  str `NOT RLIKE` *pat*.

  Операторы `RLIKE` и `NOT RLIKE` аналогичны операторам `REGEXP` и `NOT REGEXP`.

Функции

Функции вызываются для выполнения операций и возвращения их результатов. При вызове функций пробел между именем функции и следующими за ним круглыми скобками не допускается.

`NOW()` — правильно;
`NOW ()` — неправильно.

Запуск сервера в режиме ANSI (с параметром `--ansi`) позволяет задавать пробелы после имен функций, хотя при этом наблюдается и побочный эффект: все имена

функций становятся зарезервированными словами. Такое поведение также возможно и на основании специфики соединения, что зависит от клиентской программы. Например, можно запустить клиентскую программу `mysql` с параметром `--ignore-space`; в программах, написанных на языке программирования C, можно вызвать функцию `mysql_real_connect()` с параметром `CLIENT_IGNORE_SPACE`.

Задаваемые функции задаются несколькими параметрами, разделенными запятыми. Пробелы между параметрами допустимы.

`CONCAT('abc','def')` — правильно;
`CONCAT( 'abc' , 'def')` — также правильно.

Существует несколько исключений из этих синтаксических правил — это такие функции, как `TRIM()` или `EXTRACT()`:

```
TRIM(' ' FROM ' x ')                    ⇨'x'
EXTRACT(YEAR FROM '2003-01-01')         ⇨2003
```

Для каждой функции подробно описывается ее синтаксис.

Функции сравнения

- `GREATEST (expr1,expr2,...)`. Возвращает наибольший аргумент. Наибольший среди заданных аргументов определяется в соответствии со следующими правилами.
  - Если все задаваемые в функции аргументы являются целыми значениями, то они сравниваются как целые.
  - Если все задаваемые в функции аргументы являются значениями с плавающей запятой, то они сравниваются как значения с плавающей запятой.
  - Если первые два правила не подходят, аргументы сравниваются как строки. Регистр символов при сравнении учитывается лишь в том случае, если хотя бы один из заданных аргументов является двоичной строкой.

    ```
    GREATEST(2,3,1)                     ⇨3
    GREATEST(38.5,94.2,-1)              ⇨94.2
    GREATEST('a','ab','abc')            ⇨'abc'
    GREATEST(1,3,5)                     ⇨5
    GREATEST('A','b','C')               ⇨"C"
    GREATEST(BINARY 'A','b','C')        ⇨"b"
    ```

  Функция `GREATEST()` впервые появилась в MySQL 3.22.5. В более ранних версиях вместо нее можно использовать функцию `MAX()`.
- `IF(expr1,expr2,expr3)`. Если аргумент *expr1* истинен (не 0 или `NULL`), возвращается аргумент *expr2*. В противном случае эта функция возвращает аргумент *expr3*. Функция `IF()` возвращает число или строку в соответствии с контекстом, в котором задается.

  ```
  IF(1,'true','false')                  ⇨'true'
  IF(0,'true','false')                  ⇨'false'
  IF(NULL,'true','false')               ⇨'false'
  IF(1.3,'non-zero','zero')             ⇨'non-zero'
  IF(0.3,'non-zero','zero')             ⇨'zero'
  IF(0.3 != 0,'non-zero','zero')        ⇨'non-zero'
  ```

Аргумент *expr1* вычисляется как целое значение, и в последних трех примерах показано, какая ошибка может возникнуть в результате невнимательности пользователя. Вещественное значение 1.3 преобразуется в целое значение 1, которое является истинным. Однако 0.3 преобразуется в целое значение 0, которое является ложным. В последнем примере списка показан правильный вариант использования чисел с плавающей запятой. Воспользуйтесь им при выполнении сравнения. В результате этого сравнения возвратится 1 или 0 в зависимости от результата проверки.

- `IFNULL(expr1,expr2)`. Эта функция возвращает аргумент *expr2*, если выражение *expr1* имеет значение NULL. В противном случае эта функция возвращает аргумент *expr1*. Функция `IFNULL()` возвращает число или строку в зависимости от контекста, в котором задается.

```
IFNULL(NULL,'NULL')              ⇨'NULL'
IFNULL('not NULL','NULL')        ⇨'not NULL'
```

- `INTERVAL(n,n1,n2,...)`. Эта функция возвращает значение 0, если выполняется условие *n*<*n1*. Эта функция возвращает значение 1, если *n*<*n2*, и т.д. Функция возвращает значение -1, если *n* имеет значение NULL. Значения *n1*, *n2*, ... должны быть расположены в порядке возрастания (*n1*<*n2*<...), потому что при их сравнении используется быстрый двоичный поиск. В противном случае результат выполнения функции `INTERVAL()` будет непредсказуем.

```
INTERVAL(1.1,0,1,2)              ⇨2
INTERVAL(7,1,3,5,7,9)            ⇨4
```

- `ISNULL(expr)`. Эта функция возвращает значение 1, если выражение *expr* имеет значение NULL. В противном случае эта функция возвращает значение 0.

```
ISNULL(NULL)                     ⇨1
ISNULL(0)                        ⇨0
ISNULL(1)                        ⇨0
```

- `LEAST(expr1, expr2, ...)`. Эта функция возвращает наименьший аргумент. Для определения наименьшего аргумента используются те же правила, что и для нахождения наибольшего аргумента функции `GREATEST()`.

```
LEAST(2,3,1)                     ⇨1
LEAST(38.5,94.2,-1)              ⇨-1.0
LEAST('a','ab','abc')            ⇨'a'
```

Функция `LEAST()` впервые появилась в MySQL 3.22.5. В более ранних версиях вместо нее можно использовать функцию `MIN()`.

- `STRCMP(str1, str2)`. Эта функция возвращает результат 1, 0 или -1, если первый аргумент лексически больше, равен или меньше второго. Результат будет NULL, если хотя бы один из аргументов имеет значение NULL. Начиная с MySQL 4.0.0 при сравнении регистр символов не учитывается, за исключением случаев, когда один из аргументов является двоичной строкой.

```
STRCMP('a','a')                  ⇨0
STRCMP('a','A')                  ⇨0
STRCMP('A','a')                  ⇨0
STRCMP(BINARY 'a','A')           ⇨1
STRCMP(BINARY 'A','a')           ⇨-1
```

До MySQL 4.0.0 при сравнении регистр учитывался.

```
STRCMP('a','a')                    ⇨0
STRCMP('a','A')                    ⇨1
STRCMP('A','a')                    ⇨-1
```

Функции преобразования

Эти функции предназначены для преобразования выражений из одного типа в другой.

- CAST(*expr* AS *type*)

  Преобразовать тип выражения *expr* в заданный тип. Значение *type* может быть BINARY (двоичная строка), DATE, DATETIME, TIME, SIGNED, SIGNED INTEGER, UNSIGNED или UNSIGNED INTEGER.

```
CAST(304 AS BINARY)                ⇨'304'
CAST(-1 AS UNSIGNED)               ⇨
18446744073709551615
```

  Функция CAST() может пригодиться для изменения типов столбцов при создании новой таблицы с помощью оператора CREATE TABLE ... SELECT.

```
mysql> CREATE TABLE t SELECT CAST(20020101 AS DATE) AS date_val;
mysql> SHOW COLUMNS FROM t;
+----------+------+-------+-----+------------+------+
| Field    | Type | Null  | Key | Default    | Extra |
+----------+------+-------+-----+------------+------+
| date_val | date |       |     | 0000-00-00 |       |
+----------+------+-------+-----+------------+------+
mysql> SELECT * FROM t;
+------------+
| date_val   |
+------------+
| 2002-01-01 |
+------------+
```

  Функция CAST() появилась в MySQL 4.0.2. Она аналогична функции CONVERT(), но синтаксис CAST() соответствует требованиям ANSI SQL, а синтаксис CONVERT() соответствует ODBC.

- CONVERT(*expr*, *type*)
  CONVERT(*expr* USING *charset*)

  Первая форма функции CONVERT() служит тем же целям, что и CAST(), но имеет немного отличный синтаксис. Аргументы *expr* и *type* имеют то же значение, а USING преобразует все значения в определенную кодировку.

```
CONVERT(304,BINARY)                ⇨'304'
CONVERT(-1,UNSIGNED)               ⇨18446744073709551615
CONVERT('abc' USING utf8);         ⇨'abc'
```

  Функция CONVERT() появилась в MySQL 4.0.2. Форма, имеющая синтаксис USING, была добавлена в MySQL 4.1.0.

Числовые функции

Числовые функции возвращают значение NULL, если при их работе возникает ошибка. Например, если задать функции аргументы, значения которых превышают допустимый диапазон или вовсе недопустимы, функция возвратит значение NULL.

- ABS(*x*). Эта функция возвращает абсолютное значение аргумента *x*.

```
ABS(13.5)                    ⇨13.5
ABS(-13.5)                   ⇨13.5
```

- ACOS(*x*). Эта функция возвращает арккосинус аргумента *x* или значение NULL, если *x* не находится в диапазоне от -1 до +1.

```
ACOS(1)                      ⇨0.000000
ACOS(0)                      ⇨1.570796
ACOS(-1)                     ⇨3.141593
```

  Функция ACOS() впервые появилась в MySQL 3.21.8.

- ASIN(*x*). Эта функция возвращает арксинус аргумента *x* или значение NULL, если *x* не находится в диапазоне от -1 до +1.

```
ASIN(1)                      ⇨1.570796
ASIN(0)                      ⇨0.000000
ASIN(-1)                     ⇨-1.570796
```

  Функция ASIN() впервые появилась в MySQL 3.21.8.

- ATAN(*x*)
  ATAN(*y*,*x*).

  Эта функция возвращает арктангенс аргумента *x*.

```
ATAN(1)                      ⇨0.785398
ATAN(0)                      ⇨0.000000
ATAN(-1)                     ⇨-0.785398
```

  Функция ATAN() впервые появилась в MySQL 3.21.8.

- ATAN2(*x*,*y*). Эта функция подобна ATAN(*x*,*y*), однако для вычисления квадранта возвращаемого значения в ней учитываются знаки аргументов.

```
ATAN2(1,1)                   ⇨0.785398
ATAN2(1,-1)                  ⇨2.356194
ATAN2(-1,1)                  ⇨-0.785398
ATAN2(-1,-1)                 ⇨-2.356194
```

- CEILING(*x*). Эта функция возвращает наименьшее целое, не меньшее аргумента *x*.

```
CEILING(3.8)                 ⇨4
CEILING(-3.8)                ⇨-3
```

- COS(*x*). Эта функция возвращает косинус аргумента *x*, заданного в радианах.

```
COS(0)                       ⇨1.000000
COS(PI())                    ⇨-1.000000
COS(PI()/2)                  ⇨0.000000
```

  Функция COS() впервые появилась в MySQL 3.21.8.

- COT(*x*). Эта функция возвращает котангенс аргумента *x*, заданного в радианах.

```
COT(PI()/2)                    ⇨0.00000000
COT(PI()/4)                    ⇨1.00000000
```

Функция COT() впервые появилась в MySQL 3.21.16.

- CRC32(*str*)

  Вычисляет значение кода циклической проверки избыточности аргумента, который является строкой. Возвращаемое значение является 32-битовым значением без знака в диапазоне от 0 до 232–1 или NULL, если аргумент имеет значение "NULL".

```
CRC32('xyz')                   ⇨3951999591
CRC32('0')                     ⇨4108050209
CRC32(0)                       ⇨4108050209
CRC32(NULL)                    ⇨NULL
```

Функция CRC32 впервые появилась в MySQL 4.1.0.

- DEGREES(*x*). Эта функция возвращает значение аргумента *x*, преобразованное из радиан в градусы.

```
DEGREES(PI())                  ⇨180
DEGREES(PI()*2)                ⇨360
DEGREES(PI()/2)                ⇨90
DEGREES(-PI())                 ⇨-180
```

Функция DEGREES() впервые появилась в MySQL 3.21.16.

- EXP(*x*). Эта функция возвращает значение e^x, где e — основание натурального логарифма.

```
EXP(1)                         ⇨2.718282
EXP(2)                         ⇨7.389056
EXP(-1)                        ⇨0.367879
1/EXP(1)                       ⇨0.36787944
```

- FLOOR(*x*). Эта функция возвращает наибольшее целое значение, не большее аргумента *x* (другими словами, дробная часть аргумента *x* просто отбрасывается).

```
FLOOR(3.8)                     ⇨3
FLOOR(-3.8)                    ⇨-4
```

- LN(*x*)

  Является синонимом функции LOG(); впервые появилась в MySQL 4.0.3.

- LOG(*x*)

  LOG(*b*,*x*). Эта функция возвращает натуральный (с основанием e) логарифм аргумента *x*.

```
LOG(0)                         ⇨NULL
LOG(1)                         ⇨0.000000
LOG(2)                         ⇨0.693147
LOG(EXP(1)   )                 ⇨1.000000
```

Формат с двумя аргументами возвращает логарифм *x* с основанием *b*.

```
LOG(10,100)                    ⇨2.00000000
LOG10(2,256)                   ⇨2.000000
```

Формат с двумя аргументами впервые появился в MySQL 4.0.3. В ранних версиях функцию LOG() можно использовать для вычисления логарифма с произвольным основанием *b* аргумента *x*, используя формулу LOG(*x*)/LOG(*b*).

```
LOG(100)/LOG(10)                ⇨2.00000000
LOG10(100)                      ⇨2.000000
```

- LOG10(*x*). Эта функция возвращает логарифм с основанием 10 аргумента *x*.

```
LOG10(0)                        ⇨NULL
LOG10(10)                       ⇨1.000000
LOG10(100)                      ⇨2.000000
```

- LOG2(*x*). Эта функция возвращает логарифм с основанием 2 аргумента *x*.

```
LOG2(0)                         ⇨NULL
LOG2(255)                       ⇨7.994353
LOG2(32767)                     ⇨14.999956
```

Функция LOG2() позволяет определить длину значения в битах. Одним из его применений является определение памяти, необходимой для хранения значения. Функция LOG2() впервые появилась в MySQL 4.0.3.

- MOD(*m*, *n*). Эта функция аналогична оператору *m*%*n*. Более детально о его работе рассказывается в разделе "Арифметические операторы".
- PI(). Возвращает значение π.

```
PI()                            ⇨3.141593
```

Функция PI() впервые появилась в MySQL 3.21.8.

- POW(*x*, *y*). Возвращает значение x^y, другими словами, результат возведения аргумента *x* в степень *y*.

```
POW(2, 3)                       ⇨8.000000
POW(2,-3)                       ⇨0.125000
POW(4,.5)                       ⇨2.000000
POW(16,.25)                     ⇨2.000000
```

- POWER(*x*,*y*). Эта функция является аналогом функции POW() и впервые появилась в MySQL 3.21.16.
- RADIANS(*x*). Эта функция возвращает значение аргумента *x*, преобразованное из градусов в радианы.

```
RADIANS(0)                      ⇨0
RADIANS(360)                    ⇨6.2831853071796
RADIANS(-360)                   ⇨-6.2831853071796
```

Функция RADIANS() впервые появилась в MySQL 3.21.16.

- RAND();
  RAND(*n*).

Функция RAND() возвращает случайное значение с плавающей точкой в диапазоне от 0.0 до 1.0. Функция RAND(*n*) выполняет аналогичное действие, используя *n* в качестве начального значения для генератора случайных чисел. Все вызовы с одним и тем же значением *n* возвратят один и тот же результат. Это свойство можно использовать, когда нужно сгенерировать повторяющуюся последовательность чисел. (В первый раз RAND() вызывается с аргументом *n*, а затем для получения следующих чисел в последовательности вызывается без аргументов.)

```
RAND(10)                    ⇨0.18109053110805
RAND(10)                    ⇨0.18109053110805
RAND()                      ⇨0.7502322306393
RAND()                      ⇨0.20788959060599
RAND(10)                    ⇨0.18109053110805
RAND()                      ⇨0.7502322306393
RAND()                      ⇨0.20788959060599
```

В этих примерах можно увидеть, какие значения возвращает функция RAND() при задании аргумента и его пропуске.

Задание начального значения для всех клиентов производится независимо друг от друга. Когда один клиент вызывает функцию RAND(*n*) для получения случайного числа, это никак не влияет на результаты, полученные другими клиентами.

- ROUND(*x*);
  ROUND(*x*, *d*);

  Функция ROUND(*x*) возвращает значение аргумента *x*, округленное до целого числа. Функция ROUND(*x*, *d*) возвращает значение аргумента *x*, округленное до числа с *d* знаками после запятой. Если аргумент *d* равен 0, возвращаемый результат не имеет десятичной точки или дробной части.

```
ROUND(15.3   )              ⇨15
ROUND(15.5)                 ⇨16
ROUND(-33.27834,2)          ⇨-33.28
ROUND(1, 4)                 ⇨1.0000
```

  Точное поведение функции ROUND() зависит от поведения при операциях округления вашей математической библиотеки. Это значит, что результаты, которые можно получить с помощью функции ROUND(), могут варьироваться от системы к системе.

- SIGN(*x*). Эта функция возвращает значение -1, 0 или 1, если значение аргумента *x* соответственно отрицательно, равно нулю или положительно.

```
SIGN(15.803)                ⇨1
SIGN(0)                     ⇨0
SIGN(-99)                   ⇨-1
```

- SIN(*x*). Эта функция возвращает синус аргумента *x*, заданного в радианах.

```
SIN(0)                      ⇨0.000000
SIN(PI())                   ⇨0.000000
SIN(PI()/2)                 ⇨1.000000
```

  SIN(). Эта функция впервые появилась в MySQL 3.21.8.

- SQRT(*x*). Эта функция возвращает положительный квадратный корень аргумента *x*.

```
SQRT(625)                   ⇨25.000000
SQRT(2.25)                  ⇨1.500000
SQRT(-1)                    ⇨NULL
```

- TAN(*x*). Эта функция возвращает тангенс аргумента *x*, заданного в радианах.

```
TAN(0)                      ⇨0.000000
TAN(PI()/4)                 ⇨1.000000
```

Функция TAN() впервые появилась в MySQL 3.21.8.

- TRUNCATE(*x*, *d*). Возвращает значение аргумента *x* с дробной частью, усеченной до *d* знаков после запятой. Если аргумент *d* равен нулю, возвращаемый результат не имеет десятичной точки или дробной части. Если *d* больше, чем количество десятичных разрядов в *x*, то к концу дробной части добавляется соответствующее количество нулей.

```
TRUNCATE(1.23,1)          ⇨1.2
TRUNCATE(1.23,0)          ⇨1
TRUNCATE(1.23,4)          ⇨1.2300
```

Функция TRUNCATE() впервые появилась в MySQL 3.21.16.

Строковые функции

Большинство описанных в этом разделе функций возвращает строковое значение. Некоторые из них, такие как LENGTH(), используют строку в качестве аргумента и возвращают числовое значение. Для функций, строки которых задаются по позиции внутри другой строки, первый (самый левый) символ строки занимает позицию 1 (а не 0).

Начиная с MySQL 3.23.7 некоторые строковые функции рассчитаны на многобайтовые операции: INSERT(), INSTR(), LCASE(), LEFT(), LOCATE(), LOWER(), MID(), POSITION(), REPLACE(), REVERSE(), RIGHT(), RTRIM(), SUBSTRING(), SUBSTRING_INDEX(), TRIM(), UCASE() и UPPER().

- ASCII(*str*). Возвращает ASCII-код самого левого символа строки *str*. Эта функция возвращает значение 0, если строка *str* является пустой, и значение NULL, если строка *str* имеет значение NULL.

```
ASCII('abc')              ⇨97
ASCII('')                 ⇨0
ASCII(NULL)               ⇨NULL
```

Функция ASCII() впервые появилась в MySQL 3.21.2.

- BIN(*n*). Возвращает значение аргумента *n* в виде строки, записанной в двоичной форме. В этой связи следующие два выражения эквивалентны:

```
BIN(n)
CONV(n,10,2)
```

Подробнее о функции можно узнать в описании функции CONV().

Функция BIN() впервые появилась в MySQL 3.22.4.

- CHAR(*n1*, *n2*, ...). Считывает аргументы как ASCII-коды и возвращает строку, созданную путем объединения соответствующих им значений символов. Аргументы со значениями NULL игнорируются.

```
CHAR(65)                  ⇨'A'
CHAR(97)                  ⇨'a'
CHAR(89,105,107,101,115,33)  ⇨'Yikes!'
```

Функция CHAR() впервые появилась в MySQL 3.21.0.

- CHARACTER_LENGTH(*str*). Эта функция является аналогом функции CHAR_LENGTH().

- CHAR_LENGTH(). Эта функция является аналогом функции LENGTH(). За исключением того, что начиная с MySQL 3.23.6 многобайтовые символы считаются как имеющие длину 1.
- CHARSET(*str*)

  Возвращает имя кодировки, к которой представлена строка *str*.

```
CHARSET('abc')                          ⇨'latin1'
CHARSET(CONVERT('abc' USING utf8))      ⇨'utf8'
```

  Функция CHARSET() была добавлена в MySQL 4.1.0.
- COALESCE(*expr1*, *expr2*, . . .). Возвращает первый непустой элемент списка или NULL, если нет непустых аргументов.

```
COALESCE(NULL,1/0,2,'a',45+97)          ⇨'2'
COALESCE(NULL,1/0)                      ⇨NULL
```

  Функция COALESCE() впервые появилась в MySQL 3.23.3.
- CONCAT(*str1*, *str2*, . . .). Возвращает строку, созданную путем объединения всех аргументов. Возвращает значение NULL, если хотя бы один из аргументов имеет значение NULL. Функцию CONCAT() можно вызывать с одним аргументом.

```
CONCAT('abc','def')                     ⇨'abcdef'
CONCAT('abc')                           ⇨'abc'
CONCAT('abc',NULL)                      ⇨NULL
CONCAT('Hello',',','goodbye')           ⇨'Hello, goodbye'
```

  Если аргументы функции CONCAT() имеют различные кодировки (как это стало возможно начиная с MySQL 4.1), полученный результат будет иметь кодировку первого аргумента.

  Другой способ объединения строк заключается в задании их друг за другом.

```
'three' 'blind' 'mice'                  ⇨'threeblindmice'
'abc' 'def' = 'abcdef'                  ⇨1
```

- CONCAT_WS(*delim*, *str1*, *str2*, . . .)

  Возвращает строку, состоящую из строки, объединяющей второй и следующий аргументы, разделителем между которыми является строка *delim*. Возвращает значение "NULL" при *delim*, равном "NULL", но игнорирует любые пустые значения или пустые строки в списке объединяемых строк.

```
CONCAT_WS(',','a','b','c','d')          ⇨'a,b,c,d'
CONCAT_WS('*-*','lemon','','lime',NULL,'grape')   ⇨'lemon*-
*lime*-*grape'
```

- CONV(*n*, *from_base*, *to_base*). Считывая число *n*, записанное с основанием *from_base*, возвращает его строковое представление с основанием *to_base*. Результатом этой функции является значение NULL, если хотя бы один из аргументов имеет значение NULL. Аргументами *from_base* и *to_base* могут быть целые числа в диапазоне от 2 до 36. Аргумент *n* считывается как значение типа BIGINT (64-разрядное целое значение), однако может задаваться и в виде строки, поскольку числа с основанием выше 10 могут содержать нецифровые символы. (Именно поэтому функция CONV() возвращает строку; для чисел с основанием от 11 до 36 результат может содержать символы от "A" до "Z".) Если *n* явля-

ется недопустимой записью числа с заданным основанием *from_base*, возвращается значение 0. (Например, если аргумент *from_base* равен 16, а *n* – "abcdefg", функция возвратит значение 0, поскольку символ "g" не используется в шестнадцатеричных числах.)

Нецифровые символы в аргументе *n* могут задаваться как строчными, так и прописными буквами. В возвращаемом функцией результате эти символы записаны прописными буквами.

Для преобразования заданного в шестнадцатеричной системе исчисления числа 14 в двоичную можно воспользоваться следующей функцией:

```
CONV("e",16,2)                    ⇨"1110"
```

Чтобы преобразовать число 255 из двоичной записи в восьмеричную, достаточно ввести одну из следующих функций:

```
CONV(11111111,2,8)                ⇨"377"
CONV("11111111",2,8)              ⇨"377"
```

Аргумент *n* по умолчанию интерпретируется как число без знака. Если же значение *to_base* представлено отрицательным числом, *n* рассматривается как число со знаком.

```
CONV(-10,10,16)                   ⇨'FFFFFFFFFFFFFFF6'
CONV(-10,10,-16)                  ⇨'-A'
```

Функция CONV() впервые появилась в MySQL 3.22.4.

- ELT(*n*,*str1*,*str2*,...). Эта функция возвращает *n*-ю строку из списка аргументов-строк *str1*, *str2*, ... Если строка с номером *n* отсутствует, либо же эта строка или число *n* имеет значение NULL, функция также возвращает значение NULL. Первая задаваемая строка имеет индекс 1. Функция ELT() является дополнением функции FIELD().

```
ELT(3,'a','b','c','d','e')              ⇨'c'
ELT(0,'a','b','c','d','e')              ⇨NULL
ELT(6,'a','b','c','d','e')              ⇨NULL
ELT(FIELD('b','a','b','c'),'a','b','c') ⇨'b'
```

- EXPORT_SET(*n*,*on*,*off*,[*separator*,[*bit_count*]]). Эта функция возвращает строку, состоящую из записей *on* и *off*, разделенных строкой *separator*. Запись *on* используется на позиции каждого разряда, установленного в значении *n*, а *off* – на позиции каждого свободного разряда. Аргумент *bit_count* указывает максимальное число проверяемых в значении *n* разрядов. По умолчанию строкой *separator* является запятая, а значение *bit_count* равно 64. Возвращает значение "NULL", если один из аргументов равен "NULL".

```
EXPORT_SET(7,'+','-','',5)        ⇨'+++--'
EXPORT_SET(0xa,'1','0','',6)      ⇨'010100'
EXPORT_SET(97,'Y','N',',',8)      ⇨'Y,N,N,N,N,Y,Y,N'
```

Функция EXPORT_SET() впервые появилась в MySQL 3.23.2.

- FIELD(*str*,*str1*,*str2*,...). Находит строку *str* в списке строк *str1*, *str2*, ... и возвращает номер совпадающей строки. Если подобная строка отсутствует или же строка *str* имеет значение NULL, функция возвра-

щает значение 0. Первая строка имеет значение 1. Функция FIELD() зачастую используется вместе с функцией ELT().

```
FIELD('b','a','b','c')                    ⇨2
FIELD('d','a','b','c')                    ⇨0
FIELD(NULL,'a','b','c')                   ⇨0
FIELD(ELT(2,'a','b','c'),'a','b','c')     ⇨2
```

- FIND_IN_SET(*str*, *str_list*). Аргумент *str_list* представляет собой строку, состоящую из разделенных запятыми подстрок. Функция FIND_IN_SET() возвращает индекс строки *str* внутри списка *str_list*, значение 0, если строка *str* отсутствует в *str_list*, либо значение NULL, если хотя бы один из аргументов имеет значение NULL. Первая подстрока имеет индекс 1.

```
FIND_IN_SET('cow','moose,cow,pig')        ⇨2
FIND_IN_SET('dog','moose,cow,pig')        ⇨0
```

Функция FIND_IN_SET() впервые появилась в MySQL 3.21.22.

- FORMAT(*x*, *d*). Преобразует числовой аргумент *x* в строку с *d* знаками после запятой, записанную в формате 'nn,nnn.nnn'. Если аргумент *d* имеет нулевое значение, возвращаемый результат не включает десятичную точку или дробную часть.

```
FORMAT(1234.56789,3)              ⇨'1 234.568'
FORMAT(999999.99,2)               ⇨'999 999.99'
FORMAT(999999.99,0)               ⇨'1 000 000'
```

Обратите внимание на округление, выполненное в последнем примере списка, а также на эффект округления.

- HEX(*n*)

  HEX(*str*)

  . Возвращает значение аргумента *n*, записанное в виде шестнадцатеричной строки. Эта функция подобна функции CONV(), поэтому следующие два выражения эквивалентны:

```
HEX(n)
CONV(n,10,16)
```

Более детально о преобразовании рассказывается в описании функции CONV().

До MySQL 4.0.1 функция HEX() всегда интерпретировала свой аргумент как строку.

```
HEX(255)                          ⇨'FF'
HEX('255')                        ⇨'FF'
```

Начиная с MySQL 4.0.1 функция HEX() может принимать строковый аргумент и возвращать строку, состоящую из каждого символа в аргументах, представленных двумя шестнадцатеричными цифрами.

```
HEX('255')                        ⇨'323535'
HEX('abc')                        ⇨'616263'
```

Функция HEX() впервые появилась в MySQL 3.22.4.

- INSERT(*str*,*pos*,*len*,*new_str*). Возвращает строку *str*, в которой подстрока, начинающаяся с позиции *pos* и имеющая длину *len* символов, заменена строкой *new_str*. Возвращает исходную строку, если значение *pos*

находится за пределами данной строки, или значение NULL, если хотя бы один из аргументов имеет значение NULL.

```
INSERT('nighttime',6,4,'fall')        ⇨'nightfall'
INSERT('sunshine',1,3,'rain or ')     ⇨'rain or shine'
INSERT('sunshine',0,3,'rain or ')     ⇨'sunshine'
```

- INSTR(*str,substr*). Функция INSTR() является аналогом функции LOCATE() с двумя переставленными аргументами. Соответственно, эквивалентны два следующих выражения:

  INSTR(*str,substr*) и
  LOCATE(*substr,str*)

- LCASE(*str*).

  Эта функция аналогична LOWER().

- LEFT(*str,len*). Возвращает *len* левых крайних элементов из строки *str* или всю строку, если в ней меньше символов, чем определено аргументом *len*. Возвращает значение NULL, если аргумент *str* равен NULL. Возвращает пустую строку, если аргумент *len* имеет значение NULL или меньшее 1.

```
LEFT('my left foot',2)          ⇨'my'
LEFT(NULL,10)                   ⇨NULL
LEFT('abc',NULL)                ⇨''
LEFT('abc',0)                   ⇨''
```

- LENGTH(*str*). Возвращает длину строки *str*.

```
LENGTH('abc')                   ⇨3
LENGTH('')                      ⇨0
LENGTH(NULL)                    ⇨NULL
```

- LOCATE(*substr,str*); LOCATE(*substr,str,pos*). Форма функции LOCATE() с двумя аргументами возвращает позицию первого появления подстроки *substr* в строке *str*. Если строка *substr* не встречается в *str*, эта функция возвращает значение 0. Если хотя бы один из аргументов функции имеет значение NULL, функция также возвращает NULL. При наличии аргумента позиции *pos* функция LOCATE() начинает искать строку *substr* на заданном месте. Регистр символов при сравнении учитывается.

```
LOCATE('b','abc')               ⇨2
LOCATE('b','ABC')               ⇨0
LOCATE(BINARY 'b','ABC')        ⇨0
```

  До MySQL 4.0.1 этот тест был чувствительным к регистру.

```
LOCATE('b','abc')               ⇨2
LOCATE('b','ABC')               ⇨0
```

- LOWER(*str*). Эта функция возвращает строку *str*, все символы которой записаны строчными символами, или значение NULL, если аргумент *str* равен NULL.

```
LOWER('New York,NY')            ⇨'new york, ny'
LOWER(NULL)                     ⇨NULL
```

- LPAD(*str,len,pad_str*). Возвращает подстроку строки *str*, дополненную слева до указанной длины *len* (символов) строкой *left_pad*. Возвращает только строку *str*, если ее длина уже составляет *len* символов.

```
LPAD('abc',12,'def')              ⇨'defdefdefabc'
LPAD('abc',10,'.')                ⇨'.......abc'
```

Начиная с MySQL 3.23.29 LPAD() усекает полученный результат до *len* символов.

```
LPAD('abc',2,'.')                 ⇨'ab'
```

До MySQL 3.23.29 функция LPAD() возвращает *str*, если *str* уже имеет длину *len*.

```
LPAD('abc',2,'.')                 ⇨'abc'
```

Функция LPAD() впервые появилась в MySQL 3.22.2.

- LTRIM(*str*). Возвращает подстроку строки *str*, удаляя все стоящие впереди нее пробелы, или значение NULL, если строка *str* имеет значение NULL.

```
LTRIM('  abc  ')          ⇨'abc  '
```

- MAKE_SET(*n,bit0_str,bit1_str,...*). Создает значение типа SET (строку, состоящую из разделенных запятыми подстрок), основываясь на значении целого числа *n* и строк *bit0_str*, *bit1_str*, ... Для каждого разряда, установленного в двоичной записи значения *n*, в результат включается соответствующая строка. (Если установлен нулевой разряд, возвращаемый результат включает строку *bit0_str* и т.п.) Если аргумент *n* равен нулю, результатом является пустая строка. Если аргумент *n* имеет значение NULL, возвращаемый результат также имеет значение NULL. Все строки списка, имеющие значение NULL, при создании результирующей строки игнорируются.

```
MAKE_SET(8,'a','b','c','d','e')          ⇨ 'd'
MAKE_SET(7,'a','b','c','d','e')          ⇨ 'a,b,c'
MAKE_SET(2+16,'a','b','c','d','e')       ⇨ 'b,e'
MAKE_SET(2|16,'a','b','c','d','e')       ⇨ 'b,e'
MAKE_SET(-1,'a','b','c','d','e')         ⇨ 'a,b,c,d,e'
```

Результат последнего примера включает все заданные строки, поскольку в двоичной записи числа -1 все разряды установлены.

Функция MAKE_SET() впервые появилась в MySQL 3.22.2.

- MATCH(*column_list*) AGAINST(*str*)
  MATCH(*column_list*) AGAINST(*str* IN BOOLEAN MODE)

Функция MATCH выполняет операции поиска с использованием индекса FULLTEXT. Список MATCH состоит из одного или более имен столбцов, разделенных запятыми. Это должны быть столбцы, входящие в индекс FULLTEXT таблицы, по которому будет осуществляться поиск. Аргументом *str* в выражении AGAINST() отражены слово или слова, поиск которых будет осуществляться в данном столбце. Слова — это последовательность символов, состоящих из букв, цифр, одинарных кавычек или символов подчеркивания. Скобки допустимы для выражения MATCH, а для AGAINST они недопустимы.

MATCH производит релевантное ранжирование для каждой строки. Ранг — это неотрицательное число с плавающей точкой; при этом ранг, равный нулю, является индикатором того, что слова не найдены. Положительные значения отражают тот факт, что по крайней мере одно искомое слово было

найдено. Слова, представленные более чем в половине строк таблицы, считаются как имеющие нулевую релевантность, так как они достаточно общие. Кроме того, СУБД MySQL снабжена внутренним списком стоп-слов (таких как "the" и "but"), которые никогда не считаются релевантными.

При сопровождении искомой строки ключевым предложением `IN BOOLEAN MODE` искомые результаты базируются в основном на отсутствии или наличии искомых слов, независимо от того, насколько часто они встречаются в таблице. При двоичном поиске слова в искомой строке могут сопровождаться следующими операторами для того, чтобы как-то воздействовать на процесс поиска:

- \+ или -

  Ведущий знак "+" или "-" свидетельствует о том, что слово должно присутствовать или отсутствовать.

- < или >

  Ведущий знак "<" или ">" увеличивает или уменьшает вклад слова в вычисление значения релевантности.

- ~

  Ведущий символ "~" инвертирует вклад слова в вычисление значения релевантности, но полностью не исключает строки, содержащие слово, как это делает знак "-".

- *

  Завершающий символ "*" действует как групповой оператор. Например, по значению "`act*`" соответствует "`act`", "`acts`", "`action`" и т.д..

- "фраза"

  Искомую фразу необходимо заключить в двойные кавычки. Каждое слово должно быть представлено в порядке, в котором оно задано в фразе.

- ()

  Скобки группируют слова в выражения.

Слова без модификаторов при логическом поиске рассматриваются как необязательные, аналогично нелогическому поиску.

Логический режим поиска можно задать без индекса FULLTEXT, но такой поиск будет довольно медленным.

Поиск FULLTEXT впервые появился в MySQL 3.23.23, логический режим поиска в MySQL 4.0.1 и поиск по фразам – в MySQL 4.0.2.

- `MID(`*`str`*`,`*`pos`*`,`*`len`*`)` `MID(`*`str`*`,`*`pos`*`)`. Формат с тремя аргументами возвращает подстроку строки *`str`*, которая начинается с позиции *`pos`* и имеет длину *`len`* символов. Формат с двумя аргументами возвращает подстроку строки *`str`*, которая начинается с позиции *`pos`* и завершается концом строки. Возвращает NULL, если хотя бы один из аргументов имеет значение NULL.

```
MID('what a dull example',8,4)        ⇨ 'dull'
MID('what a dull example',8)          ⇨ 'dull example'
```

MID() является синонимом функции SUBSTRING() и может использоваться с любой формой синтаксиса, которую позволяет функция SUBSTRING().

- OCT(*n*). Возвращает значение *n*, записанное в виде строки в восьмеричной форме. Поэтому два следующих выражения эквивалентны:

  OCT(*n*) и

  CONV(*n*,10,8)

  Более детально о преобразовании рассказывается в описании функции CONV().

  Функция OCT() впервые появилась в MySQL 3.22.4.

- OCTET_LENGTH(*str*). Эта функция является аналогом функции LENGTH().

- ORD(*str*)

  Возвращает цифровое значение первого символа строки *str* или NULL, если *str* имеет значение "NULL". Если первый символ не является многобайтовым символом, значение ORD() равносильно значению ASCII().

```
ORD('abc')                    ⇨97
ASCII('abc')                  ⇨97
```

  Для многобайтовых символов функция ORD() возвращает значение, которое можно определить из значений ASCII отдельных байтов символа от *b1* до *bn*. Формула для вычисления этого значения имеет следующий вид:

```
( ... ((ASCII(b1)*256 + ASCII(b2)))*256 + ...) + ASCII(bn)
```

  Функция ORD() впервые была добавлена в MySQL 3.23.6.

- POSITION(*substr* IN *str*). Эта функция подобна функции LOCATE() с двумя аргументами. Соответственно, два следующих выражения эквивалентны:

```
POSITION(substr IN str) и
LOCATE(substr,str)
QUOTE(str)
```

  Эта функция обрабатывает свой аргумент и возвращает строку, которая правильно заключена в кавычки и готова для использования в операторе SQL. Она может пригодиться при создании запросов, в результате работы которых получаются другие запросы. Для непустых значений возвращаемое значение будет заключено в одинарные кавычки. Символ NUL в кодировке ASCII, обратная косая черта и комбинация <Ctrl+Z> предваряются обратной косой чертой, а полученный результат заключается в одинарные кавычки.

  Для пустых значений возвращаемым значением является слово "NULL" без одинарных кавычек.

```
QUOTE('X')                    ⇨'X'
QUOTE("'")                    ⇨'\''
QUOTE(NULL)                   ⇨NULL
```

  Функция QUOTE() впервые была введена в MySQL 4.0.3.

- REPEAT(*str*,*n*). Возвращает строку, состоящую из *n* повторений строки *str*. Возвращает пустую строку, если аргумент *n* имеет отрицательное значение. Если хотя бы один из аргументов имеет значение NULL, функция также возвращает значение NULL.

```
REPEAT('x',10)                  ⇨ 'xxxxxxxxxx'
REPEAT('abc',3)                 ⇨ 'abcabcabc'
```

Функция REPEAT() впервые появилась в MySQL 3.21.10.

- REPLACE(*str*,*from_str*,*to_str*). Возвращает строку *str*, в которой все экземпляры подстроки *from_str* заменены строкой *to_str*. Возвращает значение NULL, если хотя бы один из аргументов имеет значение NULL. Если аргумент *to_str* является пустой строкой, все имеющиеся подстроки *from_str* удаляются. Если аргумент *from_str* представляет собой пустую строку, функция REPLACE() возвращает строку *str* без изменений.

```
REPLACE('abracadabra','a','oh')     ⇨'ohbrohcohdohbroh'
REPLACE('abracadabra','a','')       ⇨ 'brcdbr'
REPLACE('abracadabra','','x')       ⇨ 'abracadabra'
```

- REVERSE(*str*). Возвращает строку, состоящую из строки *str* с записанными наоборот символами, либо значение NULL, если аргумент *str* равен NULL.

```
REVERSE('abracadabra')              ⇨ 'arbadacarba'
REVERSE('Madam, Im Adam')           ⇨ 'madA mI ,madaM'
```

Функция REVERSE() впервые появилась в MySQL 3.21.19.

- RIGHT(*str*,*len*). Возвращает *len* правых крайних символов строки *str* или всю строку, если она короче *len*. Возвращает значение NULL, если аргумент *str* равен NULL. Возвращает пустую строку, если значение аргумента *len* равно NULL или меньше 1.

```
RIGHT('rightmost',4)                ⇨ 'most'
```

- RPAD(*str*,*len*,*pad_str*). Возвращает строку *str*, дополненную справа до заданной длины *len* (символов) строкой *pad_str*. Возвращает только строку *str*, если ее длина уже составляет *len* символов.

```
RPAD('abc',12,'def')            ⇨ 'abcdefdefdef'
RPAD('abc',10,'.')              ⇨ 'abc.......'
```

Начиная с MySQL 3.23.29 функция RPAD() усекает результат до *len* символов.

```
RPAD('abc',2,'.')               ⇨ 'ab'
```

До MySQL 3.23.29 функция RPAD() возвращает *str*, если длина *str* уже равна *len*.

```
RPAD('abc',2,'.')               ⇨ 'abc'
```

Функция RPAD() впервые появилась в MySQL 3.22.2.

- RTRIM(*str*). Возвращает строку *str*, удалив пробелы в конце, или значение NULL, если строка *str* равна NULL.

```
RTRIM(' abc ')                  ⇨ ' abc'
```

- SOUNDEX(*str*). Возвращает строку, отражающую произношение содержимого строки *str*. Небуквенные символы строки *str* пропускаются. Международные небуквенные символы, не входящие в диапазон букв от A до Z, обрабатываются как гласные.

```
SOUNDEX('Cow')                  ⇨ 'C000'
SOUNDEX('Cowl')                 ⇨ 'C400'
```

```
SOUNDEX('Howl')                ⇨ 'H400'
SOUNDEX('Hello')               ⇨ 'H400'
```

- SPACE(*n*). Возвращает строку, состоящую из *n* пробелов, или пустой набор, если аргумент *n* имеет отрицательное значение. Если аргумент *n* равен NULL, функция также возвращает значение NULL.

```
SPACE(6)                       ⇨ '      '
SPACE(0)                       ⇨ ''
SPACE(NULL)                    ⇨NULL
```

Функция SPACE() впервые появилась в MySQL 3.21.16.

- SUBSTRING(*str*,*pos*); SUBSTRING(*str*,*pos*,*len*); SUBSTRING(*str* FROM *pos*); SUBSTRING(*str* FROM *pos* FOR *len*). Возвращает подстроку строки *str*, начинающуюся с позиции *pos*. Если задан аргумент *len*, возвращается подстрока длиной *len* символов. В остальных случаях возвращается вся правая крайняя часть строки *str*, начиная с позиции *pos*.

```
SUBSTRING('abcdef',3)          ⇨ 'cdef'
SUBSTRING('abcdef',3,2)        ⇨ 'cd'
```

Следующие выражения эквивалентны:

```
SUBSTRING(str,pos,len)
SUBSTRING(str FROM pos FOR len)
MID(str,pos,len)
```

- SUBSTRING_INDEX(*str*,*delim*,*n*). Возвращает подстроку строки *str*. Если аргумент *n* имеет положительное значение, функция SUBSTRING_INDEX() находит *n*-е появление строки-разделителя *delim* и возвращает всю часть строки, расположенную слева от этого разделителя. Если аргумент *n* имеет отрицательное значение, функция SUBSTRING_INDEX() находит *n*-е появление строки-разделителя *delim*, считая с правого конца *str*, и возвращает всю часть строки, расположенную справа от разделителя. Если строка *delim* в строке *str* не найдена, функция возвращает всю строку *str*. Функция SUBSTRING_INDEX() возвращает значение NULL, если хотя бы один из аргументов равен NULL.

```
SUBSTRING_INDEX('jar-jar','j',-2)          ⇨ 'ar-jar'
SUBSTRING_INDEX('sampadm@localhost','@',1) ⇨ 'sampadm'
SUBSTRING_INDEX('sampadm@localhost','@',-1)⇨ 'localhost'
```

Функция SUBSTRING_INDEX() впервые появилась в MySQL 3.21.15.

- TRIM([[LEADING | TRAILING | BOTH] [*trim_str*] FROM] *str*). Возвращает строку *str*, удаляя расположенные в начале (в конце) экземпляры строки *trim_str*. Если задано ключевое слово LEADING, функция TRIM() удаляет расположенные в начале копии строки *trim_str*. Если задано ключевое слово TRAILING, функция TRIM() удаляет расположенные в конце копии строки *trim_str*. При определении ключевого слова BOTH функция TRIM() удаляет все копии строки *trim_str*, как в начале, так и в конце строки *str*. Если ни одно из следующих ключевых слов не задано по умолчанию, устанавливается слово BOTH. Если строка *trim_str* не задана, функция TRIM() удаляет пробелы.

```
TRIM('^' FROM '^^^xyz^^')          ⇨ 'xyz'
TRIM(LEADING '^' FROM '^^^xyz^^')  ⇨ 'xyz^^'
```

```
TRIM(TRAILING '^' FROM '^^^xyz^^')    ⇨ '^^^xyz'
TRIM(BOTH '^' FROM '^^^xyz^^')        ⇨ 'xyz'
TRIM(BOTH  FROM '   abc    ')         ⇨ 'abc'
TRIM('   abc    ')                    ⇨ 'abc'
```

Функция TRIM() впервые появилась в MySQL 3.21.12.

- UCASE(*str*). Аналогична функции UPPER().
- UPPER(*str*). Функция возвращает строку *str*, записав все символы в верхнем регистре, или значение NULL, если строка *str* имеет значение NULL.

```
UCASE('New York, NY')          ⇨'NEW YORK, NY'
UCASE(NULL)                    ⇨NULL
```

Функции даты и времени

Функции даты и времени считывают аргументы различных типов. В общем, функция, ожидающая аргумент типа DATE, с таким же успехом считает аргумент типа DATETIME или TIMESTAMP, откинув временную часть значения.

Многие функции, приведенные в этом разделе, позволяют интерпретировать числовые аргументы как календарные значения.

```
MONTH('2004-07-25')                                    ⇨7
MONTH(20040725)                                        ⇨7
```

Аналогично, многие функции, обычно возвращающие календарные значения, при использовании их в числовом контексте возвращают числа.

```
CURDATE()                                              ⇨'2002-05-14'
CURDATE() + 0                                          ⇨20020514
```

Ввод недопустимых значений даты и времени для функций даты и времени приводит к возврату непредсказуемых результатов. Поэтому настоятельно рекомендуется внимательно вводить значения аргументов.

- ADDDATE(*date, INTERVAL expr interval*). Аналогична функции DATE_ADD().
- CURDATE(). Возвращает текущую дату в виде строки, записанной в формате "*CCYY-MM-DD*", или в виде числа, представленного в формате "*CCYYMMDD*". Выбор строкового или числового вида результата определяется контекстом, в котором задается функция.

```
CURDATE()                      ⇨'2002-05-14'
CURDATE()+0                    ⇨20020514
```

- CURRENT_DATE. Является аналогом функции CURDATE(). Скобки необязательны.
- CURRENT_TIME. Является аналогом функции CURTIME(). Скобки необязательны.
- CURRENT_TIMESTAMP. Является аналогом функции NOW(). Скобки необязательны.
- CURTIME(). Возвращает текущее время дня в виде строки, записанной в формате "*hh:mm:ss*", или в виде числа, представленного в формате

"*hhmmss*". Выбор строкового или числового вида результата определяется контекстом, в котором задается функция.

```
CURTIME()                              ⇨'09:51:36'
CURTIME()+0                            ⇨95136
```

Функция CURTIME() впервые появилась в MySQL 3.21.12.

- DATE_ADD(*date*,INTERVAL *expr interval*). Считывает значение даты или даты и времени *date*, прибавляет к этому значению заданный временной интервал и возвращает результат. Аргумент *expr* определяет добавляемое к *date* временное значение (или вычитает, если значение *expr* начинается с "-"), а *interval* определяет способ интерпретации интервала. Результат представляет собой значение типа DATE, если аргумент *date* является значением типа DATE и в расчете не применяются временные значения. Если подобные временные значения имеются, результатом будет значение типа DATETIME. Если аргумент *date* имеет недопустимое значение даты, эта функция возвращает значение NULL.

```
DATE_ADD('2002-12-01',INTERVAL 1 YEAR)             ⇨'2003-12-01'
DATE_ADD('2002-12-01',INTERVAL 60 DAY)             ⇨'2003-01-30'
DATE_ADD('2002-12-01',INTERVAL -3 MONTH)           ⇨'2002-09-01'
DATE_ADD('2002-12-01 08:30:00',INTERVAL 12 HOUR)   ⇨'12-01 20:30:00'
```

В табл. В.3 представлены допустимые значения аргумента *interval*, их описания и формат, в котором следует определять значения для каждого типа интервала. Для определения ключевого слова INTERVAL и спецификаторов аргумента *interval* можно использовать как строчные, так и прописные символы.

Таблица В.3. Типы интервалов функции DATE_ADD()

Тип	Описание	Формат значения
SECOND	Секунды	*ss*
MINUTE	Минуты	*mm*
HOUR	Часы	*hh*
DAY	Дни	*DD*
MONTH	Месяцы	*MM*
YEAR	Годы	*YY*
MINUTE_SECOND	Минуты и секунды	*'mm:ss'*
HOUR_MINUTE	Часы и минуты	*'hh:mm'*
HOUR_SECOND	Часы, минуты и секунды	*'hh:mm:ss'*
DAY_HOUR	Дни и часы	*'DD hh'*
DAY_MINUTE	Дни, часы и минуты	*'DD hh:mm'*
DAY_SECOND	Дни, часы, минуты и секунды	*'DD hh:mm:ss'*
YEAR_MONTH	Годы и месяцы	*'YY-MM'*

Добавляемое к значению даты выражение *expr* можно задавать в виде числа или строки. Если это значение содержит нецифровые символы, его следует определять строкой. Разделитель символов можно устанавливать по своему желанию, используя, например, любой символ пунктуации.

```
DATE_ADD('2002-12-01',INTERVAL '2:3' YEAR_MONTH)  ⇨ '2005-03-01'
DATE_ADD('2002-12-01',INTERVAL '2-3' YEAR_MONTH)  ⇨ '2005-03-01'
```

Части значения *expr* соответствуют частям, определяемым спецификатором *interval*. Например, для типа HOUR_SECOND используется формат "*hh:mm:ss*". Заданное в виде строки "15:21" значение аргумента *expr* интерпретируется как "00:15:21", а не как "15:21:00".

```
DATE_ADD('2002-12-01 12:00:00',INTERVAL '15:21'
HOUR_SECOND)          ⇨ '2002-12-01 12:15:21'
```

Если аргумент *interval* представлен значением типа YEAR, MONTH или YEAR_MONTH и в результате день превышает число дней в результирующем месяце, в качестве значения дня выбирается максимальное число дней в этом месяце.

```
DATE_ADD('2002-12-31',INTERVAL 2 MONTH)   ⇨'2003-02-28'
```

Функция DATE_ADD() впервые появилась в MySQL 3.22.4. Кроме того, начиная с MySQL 3.23.4 поддерживается альтернативный синтаксис.

```
'2002-12-31' + INTERVAL 2 MONTH  ⇨'2003-02-28'
INTERVAL 2 MONTH + '2002-12-31'  ⇨'2003-02-28'
```

- DATE_FORMAT(*date*,*format*). Форматирует значение даты или даты и времени *date* в соответствии со строкой формата *format* и возвращает результирующую строку. Функцию DATE_FORMAT() можно использовать для переформатирования значений типа DATE или DATETIME из используемой в MySQL формы в любую другую.

```
DATE_FORMAT('2002-12-01','%M %e %Y')        ⇨'December 1, 2002'
DATE_FORMAT('2002-12-01','The %D of %M')  ⇨'The 1st of December'
```

В табл. B.4 представлены допустимые спецификаторы, которые можно использовать в строке форматирования.

В MySQL 3.23.0 и более поздних версиях перед кодом формата необходимо указывать символ "%". В ранних версиях он допускается, но является необязательным. Символы, определенные в строке формата, но не представленные в таблице, переносятся в строку результата без изменений.

Если используется спецификатор времени для значения типа DATE, временная часть значения обрабатывается как "00:00:00".

```
DATE_FORMAT('2002-12-01','%i')              ⇨ '00'
```

Спецификаторы формата %v, %V, %x и %X были добавлены в MySQL 3.23.8.

Таблица B.4. Описание задания формата DATE_FORMAT()

Спецификатор	Описание
%S, %s	Секунды, записанные двумя цифрами (00, 01, ..., 59)
%i	Минуты, записанные двумя цифрами (00, 01, ..., 59)
%H	Часы, записанные двумя цифрами, 24-часовое время (01, 02, ..., 23)
%h, %I	Часы, записанные двумя цифрами, 12-часовое время (1, 2, ..., 12)
%k	Часы, представленные с помощью чисел, 24-часовое время (0, 1, ..., 23)
%l	Часы, представленные с помощью чисел, 12-часовое время (1, 2, ..., 12)

Спецификатор	Описание
`%T`	Время в 24-часовом формате (*hh:mm:ss*)
`%r`	Время в 12-часовом формате (*hh:mm:ss* AM (до полудня) или *hh:mm:ss* PM (после полудня))
`%p`	AM (до полудня) или PM (после полудня)
`%W`	Название дня недели (Sunday, Monday, ..., Saturday)
`%a`	Название дня недели в сокращенной форме (Sun, Mon, ..., Sat)
`%d`	День месяца, записанный с помощью двух цифр (00, 01, ..., 31)
`%e`	День месяца в числовой форме (1, 2, ..., 31)
`%D`	День месяца с английским суффиксом (1st, 2nd, 3rd, ...)
`%w`	День недели в числовой форме (0 — воскресенье, 1 — понедельник, ..., 6 — суббота)
`%j`	День года, представленный с помощью трех цифр (001, 002, ..., 366)
`%U`	Неделя (00, ..., 53), где воскресенье считается первым днем недели
`%u`	Неделя (00, ..., 53), где понедельник — первый день недели
`%V`	Неделя (01, ..., 53), где воскресенье считается первым днем недели
`%v`	Неделя (01, ..., 53), где понедельник — первый день недели
`%b`	Сокращенное название месяца (Jan, Feb, ..., Dec)
`%M`	Название месяца (January, February, ..., December)
`%m`	Месяц, записанный с помощью двух чисел (01, 02, ..., 12)
`%c`	Месяц в числовой форме (1, 2, ..., 12)
`%Y`	Значение года, записанное в виде числа с четырьмя числами
`%y`	Значение года, записанное в виде числа с четырьмя числами
`%X`	Год для недели, в которой первым днем считается воскресенье, в виде числа с четырьмя числами
`%x`	Год для недели, в которой первым днем считается понедельник, в виде числа с четырьмя числами
`%%`	Константа %

- DATE_SUB(*date*,INTERVAL *expr interval*). Рассчитывает значения даты аналогично функции DATE_ADD() с той лишь разницей, что значение *expr* вычитается из значения даты *date*. Более детально об этом рассказывается в описании функции DATE_ADD().

```
DATE_SUB('2002-12-01',INTERVAL 1 MONTH)   ⇨'2002-11-01'
DATE_SUB('2002-12-01',INTERVAL '13-2' YEAR_MONTH)
                                          ⇨'1989-10-01'
DATE_SUB('2002-12-01 04:53:12',INTERVAL '13-2'
INUTE_SECOND)                             ⇨'2002-12-01 04:40:10'
DATE_SUB('2002-12-01 04:53:12',INTERVAL '13-2' HOUR_MINUTE)
                                          ⇨'2002-11-30 15:51:12'
```

Функция DATE_SUB() впервые появилась в MySQL 3.22.4. Кроме того, начиная с MySQL 3.23.4 появился альтернативный синтаксис.

```
'2002-12-01' - INTERVAL 1 MONTH           ⇨'2002-11-01'
```

- DAYNAME(*date*). Возвращаст строку, содержащую название дня недели для значения даты *date*.

```
DAYNAME('2002-12-01')          ⇨ 'Wednesday'
DAYNAME('1900-12-01')          ⇨ 'Saturday'
```

- DAYOFMONTH(*date*). Возвращает числовое значение дня месяца для значения даты *date* в диапазоне от 1 до 31.

```
DAYOFMONTH('2002-12-01')       ⇨1
DAYOFMONTH('2002-12-25')       ⇨25
```

Функция DAYOFMONTH() впервые появилась в MySQL 3.21.22.

- DAYOFWEEK(*date*). Возвращает числовое значение дня недели для значения даты *date*. В соответствии со стандартом ODBC значения дня недели находятся в диапазоне от 1 (воскресенье) до 7 (суббота). См. также описание функции WEEKDAY().

```
DAYOFWEEK('2002-12-08')        ⇨1
DAYNAME('2002-12-08')          ⇨ 'Sunday'
DAYOFWEEK('2002-12-14')        ⇨7
DAYNAME('2002-12-14')          ⇨ 'Saturday'
```

- DAYOFYEAR(*date*). Возвращает числовое значение дня года для значения даты *date* в диапазоне от 1 до 366.

```
DAYOFYEAR('2002-12-01')            ⇨335
DAYOFYEAR('2004-21-31')            ⇨366
```

- EXTRACT(*interval* FROM *datetime*). Возвращает часть значения даты и времени *datetime*, указанную аргументом *interval*. В качестве последнего может выступать любой спецификатор, допустимый для функции DATE_ADD().

```
EXTRACT(YEAR FROM '2002-12-01 13:42:19')          ⇨2002
EXTRACT(MONTH FROM '2002-12-01 13:42:19')         ⇨12
EXTRACT(DAY FROM '2002-12-01 13:42:19')           ⇨1
EXTRACT(HOUR_MINUTE FROM '2002-12-01 13:42:19')   ⇨1342
EXTRACT(SECOND FROM '2002-12-01 13:42:19')        ⇨19
```

Начиная с MySQL 3.23.39 функцию EXTRACT() можно также использовать в укороченном формате.

```
EXTRACT(YEAR FROM '2004-00-12')           ⇨2004
EXTRACT(MONTH FROM '2004-00-12')          ⇨0
EXTRACT(DAY FROM '2004-00-12')            ⇨12
```

Функция EXTRACT() впервые появилась в MySQL 3.23.0.

Функция EXTRACT() впервые появилась в MySQL 3.23.0.

- FROM_DAYS(*n*). Считывая числовое значение, представляющее число дней с нулевого года (обычно происходит после вызова функции TO_DAYS()), возвращает соответствующую ему дату.

```
TO_DAYS('2009-12-01')          ⇨734107
FROM_DAYS(734107 + 3)          ⇨'2009-12-04'
```

Функцию FROM_DAYS() можно применять только для дат Григорианского календаря (используется с 1582 года).

- FROM_UNIXTIME(*unix_timestamp*);
  FROM_UNIXTIME(*unix_timestamp,format*). Считывая UNIX-значение временной метки *unix_timestamp*, например возвращаемое функцией UNIX_TIMESTAMP(), выдает значение даты и времени в формате *"CCYY-MM-DD hh:mm:ss"* или в виде числа, представленного в формате *"CCYYMMDDhhmmss"*. Выбор формата осуществляется в соответствии с контекстом, в котором задается данная функция. Если задан аргумент *format*, возвращаемое значение форматируется в виде строки так же, как и в функции DATE_FORMAT().

```
UNIX_TIMESTAMP()                    ⇨1021389416
FROM_UNIXTIME(1021389416)           ⇨'2002-05-14 10:16:56'
FROM_UNIXTIME(1021389416,'%Y')      ⇨'2002'
```

- HOUR(*time*). Возвращает значение часа для заданного временного значения *time* в диапазоне от 0 до 23.

```
HOUR('12:31:58')                    ⇨12
HOUR(123158)                        ⇨12
```

  Функция HOUR() впервые появилась в MySQL 3.21.22.

- MINUTE(*time*). Возвращает значение минут для заданного временного значения *time* в диапазоне от 0 до 59.

```
MINUTE("12:31:58")                  ⇨31
MINUTE(123158)                      ⇨31
```

  Функция MINUTE() впервые появилась в MySQL 3.21.22.

- MONTH(*date*). Возвращает числовое значение месяца года для заданного значения даты *date* в диапазоне от 1 до 12.

```
MONTH('2002-12-01')                 ⇨12
MONTH(20021201)                     ⇨12
```

  Функция MONTH() впервые появилась в MySQL 3.21.22.

- MONTHNAME(*date*). Возвращает строку с названием месяца для заданного значения даты *date*.

```
MONTHNAME('2002-12-01')             ⇨'December'
MONTHNAME(20021201)                 ⇨'December'
```

  Функция MONTHNAME() впервые появилась в MySQL 3.21.23.

- NOW(). Возвращает текущую дату и время в виде строки, записанной в формате "*CCYY-MM-DD hh:mm:ss*", или числа, представленного в формате "*CCYYMMDDhhmmss*". Выбор строкового или числового значения осуществляется в соответствии с контекстом, в котором задается эта функция.

```
NOW()                               ⇨'2002-05-14 10:19:20'
NOW()+0                             ⇨20020514101920
```

- PERIOD_ADD(*period,n*). Добавляет *n* месяцев к значению периода *period* и возвращает результат. Возвращаемое значение представлено в формате "*CCYYMM*". Аргумент *period* может иметь формат "*CCYYMM*" или "*YYMM*". Значение даты в этом аргументе не задается.

```
PERIOD_ADD(200202,12)               ⇨200302
PERIOD_ADD(0202,-3)                 ⇨200111
```

- PERIOD_DIFF(*period1*,*period2*). Рассчитывает разницу между представленными аргументами периодами и возвращает число месяцев между ними. Аргументы могут иметь формат "*CCYYMM*" или "*CCYY*". Значение даты в них не задается.

```
PERIOD_DIFF(200302,200202)          ⇨12
PERIOD_DIFF(200111,0202)            ⇨-3
```

- QUARTER(*date*). Возвращает числовое значение квартала года для заданного значения даты *date* в диапазоне от 1 до 4.

```
QUARTER('2002-12-01')               ⇨4
QUARTER('2003-01-01')               ⇨1
```

- SECOND(*time*). Возвращает числовое значение секунд для заданного временного значения *time* в диапазоне от 0 до 59.

```
SECOND('12:31:58')                  ⇨58
SECOND(123158)                      ⇨58
```

  Функция SECOND() впервые появилась в MySQL 3.21.22.

- SEC_TO_TIME(*seconds*). Считывая число секунд *seconds*, возвращает соответствующее временное значение в виде строки формата "*hh:mm:ss*" или числа, представленного в формате "*hhmmss*". Выбор строкового или числового значения осуществляется в зависимости от контекста, в котором задается функция.

```
SEC_TO_TIME(29834)                  ⇨'08:17:14'
SEC_TO_TIME(29834)+0                ⇨81714
```

  Функция SEC_TO_TIME() впервые появилась в MySQL 3.21.5.

- SUBDATE(*date*,INTERVAL *expr interval*). Эта функция является аналогом функции DATE_SUB().
- SYSDATE(). Эта функция является аналогом функции NOW().
- TIME_FORMAT(*time*,*format*). Форматирует временное значение *time* в соответствии со строкой формата *format* и возвращает результирующую строку. Строка формата аналогична используемой в синтаксисе функции DATE_FORMAT(), однако могут задаваться только спецификаторы, относящиеся к значениям времени. При использовании других спецификаторов будет возвращаться значение NULL или 0.

```
TIME_FORMAT('12:51:58','%H %i')     ⇨'12 31'
TIME_FORMAT(123158,'%H %i')         ⇨'12 31'
```

  Функция TIME_FORMAT() впервые появилась в MySQL 3.21.3.

- TIME_TO_SEC(*time*). Считывая значение времени *time*, представляющее прошедшее время, возвращает соответствующее ему общее число секунд. Возвращаемое значение может быть передано функции SEC_TO_TIME() для обратного преобразования.

```
TIME_TO_SEC('08:17:14')             ⇨29834
SEC_TO_TIME(29834)                  ⇨'08:17:14'
```

  Для заданного значения типа DATETIME или TIMESTAMP функция TIME_TO_SEC() игнорирует часть даты.

```
TIME_TO_SEC('2002-03-26 08:17:14')    ⇨29834
```

- TO_DAYS(*date*). Возвращает числовое значение, представляющее собой преобразованное в число дней с нулевого года значение даты *date*. Для преобразования обратно в дату можно задать это значение функции FROM_DAYS().

```
TO_DAYS('2002-12-01')              ⇨731550
FROM_DAYS(731550 - 365)            ⇨'2002-12-01'
```

Для заданного значения типа DATETIME или TIMESTAMP функция TIME_TO_SEC() игнорирует временную часть.

```
TO_DAYS('2002-12-01 12:14:37')        ⇨731550
```

Функцию TO_DAYS() можно применять только для дат Григорианского календаря (используется начиная с 1582 года).

- UNIX_TIMESTAMP(); UNIX_TIMESTAMP(*date*). При вызове без аргументов эта функция возвращает число секунд начиная с эпохи существования UNIX ("1970-01-01 00:00:00" GMT). Если же при вызове задается аргумент даты *date*, эта функция возвращает число секунд между указанной датой и началом эпохи. Аргумент *date* можно определить несколькими способами: в виде строки DATE или DATETIME, значения TIMESTAMP или числа, представляющего локальное время в формате "*CCYYMMDD*" или "*YYMMDD*".

```
UNIX_TIMESTAMP()                   ⇨1021389578
UNIX_TIMESTAMP('2002-12-01')       ⇨1038722400
UNIX_TIMESTAMP(20021201)           ⇨1038722400
```

- WEEK(*date*); WEEK(*date*, *first_day*). При вызове с одним аргументом эта функция возвращает число, определяющее неделю года для значения даты *date* в диапазоне от 0 до 53. Предполагается, что неделя начинается с воскресенья. При вызове с двумя аргументами возвращает то же значение, но аргумент *first_day* указывает день, с которого начинается неделя. Если значение *first_day* равно 0, следовательно, неделя начинается с воскресенья. Если же этот аргумент имеет значение 1, неделя считается начавшейся с понедельника.

```
WEEK('2002-12-08')                 ⇨50
WEEK('2002-12-08',0)               ⇨50
WEEK('2002-12-08',1)               ⇨49
```

Значение WEEK(), равное 0, отражает, что дата была раньше первого воскресенья года (дата была раньше первого понедельника, если первый аргумент равен 1).

```
WEEK('2005-01-01')                 ⇨0
DAYNAME('2005-01-01')              ⇨'Saturday'
WEEK('2006-01-01',1)               ⇨0
DAYNAME('2006-01-01')              ⇨'Sunday'
```

Функция WEEK() впервые появилась в MySQL 3.21.22, а форма этой функции с двумя аргументами – в MySQL 3.22.1.

- WEEKDAY(*date*). Возвращает числовое значение дня недели для заданного значения даты *date*. Значения дней недели могут возвращаться в диапазоне

от 0 (для понедельника) до 6 (для воскресенья). См. также описание функции DAYOFWEEK().

```
WEEKDAY('2002-12-08')          ⇨6
DAYNAME('2002-12-08')          ⇨'Sunday'
WEEKDAY('2002-12-16')          ⇨0
DAYNAME('2002-12-16')          ⇨'Monday'
```

- YEAR(*date*). Возвращает числовое значение года для заданного значения даты *date* в диапазоне от 1000 до 9999.

```
YEAR('2002-12-01')             ⇨2002
YEAR(20021201)                 ⇨2002
```

Функция YEAR() впервые появилась в MySQL 3.21.22.

- YEARWEEK(*date*)
  YEARWEEK(*date,first_day*)

При вызове с одним аргументом возвращает число в формате *CCYYWW*, представляющее год и номер недели в году, соответствующее значению *date*. Номер недели лежит в диапазоне от 0 до 53. Предполагается, что неделя начинается с воскресенья. При вызове с двумя аргументами функция YEARWEEK() возвращает аналогичное значение, но теперь аргумент *first_day* обозначает день, с которого начинается неделя. Если значение этого аргумента равно 0, неделя начинается в воскресенье. Если *first_day* равен 1, неделя начинается в понедельник.

```
YEARWEEK('2006-01-01')         ⇨200601
YEARWEEK('2006-01-01',0)       ⇨200601
YEARWEEK('2006-01-01',1)       ⇨200552
```

Следует отметить, что часть, отвечающая за год, в результате отличалась от года в аргументе *date*. Так происходит в случаях, когда функция WEEK() должна вернуть номер недели, равный 0. Функция YEARWEEK() никогда не возвращает значение, равное 0. Вместо этого она возвращает значение, состоящее из предыдущего года и номера последней недели в этом году.

```
WEEK('2005-01-01')             ⇨0
YEARWEEK('2005-01-01')         ⇨200452
```

Функция YEARWEEK() впервые появилась в СУБД MySQL 3.23.8.

Суммирующие функции

Суммирующие функции называются еще *агрегатными функциями* (*aggregate functions*). Они вычисляют одно результирующее значение на основании группы исходных значений. Результирующее значение рассчитывается только для значений выбранных строк, не равных NULL (исключением является лишь функция COUNT(*), которая подсчитывает число всех строк). Суммирующие функции могут использоваться для подсчета суммы значений целых столбцов или суммирования отдельных значений, сгруппированных по другим столбцам или их комбинациям. Более детально об этом рассказывается в разделе "Получение итоговых результатов" главы 1, "Знакомство с СУБД MySQL и SQL".

В примерах этого раздела используется таблица mytable со столбцом целочисленного типа mycol, который содержит восемь строк со значениями 1, 3, 5, 5, 7, 9, 9 и NULL.

```
mysql> SELECT mycol FROM mytbl;
+-------+
| mycol |
+-------+
|     1 |
|     3 |
|     5 |
|     5 |
|     7 |
|     9 |
|     9 |
|  NULL |
+-------+
```

- AVG(*expr*). Возвращает среднее значение аргумента *expr* для всех непустых значений выбранных строк.

```
SELECT AVG(mycol) FROM mytable        ⇨5.5714
SELECT AVG(mycol)*2 FROM mytable      ⇨11.1429
SELECT AVG(mycol*2) FROM mytable      ⇨11.1429
```

- BIT_AND(*expr*). Возвращает значение побитового оператора И для аргумента *expr* для непустых значений выбранных строк.

```
SELECT BIT_AND(mycol) FROM mytable     ⇨1
```

Функция BIT_AND() впервые появилась в MySQL 3.21.11.

- BIT_OR(*expr*). Возвращает значение побитового оператора ИЛИ для аргумента *expr* для ненулевых значений выбранных строк.

```
SELECT BIT_OR(mycol) FROM mytable      ⇨15
```

Функция BIT_OR() впервые появилась в MySQL 3.21.11.

- COUNT(*expr*), COUNT(*), COUNT(DISTINCT *expr1, expr2,...*). С любым аргументом, кроме символа "*", возвращает общее количество непустых значений в результирующем наборе. Если в функции задается аргумент "*", то результатом является общее количество всех строк в результирующем наборе, в том числе и содержащих значение NULL.

```
SELECT COUNT(mycol) FROM mytable ⇨5
SELECT COUNT(*) FROM mytable     ⇨6
```

Для таблиц типа ISAM и MyISAM функция COUNT(*), задаваемая без предложения WHERE, оптимизирована для быстрого возврата числа записей в указанной в предложении FROM таблице. Если задано сразу несколько таблиц, функция COUNT(*) возвращает произведение количества строк в отдельных таблицах.

```
SELECT COUNT(*) FROM mytable AS m1, mytable AS m2 ⇨64
```

Начиная с версии MySQL 3.23.2 COUNT(DISTINCT) может использоваться для подсчета непустых значений.

```
SELECT COUNT(DISTINCT mycol) FROM mytbl          ⇨5
SELECT COUNT(DISTINCT MOD(mycol,3)) FROM mytbl   ⇨3
```

Для выражения функция COUNT(DISTINCT) вычисляет количество комбинаций непустых значений.

- MAX(*expr*). Возвращает максимальное значение среди всех непустых значений выбранных строк в столбце *expr*. Функцию MAX() можно также использовать со строками. В этом случае она возвращает самое большое лексикографическое значение.

```
SELECT MAX(mycol) FROM mytable          ⇨9
```

- MIN(*expr*). Возвращает минимальное значение среди всех непустых значений выбранных строк в столбце *expr*. Функцию MAX() можно также использовать со строками. В этом случае она возвращает наименьшее лексикографическое значение.

```
SELECT MIN(mycol) FROM mytable          ⇨1
```

- STD(*expr*). Возвращает стандартное отклонение аргумента *expr* для всех непустых значений выбранных строк.

```
SELECT STD(mycol) FROM mytable          ⇨2.7701
```

- STDDEV(*expr*). Эта функция является аналогом функции STD().
- SUM(*expr*). Возвращает сумму всех непустых значений выбранных строк в столбце *expr*.

```
SELECT SUM(my_col) FROM mytable         ⇨89
```

Функции безопасности

Эти функции применяются для поддержки различных операций, предназначенных для обеспечения самых разнообразных функций безопасности. Некоторые из них работают в паре, где одна функция осуществляет шифрование, а другая расшифровку. Такие пары функций в своей работе обычно используют строку, играющую роль ключа или пароля. Для получения исходного значения необходимо расшифровывать значение с использованием того же самого ключа, который использовался при зашифровке этого значения. В противном случае расшифровываемый результат будет бессмысленным. Если необходимо хранить результаты работы функции шифрования, возвращающие двоичную строку, лучше всего использовать столбец, имеющий один из типов BLOB.

- AES_DECRYPT(*str*,*key_str*)

  Расшифровывает строку *str*, полученную в результате вызова функции AES_ENCRYPT(), используя при этом в качестве ключа строку *key_str*. Возвращает значение NULL в случае, если один из аргументов имеет значение NULL.

```
AES_DECRYPT(AES_ENCRYPT('secret','scramble'), 'scramble')
    ⇨'secret'
```

  Функция AES_DECRYPT() впервые появилась в MySQL 4.0.2.

- AES_ENCRYPT(*str*,*key_str*)

  Зашифровывает строку *str*, используя при этом в качестве ключа строку *key_str*, улучшенный стандарт шифрования (Advanced Encryption Standard, AES) и ключ длиной 128 бит. Возвращает значение NULL в случае, если один из аргументов имеет значение NULL. Эта строка может быть расшифрована с использованием того же ключа с помощью функции AES_DECRYPT().

Функция AES_ ENCRYPT() впервые появилась в MySQL 4.0.2.

- DECODE(*str*,*key_str*). Считывая зашифрованную строку *str*, созданную ранее посредством вызова функции ENCODE(), расшифровывает ее с помощью строки пароля *key_str* и возвращает результат расшифровки. Возвращает результирующую строку или значение "NULL", если *str* имеет значение NULL.

```
DECODE(ENCODE('secret','scramble'),'scramble') ⇨'secret'
```

- DES_DECRYPT(*str*)
  DES_DECRYPT(*str*,*key_str*)

  Расшифровывает строку *str*, которая шифруется функцией DES_ENCRYPT(). Если поддержка SSL не активизирована или расшифровать строку не удается, функция DES_DECRYPT() возвращает NULL.

  Заданный аргумент *key_str* используется как ключ расшифровки. Если аргумент *key_str* не задан, функция DES_DECRYPT() для расшифровки строки использует ключ из ключевого DES-файла сервера. Номер ключа определяется битами 0–6 первого байта зашифрованной строки. Размещение ключевого файла задается во время запуска сервера с помощью параметра `--des-key-file`. При использовании разных ключей для шифровки и расшифровки строки будет получен бессмысленный результат.

  Если строка *str* не похожа на зашифрованную строку, функция DES_DECRYPT() возвращает исходную строку. (Это происходит, например, если в первом байте не установлен бит 7.)

  Использование формы функции DES_DECRYPT() с одним аргументом требует привилегий суперпользователя.

  Функция DES_DECRYPT() впервые появилась в MySQL 4.0.1.

- DES_ENCRYPT(*str*)
  DES_ENCRYPT(*str*,*key_num*)
  DES_ENCRYPT(*str*,*key_str*)

  Выполняет DES-шифрование строки *str* и возвращает зашифрованную строку в виде двоичной строки. Зашифрованная строка может быть зашифрована с помощью функции DES_DECRYPT(). Если поддержка SSL не активизирована или расшифровать строку не удается, функция DES_DECRYPT() возвращает NULL.

  Заданный аргумент *key_str* используется как ключ расшифровки. При этом он должен иметь значение от 0 до 9, указывая на запись в ключевом DES-файле сервера. В этом случае ключ шифровки может быть получен из этой записи. Если аргумент *key_str* или *key_num* не задан, для шифрования берется первый ключ из ключевого DES-файла. (Это необязательно, как и при определении значения *key_num* равным 0.)

  Первый байт результирующей строки свидетельствует о том, как была зашифрована строка. Этот байт имеет 7-й бит, а биты 0–6 отражают номер ключа. Числами от 0 до 9 определяется, какой ключ из ключевого DES-файла был использован при шифровании строки; при использовании аргумента *key_str* это может быть значение 127.

Например, при использовании для шифрования строки ключа 3 первый байт результата будет равен 131 (т.е. 128+3). При шифровке с помощью значения *key_str* первый байт будет 255 (т.е. 128+127).

Для шифрования с помощью номера ключа сервер считывает ключевой DES-файл для определения соответствующей ключевой строки. Размещение ключевого файла задается во время запуска сервера с параметром `--des-key-file`. Ключевой файл содержит строки в следующем формате:

key_num key_str

Значение *key_num* должно иметь значение, равное числу в диапазоне от 0 до 9, а значение *key_str* представляет соответствующий ключ шифровки. Значения *key_num* и *key_str* должны разделяться как минимум одним пробелом. Строки в ключевом файле могут располагаться в любом порядке.

В отличие от функции DES_DECRYPT(), функция DES_ENCRYPT() не требует привилегий суперпользователя для использования ключей из ключевого DES-файла. (Шифровать информацию на основании ключевого файла может любой желающий, но расшифровать ее могут только привилегированные пользователи.)

Функция DES_ENCRYPT() впервые появилась в MySQL 4.0.1.

- ENCODE(*str,key_str*). Зашифровывает строку *str* с помощью строки пароля *key_str* и возвращает результат в виде двоичной строки. Строка может быть расшифрована функцией DECODE() с помощью ключевой строки.
- ENCRYPT(*str*); ENCRYPT(*str,salt*). Зашифровывает строку *str* и возвращает результирующую строку. Это шифрование является односторонним. Необязательный аргумент *salt* должен представлять собой строку из двух символов. (Начиная с версии MySQL 3.22.16 аргумент *salt* может включать больше двух символов.) При определенном значении *salt* зашифрованный результат для *str* будет одинаковым все время. Если аргумент *salt* не задается, идентичные вызовы функции ENCRYPT() приводят к различным результатам.

```
ENCRYPT('secret','AB')        ⇨'ABS5SGh1EL6bk'
ENCRYPT('secret','AB')        ⇨'ABS5SGh1EL6bk'
ENCRYPT('secret')             ⇨'9u0h1zMKCx9N2'
ENCRYPT('secret')             ⇨'avGJcOP2vakBE'
```

Функция ENCRYPT() вызывает UNIX-функцию crypt(), соответственно от этой функции зависит и результат ENCRYPT(). Во-первых, если функция crypt() отсутствует в системе, ENCRYPT() всегда возвращает значение NULL. Во-вторых, результат функции ENCRYPT() зависит от работы crypt() в каждой конкретной системе. Так, например, в некоторых системах crypt() шифрует только первые восемь символов строки.

- MD5(*str*). Вычисляет 128-битовую контрольную сумму из строки *str* по алгоритму MD5 (Message-Digest), разработанному компанией RSA Data Security, Inc. Возвращаемое значение является строкой, состоящей из 32 шестнадцатеричных цифр.

```
MD5('secret')    ⇨'5ebe2294ecd0e0f08eab7690d2a6ee69'
```

Функция MD5() впервые появилась в MySQL 3.23.2. См. также функцию SHA1().

- PASSWORD(*str*). Считывая строку *str*, возвращает зашифрованную строку пароля формы, используемую в таблицах разрешений MySQL. Это шифрование является односторонним.

```
PASSWORD('secret')        ⇨'428567f408994404'
```

Важно отметить, что функция PASSWORD() *не* использует алгоритм, применяемый в UNIX, для шифровки паролей учетных записей. Для подобного шифрования используется функция ENCRYPT().

- SHA(*str*)

Эта функция является аналогом функции SHA1().

- SHA1(*str*)

Эта функция вычисляет 160-битовую контрольную сумму из строки *str* с использованием алгоритма Secure Hash. Возвращается строка, состоящая из 40 шестнадцатеричных цифр, или пустое значение, если аргумент равен "NULL".

```
SHA1('secret')        ⇨'e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4'
```

SHA1() впервые появилась в MySQL 4.0.2. См. также функцию MD5().

Другие функции

Описанные в этом разделе функции нельзя отнести ни к одной из вышеперечисленных категорий.

- BENCHMARK(*n*, *expr*). Рассчитывает выражение *expr* число раз, определенное аргументом *n*. Особенность функции BENCHMARK() состоит в том, что она предназначена для использования в клиентских программах mysql. Она всегда возвращает значение 0, которое никакого интереса не представляет. Основной интерес представляет время, выводимое программой mysql после отображения результатов запроса.

```
mysql>SELECT BENCHMARK(100000, PASSWORD('secret'));
+----------------------------------------+
| BENCHMARK(100000, PASSWORD('secret')) |
+----------------------------------------+
|                                      0 |
+----------------------------------------+
1 row in set (2.35 sec)
```

Это время является только приблизительным индикатором скорости вычислений, поскольку отражает время обработки для клиента, а не отдельное время обработки запроса процессором сервера. На это значение времени могут влиять такие факторы: скорость загрузки на сервере, находится ли сервер в активном состоянии или загружен в момент получения запроса и т.п. Эту функцию желательно запускать несколько раз, чтобы оценить постоянство возвращаемого значения.

Функция BENCHMARK() впервые появилась в MySQL 3.22.15.

- BIT_COUNT(*n*). Возвращает число определенных аргументом разрядов, которое обрабатывается как значение типа BIGINT (64-разрядное целое значение).

```
BIT_COUNT(0)                ⇨0
BIT_COUNT(1)                ⇨1
BIT_COUNT(2)                ⇨1
BIT_COUNT(7)                ⇨3
BIT_COUNT(-1)               ⇨64
BIT_COUNT(NULL)             ⇨NULL
```

- `BIT_LENGTH(str)`

  Эта функция возвращает длину строки *str* в битах или пустое значение, если аргумент равен "NULL".

```
BIT_LENGTH('abc')           ⇨24
BIT_LENGTH('a long string') ⇨104
```

  Эта функция BIT_LENGTH() была добавлена в MySQL 4.0.2 для обеспечения ODBC-совместимости.

- CONNECTION_ID()

  Возвращает идентификатор соединения текущего соединения. Это идентификатор процесса, который сервер ассоциирует с соединением клиента.

```
CONNECTION_ID()             ⇨10146
```

  Функция CONNECTION_ID() впервые появилась в MySQL 3.23.14.

- DATABASE(). Возвращает строку с именем текущей базы данных или пустую строку, если текущая база данных не задана.

```
DATABASE()                  ⇨'sampdb'
```

- FOUND_ROWS()

  Возвращает количество строк, которые должен вернуть предыдущий оператор SELECT без выражения LIMIT. Например, следующий оператор возвращает максимум 10 строк.

```
mysql> SELECT * FROM mytbl LIMIT 10;
```

  Для того чтобы узнать, сколько строк должен вернуть оператор, если не задавать предложение LIMIT, достаточно сделать следующее:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM mytbl LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

  Функция FOUND_ROWS() была впервые добавлена в MySQL 4.0.0.

- GET_LOCK(*str*,*timeout*). Функция GET_LOCK() используется вместе с функциями RELEASE_LOCK() и IS_FREE_LOCK() для установки и снятия блокировки. Эти две функции можно применять при создании приложений, взаимодействующих друг с другом по имени блокировки.

  Функция GET_LOCK() вызывается вместе с именем блокировки, указанным в строке *str*, и значением периода, задаваемым аргументом *timeout* в секундах. Эта функция возвращает значение 1, если блокировка установлена в течении заданного периода, или значение 0, если блокировка установлена не была. Возникновение ошибки в процессе работы функции приводит к возврату значения NULL. Значение *timeout* определяет, сколько времени необходимо ждать до попытки установки блокировки (но не период времени, по достижении которого блокировка снимается). После установки блокировка остается постоянно активизированной до момента разблокировки пользователем.

Например, следующая функция устанавливает блокировку с именем 'Nellie', ожидая 10 секунд:

```
GET_LOCK('Nellie',10)
```

Блокировка применима только к строке. Невозможно заблокировать базу данных, таблицу, какую-либо строку или столбец в пределах таблицы. Другими словами, блокировка не запретит другому клиенту вносить изменения в таблицы базы данных. Именно поэтому блокировка с помощью функции GET_LOCK() является только поверхностной – другие клиенты могут легко определить, была ли установлена каким-либо пользователем блокировка.

Клиент, установивший блокировку на имя, не позволяет другим клиентам заблокировать это же имя (или подобные попытки других потоков многопотокового клиента, который поддерживает несколько соединений с сервером.) Предположим, например, что клиент 1 блокирует строку "Nellie". Если клиент 2 пытается блокировать эту же строку, она окажется недоступной до тех пор, пока клиент 1 не снимет блокировку или пока не истечет время заданного им периода. В первом случае клиент 2 впоследствии сможет успешно установить блокировку, а в последнем случае его постигнет неудача.

Так как два клиента не могут блокировать данную строку одновременно, приложения, обращающиеся к данным по именам, могут использовать состояние блокировки этого имени как индикатор доступности или недоступности строки. Например, можно создать имя блокировки, основанное на значении уникального ключа для строки в таблице, что позволит выполнять совместную ее блокировку.

Блокировка снимается только вызовом функции RELEASE_LOCK() с указанием имени блокировки:

```
RELEASE_LOCK('Nellie')
```

Функция RELEASE_LOCK() возвращает значение 1, если блокировка снята успешно, или значение 0, если блокировка установлена другим соединением (снимаются только собственные блокировки). Если блокировки с заданным именем не существует, функция возвращает значение NULL.

Разрыв клиентского соединения или повторный вызов функции GET_LOCK() приводит к автоматическому снятию блокировки (один клиент в состоянии заблокировать только одну строку в определенный момент времени). В последнем случае старая блокировка снимается до установления новой, даже если имя новой блокировки подобно имени старой. Блокировка также снимается при разрыве соединения клиента и сервера. Заметим, что если этот клиент с очень длительным соединением разрывает соединение по тайм-ауту из-за длительного периода неактивности, любая блокировка, установленная данным клиентом, будет снята.

Функцию GETLOCK(*str*,0) можно также использовать для определения, установлена ли в текущий момент времени блокировка на строку *str*. (Если эта строка не заблокирована, то вызов данной функции приведет к установке блокировки. В таком случае следует обязательно вызвать функцию RELEASE_LOCK().)

Для проверки состояния блокировки имени можно вызвать функцию IS_FREE_LOCK(*str*), возвращающую 1, если имя доступно (не используется в блокировках); 0, если имя заблокировано, или NULL, если возникла ошибка. Все три функции возвращают значение NULL, если аргумент имени блокировки равен "NULL".

- INET_ATON(*str*)

  Для заданного IP-адреса, представленного в традиционном виде строки с разделенными точками тетрадами, возвращает целочисленное представление адреса или значение "NULL", если аргумент не является допустимым IP-адресом.

```
INET_ATON('64.28.67.70')          ⇨1075594054
INET_ATON('255.255.255.255')      ⇨4294967295
INET_ATON('256.255.255.255')      ⇨NULL
INET_ATON('www.mysql.com')        ⇨NULL
```

  Функция INET_ATON() появилась в MySQL 3.23.15.

- INET_NTOA(*n*)

  Для заданного целочисленного представления IP-адреса, представленного в виде строки с разделенными точками тетрадами, возвращает соответствующее представление IP-адреса в виде строки с разделенными точками тетрадами или значение "NULL", если аргумент не является допустимым IP-адресом.

```
INET_NTOA(1075594054)         ⇨'64.28.67.70'
INET_NTOA(2130706433)         ⇨'127.0.0.1'
```

  Функция INET_NTOA() появилась в MySQL 3.23.15.

- IS_FREE_LOCK(*str*)

  Проверяет состояние блокировки, заданной строкой *str*. Функция IS_FREE_LOCK() используется совместно с функцией GET_LOCK(). Подробнее о блокировках можно узнать в описании функции GET_LOCK().

  Функция IS_FREE_LOCK() появилась в MySQL 4.0.2.

- LAST_INSERT_ID(); LAST_INSERT_ID(expr). Первая форма этой функции (без аргумента) возвращает значение AUTO_INCREMENT, которое было сгенерировано последним во время текущего сеанса работы сервера, или значение 0, если такое значение сгенерировано не было. Вторая форма функции LAST_INSERT_ID() предназначена для использования в операторе UPDATE. Результат ее обрабатывается таким же образом, как и автоматически созданное значение.

  Более детально об этой функции рассказывается в главе 2, "Работа с данными в MySQL". Результирующее значение обеих форм функции LAST_INSERT_ID() сохраняется сервером для каждого соединения в отдельности и не может быть изменено другими клиентами, даже задающими автоматическое создание новых значений.

  Форма функции LAST_INSERT_ID() с аргументом впервые появилась в MySQL 3.22.9.

- LOAD_FILE(*file_name*). Считывает файл *file_name* и возвращает его содержимое в виде строки. Файл должен располагаться на сервере и опре-

деляться с помощью абсолютного (полного) имени пути. Кроме того, этот файл должен быть открыт для общего доступа, поскольку попытка чтения защищенного файла обязательно завершится неудачей. Размещение файла на сервере вынуждает использующего функцию пользователя иметь привилегию FILE. Если хотя бы одно из этих условий не выполняется, функция LOAD_FILE() возвращает значение NULL.

Функция LOAD_FILE() впервые появилась в MySQL 3.23.0.

- MASTER_POS_WAIT(*log_file,pos*) .

  Эта функция используется для тестирования главного сервера репликации. Он блокирует главный сервер до тех пор, пока сервер не достигнет заданного места в регистрационном файле. Если подчиненный сервер уже достиг этой позиции, функция возвращает значение, если подчиненный сервер не работает, главный блокируется до тех пор, пока подчиненный не будет запущен и не достигнет заданного положения.

  Функция MASTER_POS_WAIT() возвращает количество событий регистрационного файла, которые он должен ожидать до тех пор, пока подчиненный не достигнет указанного положения, или "NULL", если возникает ошибка или информация главного сервера не была инициализирована.

  Функция MASTER_POS_WAIT() была добавлена в MySQL 3.23.32.

- RELEASE_LOCK(*str*). Функция RELEASE_LOCK() используется вместе с функцией GET_LOCK(). Более детально о ней рассказывается в описании функции GET_LOCK().

  Функция RELEASE_LOCK() впервые появилась в MySQL 3.21.7.

- SESSION_USER(). Эта функция является аналогом функции USER().
- SYSTEM_USER(). Эта функция является аналогом функции USER().
- USER(). Возвращает строку, представляющую текущее пользовательское имя клиента. В MySQL 3.22.1 и более поздних версиях эта строка имеет форму "*user@host*", где *user* — имя пользователя, а *host* — имя компьютера, с которого клиент подключается к серверу.

```
USER()                          ⇨'sampadm@localhost'
SUBSTRING_INDEX(USER(),'@',1)   ⇨'sampadm'
SUBSTRING_INDEX(USER(),'@',-1)  ⇨'localhost'
```

  До MySQL 3.22.1 значение, возвращаемое из USER(), состоит только из имени пользователя.

- VERSION(). Возвращает строку, описывающую версию сервера.

```
VERSION()                ⇨'4.0.3-beta-log'
```

  Значение состоит из номера версии сервера с суффиксами, которые могут включать одно их следующих значений.

  - -alpha, -beta или -gamma. Отражают стабильность версии MySQL.
  - -debug. Означает, что сервер работает в отладочном режиме.
  - -demo. Отражает тот факт, что сервер работает в демонстрационном режиме (использовался включительно до MySQL 3.23.30).

- `-embedded`. Обозначает встроенный сервер, `libmysqld`.
- `-log`. Обозначает включение протоколирования.
- `-max`. Означает, что сервер был оттранслирован с дополнительными возможностями.
- `-nt`. Сервер построен для работы под управлением ОС Windows, использующих NT-технологии.

Г

Синтаксис SQL

В этом приложении...

Операторы языка SQL
Переменные языка SQL
Синтаксис комментариев

В этом приложении описывается следующее.

- Все используемые в MySQL операторы языка SQL.
- Процесс создания и использования в операторах SQL пользовательских переменных.
- Синтаксис написания комментариев в SQL-коде. Комментарии, как правило, применяются для ввода описательного текста, пропускаемого сервером при выполнении, а также для сокрытия ключевых слов MySQL (эти слова исполняются сервером MySQL, но игнорируются другими серверами баз данных).

Поскольку в настоящее время развитие MySQL все еще продолжается, код SQL также постоянно совершенствуется. Чтобы узнать о новых возможностях MySQL, обратитесь к постоянно обновляемому руководству по MySQL, доступному по адресу `http://www.mysql.com/`.

В представленном ниже описании синтаксиса используются следующие условные обозначения.

- Необязательная информация заключается в квадратные скобки ([]). Если в квадратные скобки заключен список, значит, можно выбрать один из его элементов.
- Если список заключен в фигурные скобки ({}), значит, обязательно должен быть выбран один из его элементов.

- Вертикальная черта (|) разделяет альтернативные элементы списка.
- Многоточие (. . .) определяет, что предшествующая ему последовательность может быть продолжена.
- *n* обозначает целое число.
- '*string*' отражает строковое значение. Значение, заключенное в одинарные кавычки, например '*file_name*' или '*pattern*', обозначает спецификации имен файлов или шаблонов.

Описанные далее операторы могут использоваться с MySQL версии 3.22.0 (если не указано иное).

Операторы языка SQL

В этом разделе рассматривается синтаксис и описываются используемые в MySQL операторы SQL. Следует заметить, что отсутствие достаточных привилегий для запуска оператора приводит к его безуспешному завершению. Так, например, если пользователь, не обладающий правами доступа к базе данных *db_name*, попытается исполнить оператор USE *db_name*, эта попытка закончится провалом.

ALTER DATABASE

```
ALTER DATABASE db_name action_list
```

Этот оператор изменяет глобальные характеристики базы данных. Параметр *action_list* определяет одно или несколько действий, разделенных запятыми. Однако в настоящий момент возможно только одно действие:

```
[DEFAULT] CHARACTER SET charset
```

charset может быть именем кодировки или значением DEFAULT для использования по умолчанию текущей кодировки сервера.

Для работы с оператором ALTER DATABASE требуется привилегия ALTER.

Этот оператор был добавлен в MySQL 4.1.

ALTER TABLE

```
ALTER TABLE [IGNORE] TABLE tbl_name action_list
```

Оператор ALTER TABLE позволяет переименовывать таблицы и изменять их структуру. Для его выполнения задайте имя таблицы *tbl_name* и определите операции, которые необходимо выполнить над таблицей. Ключевое слово IGNORE используется в тех случаях, когда в новой таблице дублируются значения уникальных ключей. Если это ключевое слово не задано, оператор ALTER TABLE не выполняется. Если это слово имеется, строки с одинаковыми значениями для уникальных ключей удаляются.

За исключением операции переименования, оператор ALTER TABLE создает из исходной таблицы новую таблицу, в которой будут учтены все внесенные изменения. При возникновении ошибки во время выполнения процедуры новая таблица удаляется, а старая остается неизменной. Если операция завершается

успешно, исходная таблица удаляется и заменяется новой таблицей. Во время операции другие клиенты имеют возможность читать из исходной таблицы. Действия клиентов, которые пытаются одновременно обновить такую таблицу, блокируются после того, как оператор ALTER TABLE завершает работу, а новая таблица обновляется.

Предложение *action_list* определяет одну или несколько разделяемых запятыми операций. Все эти операции выполняются в порядке очередности. В качестве операций можно задавать следующие.

- ADD [COLUMN] *col_declaration* [FIRST | AFTER *col_name*)]. Добавляет столбец в таблицу. *col_declaration* — это описание столбца, которое имеет такой же формат, что и в операторе CREATE TABLE. При задании ключевого слова FIRST столбец вставляется первым. Если же задается предложение AFTER *col_name*, новый столбец вставляется после столбца *col_name*. Если позиция столбца не определена, он добавляется последним.

  ```
  ALTER TABLE t ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY
  KEY;
  ALTER TABLE t ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY
  KEY FIRST;
  ALTER TABLE t ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY
  KEY AFTER suffix;
  ```

- ADD [COLUMN] (*create_definition*, ...). Добавляет столбцы или индексы в таблицу. Параметр create_definition является определением столбца или индекса в том же формате, что и CREATE TABLE.

  Этот синтаксис был добавлен в MYSQL 3.23.11.

- ADD [CONSTRAINT *name*] FOREIGN KEY [*index_name*] (index_columns) *reference_definition*. Добавляет определение внешнего ключа. Это действие поддерживается только для таблиц типа InnoDB. Внешний ключ опирается на столбцы, перечисленные в параметре *index_columns*, который представляет собой список из одного и нескольких столбцов таблицы, разделенных запятыми. Любое ограничение, CONSTRAINT или *index_name*, если таковое задано, будет проигнорировано. Параметр *reference_definition* определяет отношение внешнего ключа к порождающей таблице. Синтаксис описывается в описании оператора CREATE TABLE. ADD FOREIGN KEY был добавлен в MySQL 3.23.50. (Существует соответствующее предложение DROP FOREIGN KEY, и в настоящее время проводится его синтаксический анализ, но само оно не отрабатывает.)

  ```
  ALTER TABLE child ADD FOREIGN KEY (par_id) REFERENCES parent
  (par_id) ON DELETE CASCADE;
  ```

- ADD FULLTEXT [KEY | INDEX] [*index_name*] (*index_columns*). Добавление индекса FULLTEXT для таблицы типа MyISAM. Индекс опирается на столбцы, перечисленные в столбцах index_columns, которые представляют собой один или несколько столбцов, разделенных запятыми. Оператор ADD FULLTEXT был добавлен в MySQL 3.23.23.

  ```
  ALTER TABLE poetry ADD FULLTEXT (author,title,stanza);
  ```

- ADD INDEX [*index_name*] (*index_columns*). Добавляет индекс в таблицу. Индекс создается по столбцам, которые перечислены в предложении

index_columns. Если указано несколько столбцов, их имена следует разделять запятыми. Для столбцов типа CHAR и VARCHAR можно создавать индекс по префиксу столбца. В этом случае для индексирования первых *n* символов значений столбца применяется синтаксис *col_name(n)*. Для столбцов типа BLOB и TEXT обязательно нужно определить значение префикса. Нельзя индексировать по всему столбцу. Если имя индекса *index_name* не определено, оно создается автоматически, основываясь на имени первого индексируемого столбца.

- ADD PRIMARY KEY (*index_columns*). Добавляет первичный ключ с именем PRIMARY по определенным столбцам. Столбцы *index_columns* определяются так же, как и для предложения ADD INDEX. Если первичный ключ уже существует, результатом выполнения этой операции будет ошибка.

```
ALTER TABLE president ADD PRIMARY KEY (last_name, first_name);
```

- ADD UNIQUE [*index_name*] (*index_columns*). Добавляет индекс с уникальным значением по таблице *tbl_name*. Параметры *index_name* и *index_columns* определяются так же, как и для предложения ADD INDEX.

```
ALTER TABLE absence ADD UNIQUE id_date (student_id, date);
```

- ALTER [COLUMN] *col_name* {SET DEFAULT *value* | DROP DEFAULT}. Изменяет установленное по умолчанию значение столбца на новое значение либо удаляет существующие значения по умолчанию. В последнем случае присваивается новое значение по умолчанию в соответствии с оператором CREATE TABLE.

```
ALTER TABLE event ALTER type SET DEFAULT 'Q';
ALTER TABLE event ALTER type DROP DEFAULT;
```

- CHANGE [COLUMN] *col_name col_declaration* [FIRST | AFTER *col_name*]. Изменяет название и описание столбца. *col_name* — это текущее имя изменяемого столбца, а *col_declaration* — его новое описание. Предложение *col_decaration* имеет такой же формат, какой используется для оператора CREATE TABLE. Обратите внимание: в описания столбца включено его новое имя, поэтому если необходимо оставить это имя неизменным, его обязательно нужно ввести дважды. Параметр FIRST или AFTER может использоваться начиная с MySQL 4.0.1 и имеет тот же эффект, что и ADD COLUMN.

```
ALTER TABLE student CHANGE name name VARCHAR(40);
ALTER TABLE student CHANGE name student_name CHAR(30) NOT NULL;
```

- DISABLE KEYS. Для таблицы MyISAM отменяет модификацию неуникальных индексов, которые обычно появляются при изменении таблицы. Действие ENABLE KEYS может использоваться для возобновления обновления индекса. Действие DISABLE KEYS было добавлено в MySQL 4.0.

```
ALTER TABLE score DISABLE KEYS;
```

- DROP [COLUMN] *col_name* [RESTRICT | CASCADE]. Удаляет определенный столбец из таблицы. Если удаляемый столбец входит в состав индексов, он одновременно будет удален и из индексов. Удаление всех столбцов индекса приводит к удалению и самого индекса.

```
ALTER TABLE president DROP suffix;
```

Ключевые слова RESTRICT и CASCADE пока не имеют никакого эффекта. Они анализируются только для обеспечения переносимости из других баз данных, но игнорируются.

- DROP INDEX *index_name*. Удаляет определенный индекс.

  ```
  ALTER TABLE member DROP INDEX name;
  ```

- DROP PRIMARY KEY. Удаляет первичный ключ из таблицы. Если же в таблице отсутствует уникальный индекс, созданный как ключ PRIMARY KEY, но при этом имеется больше одного индекса UNIQUE, удаляется первый из этих индексов.

  ```
  ALTER TABLE president DROP PRIMARY KEY;
  ```

- ENABLE KEYS. Для таблицы MyISAM возобновляет модификацию неуникальных индексов, которая была отключена оператором DISABLE KEYS. Действие ENABLE KEYS было добавлено в MySQL 4.0.

  ```
  ALTER TABLE score ENABLE KEYS;
  ```

- MODIFY [COLUMN] *col_declaration* [FIRST | AFTER *col_name*]. Изменяет описание столбца. Описание столбца *col_declaration* дается в таком же формате, как и для оператора CREATE TABLE. Оно начинается с нового имени столбца. Операция MODIFY впервые появилась в MySQL версии 3.22.16. Параметр FIRST или AFTER может использоваться начиная с MySQL 4.0.1 и имеет тот же эффект, что и для ADD COLUMN.

  ```
  ALTER TABLE student MODIFY name VARCHAR(40) DEFAULT '' NO NULL;
  ```

- ORDER BY *col_list*. Отсортировать строки таблицы в соответствии со списком, перечисленным в *col_list*, который представляет собой список столбцов, состоящий из одного или нескольких столбцов, разделенных запятыми. По умолчанию сортировка является возрастающей. За именем столбца может следовать ключевое слово ASC или DESC, тем самым определяя возрастающий или убывающий порядок сортировки. Сортировка таблицы таким образом помогает повысить производительность запросов, которые делают выборку строк в том же порядке. Это может очень пригодиться для таблиц, которые не будут модифицироваться в дальнейшем, так как в противном случае строки не сохранят порядок следования, если после операции ORDER BY будет внесены изменения. Этот параметр был добавлен в MySQL 3.23.28.

  ```
  ALTER TABLE score ORDER BY event_id, student_id;
  ```

- RENAME [AS] *new_tbl_name*. Переименовывает таблицу *tbl_name* в *new_tbl_name*.

  ```
  ALTER TABLE president RENAME prez;
  ```

  До MySQL 3.23.17 между RENAME и именем новой таблицы не задавалось никакого ключевого слова. Начиная с 3.23.17 появилось необязательное ключевое слово TO, а начиная с 3.23.23 к нему прибавилось необязательное ключевое слово AS.

- *table_options*. Определяет параметры сортировки таблицы, которые можно задавать в части *table_options* оператора CREATE TABLE.

  ```
  ALTER TABLE score TYPE = MYISAM CHECKSUM = 1;
  ```

```
ALTER TABLE sayings CHARACTER SET utf8;
```

Все ограничения на применение определенных параметров, зависящие от версии, можно найти в описании оператора CREATE TABLE.

ANALYZE TABLE

```
ANALYZE {TABLE | TABLES} tbl_name [, tbl_name] ...
```

Этот оператор предназначен для анализа указанных таблиц, но сохраняет при этом распределение ключевых значений, заданных в каждом индексе таблицы. Он действует для таблиц типа MyISAM и BDB и требует наличия привилегий SELECT и INSERT для каждой таблицы. После проведенного анализа столбец Cardinality вывода из SHOW INDEX отражает количество явных значений в индексе. Информация, полученная в результате анализа, может использоваться оптимизатором при последующих запросах, позволяя запросам с определенными типами объединений выполняться быстрее.

Проведение анализа таблицы требует блокировки чтения, которая предотвращает обновление этой таблицы во время выполнения данной операции. При запуске оператора ANALYZE TABLE по таблице, которая уже была проанализирована и с тех пор не изменялась, анализ производиться не будет.

Оператор ANALYZE TABLE делает вывод в формате, описанном для оператора CHECK TABLE.

Оператор ANALYZE TABLE был добавлен в MySQL 3.23.14.

BACKUP TABLE

```
BACKUP {TABLE | TABLES} tbl_name [, tbl_name] ... TO 'dir_name'
```

Делает резервную копию таблицы или таблиц в каталог, обозначенный *'dir_name'* и представляющий собой имя каталога на серверном узле, где будет записан резервный файл. Оператор BACKUP TABLE работает только для таблиц MyISAM и требует привилегий SELECT и FILE. Он копирует структуру таблицы и файлы данных (файлы .frm и .MYD), которые являются минимумом, необходимым для восстановления таблицы. Индексные файлы не копируются, так как они могут легко воссоздаваться по необходимости (с помощью команды RESTORE TABLE), опираясь на структуру таблицы и файлы данных.

Таблицы блокируются от чтения отдельно по мере их резервного копирования. При резервном копировании нескольких таблиц вполне возможна ситуация, когда во время копирования одних таблиц другие таблицы будут модифицированы. Если необходимо обеспечить копирование всех таблиц как единой группы с целостным содержимым, которое было на момент начала выполнения операторов BACKUP TABLE, нужно воспользоваться операторами LOCK TABLE, чтобы сначала заблокировать их, и UNLOCK TABLE — для их разблокирования после резервного копирования. Конечно, это приведет к недоступности таблиц на более продолжительный период времени.

Файлами, созданными оператором BACKUP TABLE, будет владеть учетная запись, под управлением которой работает сервер. Любая более ранняя резервная копия будет перезаписана.

Оператор BACKUP TABLE был добавлен в MySQL 3.23.25.

Чтобы создать резервную копию таблицы t в каталоге /var/mysql/bkup, необходимо задать команду:

```
BACKUP TABLE t TO '/var/mysql/bkup';
```

BEGIN

```
BEGIN [WORK]
```

Начало транзакции с отключением режима автовыполнения до следующего оператора COMMIT или ROLLBACK. Операторы, отрабатывающие при отключенном режиме автовыполнения, выполняются или откатываются единым блоком.

После того как транзакция была выполнена или был сделан ее откат, режим автовыполнения восстанавливается, как это было до оператора BEGIN. Для манипулирования режимом автовыполнения используйте оператор SET AUTOCOMMIT. (Подробнее о нем — в описании оператора SET.)

Появление оператора BEGIN посередине транзакции приводит к завершению транзакции.

Оператор BEGIN был добавлен в MySQL 3.23.17. Оператор BEGIN WORK был добавлен как его синоним в MySQL 3.23.19.

CHANGE MASTER

```
CHANGE MASTER TO master_defs
```

Применяется на подчиненных серверах репликации для изменения параметров, определяющих главный компьютер, его подключение или используемый журнал регистрации. Параметр *master_defs* представляет собой список, содержащий одно или несколько определений параметров в формате *param* = *value*, разделенных запятыми. Допустимые определения имеют следующий вид.

- MASTER_CONNECT_RETRY = *n*. Время ожидания между попытками подключения в секундах.
- MASTER_HOST = '*host_name*'. Узел, на котором работает главный сервер.
- MASTER_LOG_FILE = '*file_name*'. Имя двоичного журнала обновления, который будет использоваться при репликации.
- MASTER_LOG_POS = *n*. Место в главном журнале регистрации, с которого будет начинаться или завершаться репликация.
- MASTER_PASSWORD = '*pass_val*'. Пароль подключения к главному серверу.
- MASTER_PORT = *n*. Номер порта для подключения к главному серверу.
- MASTER_USER = '*user_name*'. Имя пользователя для подключения к главному серверу.
- RELAY_LOG_FILE = '*file_name*'. Имя файла регистрации переключения подчиненных серверов.
- RELAY_LOG_POS = *n*. Текущее положение в журнале переключения подчиненных серверов.

За исключением имени узла или номера порта, изменяются только те параметры, которые изменяются явным образом. Изменение узла или порта обычно

свидетельствует о том, что вы переключаетесь на другой главный сервер, поэтому в таких случаях имени журнала двоичного обновления и положению присваиваются пустая строка и нуль соответственно.

Оператор CHANGE MASTER был добавлен в MySQL 3.23.23. Параметры RELAY_LOG_FILE и RELAY_LOG_POS были добавлены в MySQL 4.0.2 (до этого момента журналы переключения репликации еще отсутствовали).

CHECK TABLE

```
CHECK {TABLE | TABLES} tbl_name [, tbl_name] ... [options]
```

Этот оператор проверяет таблицы на ошибки. Он работает с таблицами типа MyISAM и начиная с MySQL 3.23.39 с таблицами InnoDB. Для его выполнения требуется привилегия SELECT.

Необязательный параметр *options* может содержать перечень одного или нескольких следующих параметров (не разделенных запятыми).

- CHANGED. Проверить только те таблицы, которые претерпели изменения после последней проверки или некорректного закрытия.
- EXTENDED. Выполнить расширенную проверку полной целостности таблицы. Например, он проверяет соответствие ключей в индексах и строках данных. Этот параметр может замедлить работу.
- FAST. Проверка только тех таблиц, которые были закрыты правильно.
- MEDIUM. Проверка индекса и таблиц на предмет наличия проблем и выполнение проверки контрольной суммы. Этот параметр задан по умолчанию, если не задано ничего другого.
- QUICK. Проверка только индексов, но не строк данных.

Оператор CHECK TABLE возвращает информацию о результате операции. Например:

```
mysql> CHECK TABLE t;
+--------+-------+----------+----------+
| Table  | Op    | Msg_type | Msg_text |
+--------+-------+----------+----------+
| test.t | check | status   | OK       |
+--------+-------+----------+----------+
```

Операторы ANALYZE TABLE, OPTIMIZE TABLE и REPAIR TABLE возвращают информацию в этом же формате. Столбец Table содержит имя таблицы, с которой производилась эта операция. Столбец Op отражает тип операции и возвращает значение check, analyze, optimize или repair. Столбцы Msg_type и Msg_text обеспечивают информацию о результате операции.

Оператор CHECK TABLE был добавлен в MySQL 3.23.13, но он не работает под управлением Windows до 3.23.25. Параметры QUICK, FAST и MEDIUM были добавлены в MySQL 3.23.16, 3.23.23 и 3.23.31 соответственно. Начиная с 3.23.15 и до 3.23.25 были разрешены только одиночные параметры, которым должно предшествовать выражение TYPE =. После этого выражение TYPE = было исключено и стало разрешено задание нескольких функций.

COMMIT

COMMIT . Фиксирует изменения, сделанные операторами, являющимися частью текущей транзакции, и записывает их в базу данных сделанных изменений. Оператор COMMIT эффективен только с типами таблиц, в которых поддерживаются транзакции. (Для типов таблиц, в которых не поддерживаются транзакции, операторы выполняются немедленно.)

Оператор COMMIT не действует, если режим автовыполнения не был отключен оператором BEGIN или установкой AUTOCOMMIT в 0.

Некоторые из операторов явным образом завершают транзакцию при выполнении оператора COMMIT.

```
ALTER TABLE
BEGIN
CREATE INDEX
DROP DATABASE
DROP INDEX
DROP TABLE
LOAD MASTER DATA
LOCK TABLES
RENAME TABLE
SET AUTOCOMMIT = 1
TRUNCATE TABLE
UNLOCK TABLES (если таблицы в данный момент заблокированы)
```

Оператор COMMIT был добавлен в MySQL 3.23.14.

CREATE DATABASE

```
CREATE DATABASE [IF NOT EXISTS] db_name [[DEFAULT] CHARACTER SET charset]
```

Создает базу данных с заданным именем. Этот оператор не выполняется, если такая база данных уже существует или пользователь не обладает достаточными привилегиями для ее создания. Попытки создать базу данных с именем, которое уже существует, завершаются ошибкой, но если задано предложение IF NOT EXISTS, база данных не создается, но и ошибка не появляется. Это предложение было добавлено в MySQL 3.23.12.

Начиная с MySQL 4.1 предложение DEFAULT CHARACTER SET может быть использовано для спецификации атрибута кодировки по умолчанию для базы данных. Атрибут *charset* может быть именем кодировки или DEFAULT, чтобы база данных использовала текущую кодировку по умолчанию. Атрибуты базы данных хранятся в файле db.opt, размещенном в каталоге базы данных.

CREATE FUNCTION

```
CREATE [AGGREGATE] FUNCTION function_name
  RETURNS {STRING | REAL | INTEGER}
  SONAME 'shared_library_name'
```

Загружает определенную пользователем функцию (user-defined function – UDF) в таблицу *func* базы данных *mysql*. Впоследствии на эту функцию из операторов SQL можно ссылаться по имени *function_name*. Следующее за предложением RETURNS ключевое слово определяет тип значений, возвращаемых функцией. На-

звание файла с исполняемым кодом или функцией и путь к нему определяются элементом '*shared_library_name*'.

Ключевое слово AGGREGATE определяет, что функция является групповой, например, как функции SUM() или MAX(). Это ключевое слово впервые появилось в MySQL версии 3.23.5.

Для выполнения оператора CREATE FUNCTION необходимо, чтобы сервер был построен в виде динамически связанного двоичного кода (не статического), поскольку механизмом UDF используются динамические связи. Указания по написанию определяемых пользователем функций можно найти в справочном руководстве по MySQL.

CREATE INDEX

CREATE [UNIQUE] INDEX *index_name* ON *tbl_name* (*index_columns*). Добавляет индекс *index_name* по таблице *tbl_name*. Этот оператор выполняется как оператор ALTER TABLE ADD INDEX, ALTER TABLE ADD UNIQUE или ALTER TABLE ADD FULLTEXT, в зависимости от того, добавлено ли ключевое слово UNIQUE или FULLTEXT. Более детальную информацию о нем можно найти в описании оператора ALTER TABLE. Оператор CREATE INDEX не может быть использован для создания индекса PRIMARY KEY; используйте вместо него оператор ALTER TABLE.

Если необходимо создать несколько индексов по таблице, лучше всего использовать оператор ALTER TABLE непосредственно; можно создать их непосредственно с одним оператором. Такой метод быстрее, чем добавление индексов по отдельности.

Оператор CREATE INDEX впервые появился в MySQL 3.22. Параметр для создания необязательных индексов FULLTEXT был добавлен в MySQL 3.23.23.

CREATE TABLE

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
   (create_definition,...)
   [table_options]
   [[IGNORE | REPLACE] select_statement]
   [[IGNORE | REPLACE] [AS] select_statement]

create_definition:
   {   col_declaration [reference_definition]
     [CONSTRAINT symbol] PRIMARY KEY (index_columns)
     | [CONSTRAINT symbol] UNIQUE [INDEX | KEY] [index_name]
(index_columns)
     | {INDEX | KEY} [index_name] (index_columns)
     | FULLTEXT [INDEX | KEY] [index_name] (index_columns)
     | [CONSTRAINT symbol] FOREIGN KEY [index_name] (index_columns)
        [reference_definition]
     | [CONSTRAINT symbol] CHECK (expr)
   }

col_declaration:
    col_name col_type
   [NOT NULL | NULL] [DEFAULT default_value]
```

```
  [AUTO_INCREMENT] [PRIMARY_KEY] [UNIQUE [KEY]]
  [COMMENT 'string']

reference_definition:
    REFERENCES tbl_name [(index_columns)]
  [ON DELETE reference_action]
  [ON UPDATE reference_action]
  [MATCH FULL | MATCH PARTIAL]

reference_action:
  {RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT}
```

Оператор CREATE TABLE создает новую таблицу с именем *tbl_name* в текущей базе данных. Если имя задано в формате *db_name*.*tbl_name*, таблица задается в заданной базе данных. Если указывается ключевое слово TEMPORARY, таблица существует только до конца текущего клиентского соединения или до запуска оператора DROP TABLE, после чего удаляется. Временную таблицу может "видеть" только создавший ее пользователь.

Как правило, попытка создать таблицу с уже используемым именем приводит к появлению ошибки. Однако эта ошибка не появляется в двух случаях. Во-первых, при определении предложения IF NOT EXISTS ошибка не выводится, хотя и таблица при этом не создается. Во-вторых, если определено ключевое слово TEMPORARY, а существующая таблица не является временной, создается новая временная таблица, однако на время ее существования исходная таблица *tbl_name* недоступна для клиента. Все остальные клиенты могут продолжать с ней работать. Исходная таблица становится доступной в следующем сеансе работы или сразу после удаления временной таблицы с помощью оператора DROP TABLE или после переименования временной таблицы.

Список *create_definition* определяет имена создаваемых столбцов и индексов. Он необязателен, если таблица создается с помощью вложенного оператора SELECT. Предложение *table_options* позволяет определить различные свойства таблицы. При определении оператора *select_statement* (представленного в форме любого оператора SELECT) таблица создается с помощью результирующего набора, возвращаемого оператором SELECT. Более подробно эти предложения описываются в следующих разделах.

Ключевое слово TEMPORARY, предложения IF NOT EXISTS и *table_options*, а также возможность создания таблицы из результатов оператора SELECT впервые появились в версии MySQL 3.23.

Определения столбцов и индексов. В качестве описания *create_definition* может выступать описание столбца, определение индекса, предложение FOREIGN KEY или CHECK. Предложение CHECK задается для совместимости с другими системами управления базами данных и во всех остальных случаях игнорируется. То же самое справедливо и для предложения FOREIGN KEY во всех случаях, за исключением таблицы InnoDB.

Описание столбца *col_declaration* начинается с имени столбца *col_name* и его типа *col_type* и может сопровождаться несколькими необязательными ключевыми словами. В качестве типов столбца можно выбрать любой тип из описанных в приложении Б, "Типы данных". В этом приложении описаны также специфические атрибуты типов. При определении типа столбца можно определить следующие ключевые слова.

- NULL или NOT NULL. Определяет, может ли столбец содержать значения NULL. По умолчанию установлено значение NULL.
- DEFAULT *default_value*. Определяет значение по умолчанию для столбца. Это ключевое слово нельзя использовать для типа BLOB или TEXT.

  Если значение по умолчанию не определено, то оно присваивается автоматически. Так, в столбцах, в которых можно вводить значение NULL, по умолчанию устанавливается значение NULL. Если же это значение запрещено для ввода, значение по умолчанию устанавливается следующим образом.
  - Для числовых столбцов, кроме столбцов AUTO_INCREMENT, значение по умолчанию устанавливается равным 0. В столбце AUTO_INCREMENT значением по умолчанию является следующее значение в последовательности столбца.
  - Для типов даты и времени, кроме TIMESTAMP, по умолчанию устанавливается значение "ZERO", принятое для данного типа (например, "0000-00-00" для типа DATE). Для типа TIMESTAMP по умолчанию устанавливается текущее значение даты и времени и нулевое значение для всех последующих столбцов TIMESTAMP.
  - Для строковых типов, кроме типа ENUM, значением по умолчанию является пустая строка. Для типа ENUM таким значением является первый элемент перечня.
- AUTO_INCREMENT. Это ключевое слово применяется только к столбцам целочисленных типов. Особенность столбца AUTO_COLUMN заключается в том, что при вставке в него значения NULL реально вставляется целое число, которое будет на единицу больше максимального текущего значения столбца. По умолчанию значения столбца AUTO_INCREMENT начинаются с 1. Для таблиц типа MyISAM (и для таблиц типа HEAP, начиная с MySQL 4.1) первое значение может быть определено явным образом параметром AUTO_INCREMENT = *n*. Кроме того, столбец должен быть определен как индекс UNIQUE или PRIMARY KEY и не должен иметь значения NOT NULL. Число столбцов AUTO_INCREMENT в таблице не может превышать одного.
- PRIMARY KEY. Определяет столбец как индекс PRIMARY KEY. PRIMARY KEY должен быть NOT NULL.
- UNIQUE [KEY]. Определяет столбец как индекс UNIQUE. Этот атрибут можно определять в MySQL версии 3.23 и выше.
- COMMENT '*string*'. Определяет описательный комментарий, связанный со столбцом. До MySQL 4.1 этот атрибут анализировался, но игнорировался, а начиная с версии 4.1 он запоминается и отображается операторами SHOW CREATE TABLE и SHOW FULL COLUMNS.

Предложения PRIMARY KEY, UNIQUE, INDEX, KEY и FULLTEXT определяют индексы, причем первые два определяют индексы, которые должны содержать уникальные значения. Предложения INDEX и KEY являются синонимами и определяют индексы, которые могут содержать повторяющиеся значения. Определяемый индекс базируется на перечисленных в *index_columns* столбцах, каждый из которых должен быть столбцом таблицы *tbl_name*. Если указывается сразу несколько

столбцов, их названия следует разделить запятыми. Для столбцов типа CHAR и VARCHAR можно индексировать префикс столбца, используя синтаксис *col_name*(*n*) для индексирования первых *n* символов значений столбца. (Исключением являются таблицы типа InnoDB, в которых использование префиксов индексов недопустимо.) Для столбцов BLOB и TEXT необходимо индексировать по значению префикса; индексировать по всему столбцу нельзя. В индексе FULLTEXT префиксы для столбцов игнорируются. Если пользователь не указывает имя индекса *index_name*, оно выбирается автоматически на основании имени первого индексируемого столбца.

Индексы FULLTEXT допустимы только для таблиц типа MyISAM и только для столбцов типа TEXT и CHAR (не объявленный как BINARY) и VARCHAR.

В версиях, предшествующих MySQL 3.23, индексируемые столбцы в таблицах типа MyISAM и HEAP 4.0.2 необходимо определять как NOT NULL. Во всех более поздних версиях таким образом описывать необходимо только столбцы, входящие в индексы PRIMARY KEY.

Табличные параметры. Предложение *table_options* появилось в MySQL версии 3.23. Если оно присутствует, то может содержать один или несколько описанных ниже параметров. Если определяется сразу несколько параметров, они не разделяются запятыми. Каждая определенная спецификация применяется ко всем типам таблиц (если не указано иное).

- AUTO_INCREMENT = *n*. Первое значение AUTO_INCREMENT, генерируемое для таблицы (используется только для таблиц MyISAM и, начиная с MYSQL 4.1, для таблиц типа HEAP).
- AVG_ROW_LENGTH = *n*. Приблизительная средняя длина строки для таблиц типа MyISAM. Для таблиц типа MyISAM, MySQL для определения максимального размера файла данных используется произведение значений AVG_ROW_LENGTH и MAX_ROWS. Дескриптор MyISAM может использовать с таблицей указатели длиной 1–8 байт. По умолчанию длины указателя должно хватить для адресации 4 Гбайт. Если требуется таблица большего размера (и используемая операционная система имеет поддержку файлов большего размера), параметры таблиц MAX_ROWS и AVG_ROW_LENGTH позволяют дескриптору MyISAM настраивать длину внутреннего указателя. Большое произведение этих значений вынуждает дескриптор использовать указатели большего размера. (Соответственно, уменьшение произведения позволяет использовать указатели меньшего размера. Если говорить об одной таблице, это не позволит сэкономить много пространства, но при наличии большого количества таблиц малого размера суммарная экономия может оказаться существенной.)
- [DEFAULT] CHARACTER SET *charset*. Задает кодировку по умолчанию. Значение *charset* может быть именем кодировки или DEFAULT для указания кодировки базы данных или использования кодировки сервера, если кодировка не указана. Этот параметр определяет кодировку, которая используется для символических столбцов, объявленных без указания какой-либо кодировки. В следующем примере столбцу c1 будет присвоена кодировка sjis, а столбцу c2 — ujis:

```
CREATE TABLE t
(
    c1 CHAR(50) CHARACTER SET sjis,
    c2 CHAR(50)
) CHARACTER SET ujis;
```

Этот параметр также имеет отношение ко всем последующим модификациям таблицы, сделанным с помощью оператора ALTER TABLE для символических столбцов, в которых кодировка не задана напрямую.

Оператор CHARACTER SET был добавлен в MySQL 4.1. Он может быть задан любой из нескольких возможных синонимических форм. Следующие объявления являются эквивалентными:

```
CHARACTER SET charset
CHARSET = charset
CHARSET charset
```

Эти синонимические формы также могут использоваться и в других местах, где могут задаваться, таких как определения столбцов или операторах CREATE DATABASE и ALTER DATABASE.

- CHECKSUM = {0 | 1}. Если этот параметр имеет значение 1, MySQL рассчитывает контрольную сумму для каждой строки таблицы. Это незначительно отразится на скорости обновления таблицы, однако повысит шансы успешной проверки таблицы (используется только для таблиц MyISAM).
- COMMENT = '*string*'. Комментарий к таблице, максимальная длина которого не должна превышать 60 символов. Этот комментарий отображается операторами SHOW CREATE TABLE и SHOW TABLE STATUS.
- DATA DIRECTORY = '*dir_name*'. Этот параметр используется только для таблиц MyISAM и определяет каталог, куда должен записываться файл с данными (.MYD). Значение '*dir_name*' должно быть полным путем. Этот параметр был добавлен в MySQL 4.0 и работает, только если сервер был запущен без параметра --skip-symlink. Во многих операционных системах (Mac OS X, FreeBSD и BSDI) символические ссылки не являются безопасными и отключены по умолчанию.
- DELAY_KEY_WRITE = {0 | 1}. Если этот параметр имеет значение 1, кэш индекса периодически обновляется для всей таблицы, а не после каждой операции вставки (используется только для таблиц формата MyISAM).
- INDEX DIRECTORY = '*dir_name*'. Этот параметр используется только для таблиц MyISAM и определяет каталог, куда должен записываться индексный файл (.MYI). Наименование '*dir_name*' должно быть полным путем. Этот параметр был добавлен в MySQL 4.0, и на него накладываются те же ограничения, что и на параметр DATA DIRECTORY.
- INSERT_METHOD = {NO | FIRST | LAST}. Параметр используется в таблицах типа MERGE для определения правила добавления строк. Значение NO полностью отключает вставки. Значения FIRST или LAST отражают метод вставки строк в начало или в конец таблиц MyISAM, которые составляют таблицы MERGE. Этот параметр был добавлен в MySQL 4.0.

- MAX_ROWS = *n*. Максимальное число строк, которое планируется хранить в таблице. Описание параметра AVG_ROW_LENGTH отражает принцип использования значений. (Только для таблиц MyISAM.)
- MIN_ROWS = *n*. Минимальное число строк, которое планируется хранить в таблице. Этот параметр может использоваться для таблиц HEAP при подсказке дескриптору HEAP метода оптимизации использования памяти.
- PACK_KEYS = {0 | 1 | DEFAULT}. Этот параметр управляет сжатием индекса для таблиц типа MyISAM и ISAM, которые позволяют обрабатывать аналогичные сжимаемые значения. Это, безусловно, отражается на скорости обновления, однако при этом повышается эффективность извлечения.

  Если этот параметр имеет значение 1, это обозначает сжатие значений строк (CHAR и VARCHAR) и (для таблиц MyISAM) цифровых индексных значений. Если же этот параметр имеет значение 0, сжатие индексов не осуществляется. С версии MySQL 4.0 может использоваться значение DEFAULT, которое задает сжатие только для длинных строковых столбцов.
- PASSWORD = '*string*'. Определяет пароль для шифрования файла описания таблицы. Как правило, этот параметр не приводит к желаемому эффекту, поскольку активизируется только для определенных клиентов.
- RAID_TYPE = {1 | STRIPED | RAID0}
  RAID_CHUNKS = *n*
  RAID_CHUNKSIZE = *n*

  Эти параметры используются вместе и впервые появились в MySQL 3.23.12 для использования с таблицами типа MyISAM. Они позволяют получить бо́льшие эффективные размеры таблиц. Параметры не действуют, если MySQL не настроена с параметром `--with-raid` во время построения.

  Значением параметра RAID_TYPE по умолчанию является STRIPED; другие два типа фактически являются псевдонимами STRIPED. Параметры RAID_CHUNKS и RAID_CHUNKSIZE управляют распределением пространства, которое будет использовано под данные таблиц. Сервер создает несколько подкаталогов в каталоге базы данных (количество их определяется значением RAID_CHUNKS) и создает в каждом из них файл *tbl_name*.MYD. По мере добавления строк в таблицы сервер пишет их в файл, расположенный в первом каталоге, до тех пор, пока он не заполнится полностью, а затем переходит к следующем каталогу. Размер файла в каждом каталоге управляется значением RAID_CHUNKSIZE, которое измеряется в Мбайт (1024 байт). Каталоги именуются с помощью десятичных цифр в последовательности 00, 01 и т.д. Например, если значение RAID_CHUNKS равно 256, а RAID_CHUNKSIZE – 1000, сервер создает 256 каталогов с именами от 00 до ff и записывает до 1000 Мбайт данных в файле, содержащемся в каждом каталоге.
- ROW_FORMAT = {DEFAULT | FIXED | DYNAMIC | COMPRESSED}. Этот параметр имеет отношение только к таблицам типа MyISAM и определяет тип хранения строк (был добавлен в MySQL 3.23.6).
- TYPE = {ISAM | MYISAM | MERGE | HEAP | BDB | INNODB}. Определяет формат хранения таблицы. Основные характеристики форматов хранения описаны в разделе "Спецификация типа памяти" главы 3, "Синтаксис и ис-

пользование языка SQL". В версии MySQL 3.23 и более поздних по умолчанию используется формат MyISAM, если не заданы другие настройки сервера (во время построения сервера или его запуска с параметром `--default-tabletype`). Типы `MRG_MYISAM`, `BERKELEYDB` и `INNOBASE` являются синонимами `MERGE`, `BDB` и `INNODB` соответственно. Определение допустимого типа таблиц, для которого отсутствует дескриптор, приводит к использованию формата хранения, принятого по умолчанию. Задание ошибочного значения данного параметра приводит к ошибочным результатам. Этот параметр был добавлен в MySQL 3.23. В более ранних версиях MySQL оператор `CREATE TABLE` создает таблицы в формате ISAM.

- `UNION = (tbl_list)`. Этот параметр используется для таблиц типа MERGE. Он задает список таблиц типа MyISAM, составляющих таблицу MERGE.

Вложенный оператор SELECT. При добавлении предложения *`select_statement`* (запрос `SELECT`) таблица создается с помощью результирующего набора, возвращаемого запросом. Строки, содержащие дублирующие значения в столбцах уникального индекса, либо пропускаются, либо заменяются существующими строками. Решение об этом принимается в зависимости от наличия слова `IGNORE` или `REPLACE`. Если ни одно из них не определено, возникает ошибка и все оставшиеся записи пропускаются.

Поддержка внешних ключей. Дескриптор таблиц InnoDB обеспечивает поддержку внешних ключей. Внешний ключ в подчиненной таблице объявляется как `FOREIGN KEY`. Кроме того, могут быть заданы необязательное имя индекса, перечень столбцов, составляющих внешний ключ, и определение `REFERENCES`. Если имя индекса задано, оно будет проигнорировано. Определение `REFERENCES` задает порождающую таблицу и столбцы, к которым относится внешний ключ, и определяет, что необходимо сделать, когда обнаружена порождающая запись. Действия, которые осуществляет дескриптор InnoDB, включают `CASCADE` (удаление соответствующих записей порожденной таблицы) и `SET NULL` (установление в соответствующих столбцах внешнего ключа значения `NULL`). Действия `RESTRICT`, `NO ACTION` и `SET DEFAULT` анализируются, но игнорируются.

Предложения `ON UPDATE` и `MATCH` в определениях `REFERENCE` анализируются, но не выполняются. (Для таблиц типа, отличного от InnoDB, определение внешнего ключа игнорируется.)

Предположим, например, что необходимо создать таблицу с тремя столбцами. Столбец `id` представляет собой ключ `PRIMARY KEY`, а два столбца имени индексируются вместе. Можно воспользоваться следующим оператором:

```
CREATE TABLE customer
(
   id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
   last_name CHAR(30) NOT NULL,
   first_name CHAR(20) NOT NULL,
   PRIMARY KEY (id),
   INDEX (last_name, first_name)
);
```

Теперь создадим временную таблицу и для ускорения работы сделаем ее HEAP-таблицей (т.е. загрузим ее в память):

```
CREATE TEMPORARY TABLE tmp_table
   (id MEDIUMINT NOT NULL UNIQUE, name CHAR(40))
   TYPE = HEAP;
```

Создать новую таблицу, как копию уже существующей, можно следующим образом:

```
CREATE TABLE prez_copy SELECT * FROM president;
```

Чтобы создать таблицу, используя только часть другой таблицы, достаточно ввести следующий оператор:

```
CREATE TABLE prez_alive SELECT last_name, first_name, birth
   FROM president WHERE death IS NULL;
```

Если описание определяется для таблицы, созданной и заполненной с помощью вложенного оператора SELECT, то заданное описание применяется уже после вставки содержимого таблицы. Так, например, определить включение извлеченного столбца в ключ PRIMARY KEY можно следующим образом:

```
CREATE TABLE new_tbl (PRIMARY KEY (a)) SELECT a, b, c FROM old_tbl;
```

DELETE

```
DELETE [LOW_PRIORITY] [QUICK] FROM tbl_name
[WHERE where_expr] [ORDER BY ...] [LIMIT n]

DELETE [LOW_PRIORITY] [QUICK] tbl_name [, tbl_name] ...
FROM tbl_name [, tbl_name] ...
 [WHERE where_expr]

DELETE [LOW_PRIORITY] [QUICK] FROM tbl_name [, tbl_name] ...
USING tbl_name [, tbl_name] ...
[WHERE where_expr]
```

Первый формат оператора удаляет строки из таблицы *tbl_name*. Удалению подлежат все строки, удовлетворяющие критерию, заданному в предложении WHERE:

```
DELETE FROM score WHERE event_id = 14;
DELETE FROM member WHERE expiration < CURDATE();
```

Если предложение WHERE отсутствует, *из таблицы удаляются все записи.*

При определении ключевого слова LOW_PRIORITY оператор не выполняется до тех пор, пока все клиенты не закончат чтение таблицы. Это ключевое слово впервые появилось в версии MySQL 3.22.5.

Для таблиц MyISAM указание предложения QUICK помогает ускорить работу оператора, дескриптор MyISAM не выполнит свою обычную процедуру слияния ветвей индексного дерева. Предложение QUICK было добавлено в MySQL 3.23.25.

Значение *n* предложения LIMIT определяет максимальное число строк, которые могут быть удалены. Предложение LIMIT впервые появилось в MySQL 3.22.7. С предложением ORDER BY строки удаляются в указанном порядке. Совместно с LIMIT это позволяет точнее управлять удалением строк. Предложение ORDER BY было добавлено в MySQL 4.0.0, его синтаксис совпадает с синтаксисом предложения ORDER BY оператора SELECT. После выполнения оператор DELETE возвращает количество удаленных записей. Исключение составляют лишь те случаи, когда не определено предложение WHERE и удаляются все записи таблицы. В MySQL

эта процедура оптимизирована, и вместо последовательного удаления всех строк создаются новые файлы данных и индексов. Хотя такой подход значительно ускоряет работу, подсчет строк не выполняется, в результате чего возвращается нулевое значение. Для подсчета числа удаленных строк следует определить предложение WHERE, удовлетворяющее всем строкам. Например:

```
DELETE FROM tbl_name WHERE 1;
```

Этот оператор, однако, будет выполняться несколько медленней, чем оператор без предложения WHERE.

Если вам не нужен счетчик строк, воспользуйтесь другим способом очистки таблицы — оператором TRUNCATE TABLE.

Второй и третий формат оператора DELETE позволяют одновременно удалять строки сразу из нескольких таблиц. Они также позволяют идентифицировать удаляемые строки с помощью объединений между таблицами. Эти форматы появились в версиях MySQL 4.0.0 и 4.0.2 соответственно. Имена, перечисленные в таблицах, можно задавать как *tbl_name* или *tbl_name*.*; причем последняя форма поддерживается для сохранения совместимости с форматом ODBC.

Для того чтобы удалить строки в таблице t1, имеющие значения столбцов id, совпадающие с аналогичными значениями столбцов id в таблице t2, можно воспользоваться вторым форматом синтаксиса:

```
DELETE t1 FROM t1, t2 WHERE t1.id = t2.id;
```

или третьим:

```
DELETE FROM t1 USING t1, t2 WHERE t1.id = t2.id;
```

DESCRIBE

```
{DESCRIBE | DESC} tbl_name [col_name | 'pattern']
{DESCRIBE | DESC} select_statement
```

Этот оператор выводит такую же информацию, как и оператор SHOW COLUMNS. Более подробно о его использовании рассказывается в разделе, посвященном описанию оператора SHOW. Синтаксис с указанием имени столбца позволяет ограничить вывод только указанным столбцом. Завершающая строка интерпретируется как шаблон для оператора LIKE и ограничивает вывод только теми столбцами, которые соответствуют шаблону.

Для того чтобы посмотреть данные только по столбцу last_name таблицы president, необходимо:

```
DESCRIBE president last_name;
```

Для того чтобы посмотреть данные по столбцам last_name и first_name таблицы president, необходимо:

```
DESCRIBE president '%name';
```

Оператор DESCRIBE с вложенным оператором SELECT является синонимом оператора EXPLAIN. Подробнее об этом можно узнать в описании оператора EXPLAIN. (Действительно, в MySQL операторы DESCRIBE и EXPLAIN являются полными синонимами, но оператор DESCRIBE чаще используется для получения описания таблиц, а оператор EXPLAIN для получения информации о выполнении оператора SELECT.)

DO

```
DO expr [, expr] ...
```

Оценка выражений без возвращения какого-либо результата. Этот оператор удобен для оценки оператора SELECT, так как он не дает результирующего набора. Например, оператор DO может быть использован для установки переменных или вызова функций, в которых вы заинтересованы, прежде всего, из-за их побочных эффектов, а не возвращаемых результатов.

```
DO @sidea := 3, @sideb := 4, @sidec :=
SQRT(@sidea*@sidea+@sideb*@sideb);
DO RELEASE_LOCK('mylock');
```

Оператор DO был добавлен в MySQL 3.23.47.

DROP DATABASE

```
DROP DATABASE [IF EXISTS] db_name
```

Удаляет определенную базу данных. Рекомендуется всегда проверять выполнение оператора по завершении его работы. Оператор приостановит свою работу, если указанная таблица не существует (и не определено предложение IF EXISTS) или запустивший его пользователь не обладает достаточными привилегиями. Предложение IF EXISTS можно использовать, чтобы избежать вывода сообщения об ошибке при отсутствии указанной базы данных. Это предложение впервые появилось в MySQL 3.22.2.

Как известно, база данных представляется в виде каталога, расположенного в каталоге данных. Если в этот каталог базы данных поместить файлы, не имеющие никакого отношения к таблицам, то в процессе выполнения оператора DROP DATABASE они удалены не будут. Не удалится и собственно каталог. Поэтому его имя по-прежнему можно будет найти в распечатке оператора SHOW DATABASES.

DROP FUNCTION

```
DROP FUNCTION function_name
```

Удаляет определенную пользователем функцию, загруженную ранее с помощью оператора CREATE FUNCTION.

DROP INDEX

```
DROP INDEX index_name ON tbl_name
```

Удаляет индекс *index_name* таблицы *tbl_name*. Этот оператор обрабатывается так же, как и ALTER TABLE DROP INDEX. Детально о его функциях рассказывается в разделе, посвященном оператору ALTER TABLE. Оператор DROP INDEX не может использоваться для удаления индекса PRIMARY KEY; для этих целей предназначен оператор ALTER TABLE.

Оператор DROP INDEX впервые появился в MySQL версии 3.22.

DROP TABLE

```
DROP TABLE [IF EXISTS] tbl_name [,tbl_name] ... [RESTRICT | CASCADE]
```

Удаляет указанную таблицу (или таблицы) из базы данных. Если в оператор включено предложение IF EXISTS, удаление несуществующей базы данных не приводит к ошибке. Это предложение впервые появилось в MySQL версии 3.22.2.

Ключевые слова RESTRICT и CASCADE не действуют. Они анализируются для совместимости с перенесенным кодом из других баз данных и будут проигнорированы. Эти ключевые слова можно использовать с MySQL 3.23.29.

EXPLAIN

```
EXPLAIN tbl_name [col_name | 'pattern']
EXPLAIN select_statement
```

Первая форма этого оператора является аналогом оператора DESCRIBE *tbl_name*. Для получения детальной информации об этом операторе обратитесь к описанию его синтаксиса.

Вторая форма оператора EXPLAIN предоставляет информацию о том, как MySQL выполнит представленные за ключевым словом EXPLAIN операторы SELECT.

```
EXPLAIN SELECT score.* FROM score, event
  WHERE score.event_id = event.event_id AND event.event_id = 14;
```

Выводимый результат выполнения оператора EXPLAIN состоит из одной из нескольких строк со следующими столбцами.

- table. Таблица, на которую ссылается строка вывода.
- type. Тип соединения, выполняемый системой MySQL. В качестве возможных типов могут выступать (в порядке ухудшения качества) system, const, eq_ref, ref, range, index и ALL. Типы, обеспечивающие наивысшее качество соединения, задают больше ограничений, т.е. при их определении MySQL в процессе выборки приходится просматривать наименьшее число строк.
- possible_keys. Индексы, которые MySQL рассматривает как возможные кандидатуры для нахождения строк в таблице. Значение NULL указывает на отсутствие индексов.
- key. Индекс, используемый MySQL для нахождения строк таблицы. Значение NULL указывает, что ни один индекс использоваться не будет.
- key_len. Длина используемой части индекса. Если MySQL будет использовать левый крайний префикс индекса, это значение не может быть больше полной длины строки индекса.
- ref. Значение, с которыми MySQL сравнивает значения индекса. Слово const или ??? указывает на сравнение с константой, а имя столбца отражает операцию сравнения столбцов.
- rows. Приблизительное число строк таблицы, которое MySQL необходимо исследовать перед выполнением запроса. Приведенное в этом столбце значение отражает приблизительное общее число комбинаций строк, которые необходимо просмотреть во всех таблицах.

- Extra. Значение Only index в этом столбце указывает, что для извлечения данных достаточно информации индексов и просматривать файл данных не нужно. Значение where used указывает на использование информации в предложении WHERE оператора SELECT.

FLUSH

FLUSH *flush_option* [, *flush_option*] ...

Обновляет различные кэши, используемые сервером. В качестве параметра *flush_option* может выступать любой из следующих элементов.

- DES_KEY_FILE. Перезагружает файл DES-ключей, который используется функциями DES_ENCRYPT() и DES_DECRYPT() для кодирования и декодирования. Этот параметр был добавлен в MySQL 4.0.1.
- HOSTS. Обновляет кэш компьютера.
- LOGS. Обновляет файлы журналов, закрывая и заново открывая их.
- MASTER. Переименован в параметр RESET MASTER, который теперь и должен использоваться вместо него.
- PRIVILEGES. Перезагружает таблицы разрешений. При обновлении таблиц с помощью оператора GRANT или REVOKE сервер автоматически перезагружает их копии, находящиеся в памяти. При обновлении таблиц напрямую с помощью таких операторов, как INSERT и UPDATE, необходимо явно задать серверу их перезагрузку. Этот оператор также сбрасывает пределы управления ресурсами в нуль, как и параметр USER_RESOURCES.
- QUERY CACHE. Обновление кэша запросов для его дефрагментации без удаления операторов из кэша. (Для того чтобы очистить кэш полностью, воспользуйтесь оператором RESET QUERY CACHE.) Этот параметр был добавлен в MySQL 4.0.1.
- SLAVE. Этот оператор был переименован в RESET SLAVE, который теперь и должен использоваться вместо него.
- STATUS. Повторно инициализирует переменные состояния. Этот параметр был добавлен в MySQL 3.22.11.
- TABLES [*tbl_name* [, *tbl_name*] ...]. Без имени таблицы закрывает все таблицы, открытые в кэше таблиц. Начиная с MySQL 3.23.23 появилась возможность задавать необязательные имена таблиц, которые разделяются запятыми, что позволяет обновить только указанные таблицы, а не все таблицы базы данных. Так же как оператор FLUSH TABLE, является синонимом FLUSH TABLES.

  Если задействован кэш запросов, оператор FLUSH TABLES также обновляет кэш запросов.
- TABLES WITH READ LOCK. Обновляет все таблицы во всех базах данных, а затем блокирует их для чтения. Эту блокировку можно снять оператором UNLOCK TABLES. Данный параметр позволяет клиентам читать из таблиц, но блокирует любые изменения в них, что применимо при выполнении процедуры резервного копирования всего сервера и гарантирует неизменность таблиц на протяжении процесса резервного копирования.

Конечно, с точки зрения клиента это означает, что за период блокировки обновление таблиц увеличивается. Этот параметр был добавлен в MySQL 3.23.18.

- USER_RESOURCES. Обновление счетчиков для ограничения управления ресурсами учетных записей (таких как MAX_QUERIES_PER_HOUR). Счетчики учетных записей, достигшие предельных значений, снова готовы продолжать выполнять свои функции. Этот параметр был добавлен в MySQL 4.0.2.

Оператор FLUSH требует привилегии RELOAD. Оператор FLUSH впервые появился в MySQL 3.22.9, а некоторые параметры – позднее.

GRANT

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)] ] ...
  ON {*.* | * | db_name.* | db_name.tbl_name | tbl_name}
  TO account [IDENTIFIED BY 'password']
    [, account [IDENTIFIED BY 'password'] ] ...
  [REQUIRE security_options]
  [WITH grant_or_resource_options]
```

Оператор GRANT предоставляет привилегии доступа одному или нескольким пользователям MySQL. Предоставляемые привилегии определяются значением *priv_type*. Это значение, в свою очередь, состоит из одного или нескольких разделенных запятыми типов привилегий, выбранных из следующего списка. ALL используется отдельно само по себе. Другие привилегии можно задавать по одному или все сразу в виде списка, разделенного запятыми. Значение ALL обозначает комбинацию всех привилегий, за исключением привилегии GRANT OPTION, которая всегда задается отдельно или с предложением WITH GRANT OPTION.

Спецификация привилегии	Операция, разрешенная привилегией
ALTER	Изменение таблиц и индексов
CREATE	Создание баз данных и таблиц
CREATE TEMPORARY TABLES	Создание временных баз данных
DELETE	Удаление записей из таблиц
DROP	Удаление баз данных и таблиц
EXECUTE	Выполнение хранимых процедур (зарезервировано на будущее)
FILE	Чтение и запись файлов на сервере
GRANT OPTION	Присвоение привилегий другим учетным записям
INDEX	Создание или удаление индексов
INSERT	Вставка новых записей в таблицы
LOCK TABLES	Закрытие таблицы явным образом с помощью оператора LOCK TABLES
PROCESS	Просмотр информации о потоках, выполняемых внутри сервера, и завершение их работы
REFERENCES	Не используется
RELOAD	Перезагрузка таблиц разрешений и обновление журналов, кэша главного компьютера и кэша таблиц

Спецификация привилегии	Операция, разрешенная привилегией
REPLICATION CLIENT	Запрос о местоположении главного и подчиненного серверов
REPLICATION SLAVE	Действует в качестве подчиненного сервера репликации
SELECT	Выборка записей из таблиц
SHOW DATABASES	Право на использование оператора SHOW DATABASES
SHUTDOWN	Завершение работы сервера
SUPER	Удаление процессов и выполнение других действий суперпользователя
UPDATE	Изменение существующих записей таблиц
ALL	Все операции (за исключением GRANT). Аналог – ALL PRIVILEGES
USAGE	Специальная привилегия "без привилегий"

Привилегия LOCK TABLES может применяться только для таблиц, для которых имеется привилегия SELECT, и позволяет задавать любой тип блокировки, а не только блокировку чтения.

Пользователь всегда имеет возможность просмотра и удаления своих процессов, однако привилегия SUPER позволяет просматривать и удалять любые процессы.

Привилегии CREATE TEMPORARY TABLES, EXECUTE, LOCK TABLES, REPLICATION CLIENT, REPLICATION SLAVE, SHOW DATABASES и SUPER были добавлены в MySQL 4.0.2. До версии 4.0.2 операции, требующие привилегий SHOW DATABASES, привилегии репликации или SUPER управлялись привилегиями SELECT, FILE и PROCESS соответственно.

Предложение ON определяет, какому уровню данных предоставляются привилегии. Для этого применяются спецификации, описанные в приведенной ниже таблице.

Спецификация привилегии	Уровень применения привилегии
ON *.*	Глобальные привилегии: все базы данных и все таблицы
ON *	Глобальные привилегии, если не определена база данных по умолчанию. В любом другом случае привилегии на доступ к текущей базе данных
ON *db_name*	Привилегии уровня базы данных: все таблицы указанной базы данных
ON *db_name.tbl_name*	Привилегии уровня таблицы: все столбцы указанной таблицы
ON *tbl_name*	Привилегии уровня таблицы: все таблицы указанной таблицы базы данных по умолчанию

Если в предложении ON указывается имя конкретной таблицы, привилегии можно предоставлять для доступа к определенным столбцам. При перечислении в списке *column_list* их названия разделяются запятыми. (Это имеет отношение только к привилегиям INSERT, REFERENCES, SELECT и UPDATE, которые являются единственными привилегиями, предоставляемыми для отдельных столбцов.)

В предложении TO определяется пользователь (или пользователи), которому предоставляются привилегии. Каждый спецификатор user включает имя *'user_name'@'host_name'* с необязательным предложением IDENTIFIED BY. Части *user_name* и *host_name* заключать в кавычки не обязательно, если они не

содержат специальных символов. Но если они берутся в кавычки, то должны браться в кавычки раздельно. (Например, `bill@%.com` берется в кавычки как `'bill'@'%.com'`, а не как `'bill@%.com'`.) В качестве имени пользователя *user_name* может выступать как обычное имя, так и пустая строка (`''`), которая определяет анонимного пользователя. В качестве имени компьютера *host_name* можно указывать ключевое слово *localhost*, имя компьютера, IP-адрес или образец, соответствующий имени домена или сетевому адресу. Символы типа "%" и "_" имеют такое же значение, как и в операторе LIKE. Предложение *user_name* без определенного имени компьютера является аналогом записи *'user_name'@'%'*. В MySQL версии 3.23 и выше в качестве имени компьютера *host_name* может выступать пара IP-адрес/маска сети, представленная в записи *n.n.n.n/m.m.m.m*, где *n.n.n.n* — IP-адрес, а *m.m.m.m* — сетевая маска, используемая для сетевого номера.

Предложение IDENTIFIED BY позволяет присваивать пароль пользователю. Этот пароль должен задаваться простым текстом (без функции PASSWORD), в отличие от способа определения пароля с помощью оператора SET PASSWORD. Если такой пользователь уже существует, а предложение IDENTIFIED BY определено, новый пароль заменит старый. В любом другом случае старый пароль остается неизменным.

Предложение REQUIRE позволяет установить безопасные соединения и информацию, которую должен предоставить при этом клиент. Ключевое слово REQUIRE может иметь следующие атрибуты.

- NONE. Для обозначения того, что безопасное соединение не требуется.
- Общий тип соединения SSL, который указывает, что соединения данной учетной записи используют протокол SSL.
- X509. Пользователь должен предоставить действительный сертификат X509. В этом случае клиент может предоставить сертификат X509.
- Один или несколько следующих параметров требует, чтобы соединение устанавливалось с соответствующими характеристиками:
  - CIPHER `'str'` требует установки соединения с шифром, указанным в строке `'str'`;
  - ISSUER `'str'` требует, чтобы организация, выдавшая сертификат, имела название, заданное строкой `'str'`;
  - SUBJECT `'str'` требует, чтобы тема сертификата совпадала со значением, заданным строкой `'str'`.

При указании более чем одного из этих параметров их необходимо разделять служебным словом AND. Порядок следования параметров не имеет никакого значения.

Предложение WITH указывает, что учетная запись позволяет присваивать привилегии, которыми наделена учетная запись, другим учетным записям. Начиная с MySQL 4.0.2 оно может использоваться для наложения ограничений на потребление ресурсов данной учетной записью. Допустимые параметры перечислены ниже. Можно задавать сразу несколько параметров, а их порядок не будет иметь никакого значения.

- GRANT OPTION. Эта учетная запись может присваивать свои привилегии другим учетным записям, включая право на присвоение привилегий.
- MAX_CONNECTIONS_PER_HOUR *n*. Эта учетная запись имеет право делать *n* соединений с сервером в час.
- MAX_QUERIES_PER_HOUR *n*. Эта учетная запись имеет право делать к серверу *n* запросов в час.
- MAX_UPDATES_PER_HOUR *n*. Эта учетная запись имеет право делать к серверу *n* модифицирующих запросов в час.

Значение *n*, устанавливающее предел, равное 0, означает "нет ограничений".

Оператор GRANT впервые появился в MySQL 3.22.11. Предложение REQUIRE было добавлено в MySQL 4.0.0; его параметр NONE был добавлен в 4.0.4, в это же время разделитель параметров AND между параметрами REQUIRE стал необязательным. Параметры управления ресурсами были добавлены в 4.0.2.

Ниже представлено несколько примеров использования оператора GRANT. Дополнительные примеры можно найти в главе 11, "Общее администрирование MySQL". Информация об установке безопасных соединений с использованием протокола SSL содержится в главе 12, "Безопасность".

Сначала создадим пользователя paul, который с любого компьютера сможет получать доступ ко всем таблицам базы данных sampdb. Два следующих оператора являются эквивалентными, поскольку отсутствующая часть имени компьютера в идентификаторе user является аналогом символа "%":

```
GRANT ALL ON sampdb.* TO 'paul' IDENTIFIED BY 'secret';
GRANT ALL ON sampdb.* TO 'paul'@'%' IDENTIFIED BY 'secret';
```

Теперь создадим пользователя с правами чтения таблиц базы данных menagerie. Этот пользователь сможет подключаться к ним только с компьютеров, принадлежащих домену xyz.com:

```
GRANT SELECT ON menagerie.* TO 'lookonly'@'%.xyz.com'
  IDENTIFIED BY 'ragweed';
```

Для создания пользователя, который обладает полными привилегиями для доступа к таблице member базы данных sampdb, но может подключиться только с одного компьютера, воспользуйтесь следующим оператором:

```
GRANT ALL ON sampdb.member TO 'member_mgr'@'boa.snake.net'
  IDENTIFIED BY 'doughnut';
```

Чтобы создать суперпользователя, который может выполнять все операции, в том числе и предоставлять привилегии другим пользователям, однако может подключаться только с локального компьютера, примените оператор

```
GRANT ALL ON *.* TO 'superduper'@'localhost' IDENTIFIED BY 'homer'
  WITH GRANT OPTION;
```

И наконец, создадим анонимного пользователя для базы данных menagerie, который может подключаться с локального компьютера без пароля:

```
GRANT ALL ON menagerie.* TO ''@'localhost';
```

Создадим учетную запись для удаленного пользователя, который будет подключаться с использованием протокола SSL и предоставлять действительный сертификат X509:

```
GRANT ALL ON privatedb.*
TO 'paranoid'@'%.mydom.com' IDENTIFIED BY 'keepout'
REQUIRE X509;
```

Создадим учетную запись для пользователя с ограниченным доступом, который может издавать не больше 100 запросов в час, из которых только 10 могут быть запросами обновления:

```
GRANT ALL ON test.*
TO 'caleb'@'localhost' IDENTIFIED BY 'rosepetal'
WITH MAX_QUERIES_PER_HOUR 100 MAX_UPDATES_PER_HOUR 10;
```

HANDLER

```
HANDLER tbl_name OPEN [AS alias_name]
HANDLER tbl_name READ
   [FIRST | NEXT]
   [where_clause] [limit_clause]
HANDLER tbl_name READ index_name
   [FIRST | NEXT | PREV | LAST | [< | <= | = | => | >] (expr_list)]
   [where_clause] [limit_clause]
HANDLER tbl_name CLOSE
```

Оператор HANDLER обеспечивает интерфейс низкого уровня для дескрипторов MyISAM и InnoDB, которые обходят оптимизатор и получают непосредственный доступ к содержимому. Чтобы получить непосредственный доступ к дескриптору через интерфейс HANDLER, сначала для него необходимо открыть таблицу оператором HANDLER ... OPEN. Таблица будет доступной до тех пор, пока не будет задан оператор HANDLER ... CLOSE для закрытия доступа явным образом, или до завершения соединения. Пока таблица открыта, для получения доступа к ее содержимому можно воспользоваться оператором HANDLER ... READ.

Оператор HANDLER не обеспечивает защиты от параллельных обновлений. Он не блокирует таблицу, поэтому таблица может быть изменена, когда HANDLER будет открыт. Поэтому нет никакой гарантии того, что изменения, происходящие в таблицах, найдут отражение в записях, прочитанных вами из файла.

Оператор HANDLER был добавлен для таблиц MyISAM в MySQL 4.0.0 и может работать с таблицами InnoDB начиная с версии 4.0.3.

INSERT

```
INSERT [LOW PRIORITY | DELAYED] [IGNORE] [INTO]
   tbl_name [(column_list)] VALUES (expr [, expr] ...) [, (...)] ...
INSERT [LOW PRIORITY | DELAYED] [IGNORE] [INTO]
   tbl_name [(column_list)] SELECT ...
INSERT [LOW PRIORITY | DELAYED] [IGNORE] [INTO]
   tbl_name SET col_name=expr [, col_name=expr] ...
```

Вставляет строки в существующую таблицу *tbl_name* и возвращает количество добавленных строк. Ключевое слово INTO необходимо использовать только в MySQL 3.22.5.

При определении ключевого слова LOW_PRIORITY оператор не выполняется, пока хоть один пользователь читает данные таблицы. Это слово впервые появилось в MySQL версии 3.22.5.

При определении ключевого слова DELAYED строки помещаются в очередь для более поздней вставки, и оператор завершает свое выполнение. Благодаря этому клиент может продолжить свою работу. Однако в этом случае функция LAST_INSERT_ID() не возвратит значение AUTO_INCREMENT для столбца AUTO_INCREMENT. Ключевое слово DELAYED впервые появилось в MySQL версии 3.22.15 и работает только для таблиц ISAM и MyISAM.

Если задано ключевое слово IGNORE, все строки с дублирующим значением для уникальных ключей игнорируются. Если же это слово не задано, а такие строки имеются, возникает ошибка и последующие строки не вставляются. Ключевое слово IGNORE впервые появилось в MySQL версии 3.22.10.

Первая форма оператора INSERT требует определения списка VALUES() всех вставляемых значений. Если список столбцов *column_list* не определен, список VALUES() должен задавать по одному значению для каждого столбца таблицы. Если в списке *column_list* указывается несколько имен столбцов, их следует разделять запятыми. Соответственно для каждого указанного столбца в списке VALUES() необходимо определить по одному значению. Столбцам, имена которых не указаны в списке, присваиваются значения по умолчанию. В версиях MySQL 3.22.5 и более поздних можно определять несколько списков значений, что позволяет с помощью одного оператора INSERT вставлять сразу несколько строк. В MySQL 3.23.3 и более поздних версиях *column_list* и VALUES() могут быть пустыми. В таком случае вставляется новая строка, во всех столбцах которой устанавливаются значения по умолчанию.

```
INSERT INTO t () VALUES();
```

Начиная с MySQL 4.0.3 в списке VALUES() для установки в столбцах их значений по умолчанию явным образом может использоваться слово DEFAULT. При этом знание точного его значения не обязательно.

Вторая форма оператора INSERT выбирает записи в соответствии с установками оператора SELECT и вставляет их в таблицу *tbl_name*. Оператор SELECT должен выбирать столько столбцов, сколько содержится в таблице *tbl_name* или определено в списке *column_list*. Если определен список столбцов, то не указанные в нем столбцы заполняются значениями по умолчанию. Выбирать записи из таблицы, в которую вы производите вставку, нельзя.

Третья форма оператора INSERT, впервые появившаяся в MySQL 3.22.10, вставляет определенные в предложении SET столбцы, устанавливая в них заданные выражения. Если имя столбцов не указывается, в них устанавливаются значения по умолчанию.

```
INSERT INTO absence (student_id, date)
       VALUES(14, '1999-11-03'),(34,NOW())
INSERT INTO absence SET student_id = 14, date = '1999-11-03'
INSERT INTO absence SET student_id = 34, date = NOW()
INSERT INTO score (student_id, score, event_id)
  SELECT student_id, 100 AS score, 15 AS event_id FROM student
```

KILL

```
KILL thread_id
```

Удаляет на сервере процесс с идентификатором *thread_id*. Для выполнения этого оператора пользователь должен обладать привилегией SUPER (до

MySQL 4.0.2 – PROCESS). Иначе пользователь сможет удалить только свой собственный поток.

Команда `mysqladmin kill` позволяет выполнить аналогичную операцию, хотя в ее командной строке можно задать сразу несколько идентификаторов потоков. В строке оператора KILL можно задать только один идентификатор.

Этот оператор впервые появился в MySQL 3.22.9.

LOAD DATA

```
LOAD DATA [LOW PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
   [IGNORE | REPLACE]
   INTO TABLE tbl_name
   import_options
   [IGNORE n LINES]
   [(column_list)]
```

Оператор LOAD DATA считывает записи из файла *file_name* и загружает их в таблицу *tbl_name*. Этот оператор работает быстрее, чем INSERT.

При задании ключевого слова LOW_PRIORITY оператор не выполняется, если кто-либо из пользователей в данный момент считывает данные из таблицы. Это ключевое слово впервые появилось в MySQL 3.23.

Ключевое слово CONCURRENT используется только для таблиц MyISAM. Он позволяет другим клиентам производить выборку из таблицы, пока в нее загружаются другие строки. Ключевое слово CONCURRENT было добавлено в MySQL 3.23.38.

Без ключевого слова LOCAL заданный файл считывается сервером с компьютера, на котором установлен сервер MySQL. Соответственно, пользователь, запускающий этот оператор, должен обладать привилегией FILE, а сам файл должен располагаться в каталоге базы данных или быть доступным для считывания. Если определено ключевое слово LOCAL, файл считывается с компьютера клиента и отправляется на сервер по сети. В этом случае необходимость в привилегии FILE отпадает. Впервые ключевое слово LOCAL появилось в MySQL версии 3.22.15. Однако начиная с 3.23.49 LOCAL может включаться или отключаться избирательно. Когда оно отключено на стороне сервера, его нельзя использовать со стороны клиента. Если оно включено на стороне сервера, но по умолчанию отключено на клиентской стороне, включить его можно только явным образом. Например, для включения возможности LOCAL с программой `mysql` может использоваться флаг `--local-infile`.

Если слово LOCAL не задано, сервер пытается самостоятельно найти заданный файл, руководствуясь следующими правилами.

- Если имя '*file_name*' задано как абсолютный путь, сервер ищет файл, начиная с корневого каталога.
- Если имя '*file_name*' задано как относительный путь, его интерпретация зависит от того, содержит ли это имя один или больше компонентов. Если определен только один компонент, сервер ищет файл в каталоге установленной по умолчанию базы данных. Если же заданное имя содержит несколько компонентов, сервер ищет файл, начиная с каталога данных сервера.

При определении ключевого слова LOCAL заданное имя файла интерпретируется следующим образом:

- если имя *'file_name'* задано как абсолютный путь, программа-клиент ищет файл, начиная с корневого каталога;
- если имя *'file_name'* задано как относительный путь, программа-клиент ищет файл, начиная с текущего каталога пользователя.

Для ОС Windows обратные косые черты в именах файлах могут отображаться обычными косыми ('/') или двойными обратными косыми чертами ('\\').

В зависимости от использования ключевых слов IGNORE и REPLACE, строки с дублирующими значениями уникальных индексов либо пропускаются, либо вставляются вместо существующих. Если ни первое ни второе слово не определено, а такая запись встречается, возникает ошибка и все оставшиеся строки будут пропущены. Если LOCAL задано, передача файла не может быть прервана; таким образом, по умолчанию поведение совпадает с поведением IGNORE, если ни один из параметров обработки дублирующих записей не задан.

Предложение *import_options* определяет формат данных. Задаваемые в нем параметры могут также использоваться и в предложении *export_options* оператора SELECT ... INTO OUTFILE. Предложение *import_options* имеет следующий синтаксис:

```
[FIELDS
   [TERMINATED BY 'string']
   [OPTIONALLY] ENCLOSED BY 'char']
   [ESCAPED BY 'char']]
[LINES
   [STARTING BY 'string']
   [TERMINATED BY 'string'] ]
```

Для представления специальных символов в значениях *'string'* и *'char'* могут применяться следующие управляющие последовательности.

Последовательность	Значение
\0	ASCII-символ 0
\b	Забой
\n	Новая строка (перевод строки)
\r	Возврат каретки
\s	Пробел
\t	Табуляция
\'	Одинарная кавычка
\"	Двойные кавычки
\\	Обратная косая черта (\)

В MySQL 3.22.10 и более поздних версиях для определения комбинаций символов можно также использовать шестнадцатеричные константы. Предложение LINES TERMINATED BY 0x02, например, определяет, что строки разрываются после нажатия комбинации клавиш <Ctrl+B> (ASCII 2).

Если задано ключевое слово FIELDS, необходимо определить по крайней мере одно из предложений, TERMINATED BY, ENCLOSED BY или ESCAPED BY, причем строго в установленном порядке. Если заданы оба слова, FIELDS и LINES, параметры FIELDS должны задаваться первыми.

Предложение FIELDS включает следующие части.

- Предложение TERMINATED BY определяет символ или символы, которые ограничивают значения внутри строки.
- Предложение ENCLOSED BY определяет символ кавычек, который ограничивает значения поля. Оно может задаваться как с ключевым словом OPTIONALLY, так и без него. Для операторов вывода (SELECT ... INTO OUTFILE) определенный в предложении ENCLOSED BY символ используется для выделения значений поля в выводимой строке. Если задано ключевое слово OPTIONALLY, значения выделяются определенными символами только для столбцов типа CHAR и VARCHAR.

  Чтобы включить определенный предложением ENCLOSED BY символ в значение поля, его необходимо либо продублировать, либо установить после символа ESCAPED BY. В любом другом случае он будет рассматриваться как знак конца поля. Для включения такого символа в значения полей выводимых строк первым устанавливается символ, определенный предложением ESCAPED BY.
- Заданный в предложении ESCAPED BY символ используется для определения того, что следующий за ним символ является специальным. Так, в предыдущих примерах в качестве символа выхода выступает обратная косая черта ("\"). Следовательно, в вводимой информации "\N" (обратная косая черта и N) интерпретируется как NULL, а "\0" (обратная косая черта и ASCII-символ "0") – как нулевой байт. Во всех остальных случаях символ выхода отбрасывается, а следующий за ним символ помещается в строку. Например, запись "\"" интерпретируется как символ двойной кавычки, даже если предыдущим предложением ENCLOSED BY определено, что двойные кавычки указывают на конец строки.

В выводимой информации символ выхода используется для кодирования NULL и нулевых байтов с помощью записей "\N" и "\0" соответственно. Кроме того, чтобы ввести в строке символы, определенные предложениями ESCAPED BY и ENCLOSED BY в качестве символов выхода и окончания строки, необходимо установить перед ними еще один управляющий символ. Если символ ESCAPED BY пуст (ESCAPED BY ''), подобная обработка не выполняется. Чтобы ввести управляющий символ "\", введите его дважды (ESCAPED BY '\\').

Части предложения LINES используются следующим образом.

- Значение LINES STARTING BY определяет символ или символы, указывающие на начало строк.
- Значение предложения LINES TERMINATED BY определяет символ или символы, указывающие на окончание строк.

Если предложения FIELDS и LINES не определены, используются значения, установленные по умолчанию. С помощью операторов их можно было бы задать следующим образом:

```
FIELDS
   TERMINATED BY '\t'
   ENCLOSED BY ''
```

```
   ESCAPED BY '\'
LINES
   STARTING BY ''
   TERMINATED BY 'n'
```

Другими словами, если поля в строке не выделены кавычками, обратная косая черта рассматривается как управляющий символ, а строки обрываются после ввода символов перехода на новую строку.

Если значения TERMINATED BY и ENCLOSED BY предложения FIELDS являются пустыми (незаполненными), используется формат строки фиксированной ширины, в которой поля не отделяются друг от друга специальными разделителями. В этом случае значения столбцов считываются (или записываются при выводе) на основании заданной ширины столбца. Так, например, столбцы значений VARCHAR(15) и MEDIUMINT(5) считываются соответственно как 15- и 5-символьные поля. Значения NULL записываются как строки пробелов.

Значения NULL во входном файле данных отображаются последовательностью \N, не взятой в кавычки. Если символ, заданный в FIELDS ENCLOSED BY, не является пустым, все значения, отличные от NULL, должны быть ограничены символом, заданным как ограничитель, а не взятые в кавычки значения NULL будут интерпретированы как значения NULL.

Если задается предложение IGNORE *n* LINES, первые *n* строк ввода пропускаются. Например, если файл данных содержит строку с заголовками столбцов, которую вставлять в таблице базы данных необязательно, можно добавить предложение IGNORE 1 LINES:

```
LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl IGNORE 1 LINES;
```

Если список *column_list* не определен, предполагается, что строки ввода содержат по одному значению для каждого столбца таблицы. Если же в этом списке содержится одно или несколько разделенных запятыми имен столбцов, строки ввода должны включать значения для всех перечисленных столбцов. Столбцам, которые не указаны в списке, присваиваются значения по умолчанию. Если же в строке ввода указано меньше значений, чем имен столбцов в списке *column_list*, неопределенным столбцам присваиваются значения по умолчанию.

При обработке файла с отформатированным текстом, созданным на компьютере с Windows, можно использовать установленный по умолчанию разделитель столбца. Следует учитывать, что строки в таком тексте наверняка разрываются с помощью пары управляющих символов "возврат каретки/переход на новую строку". Для загрузки такого файла определите новый ограничитель строки ("\r" отражает символ возврата каретки, а "\n" – переход на новую строку):

```
LOAD DATA LOCAL INFILE 'my_tbl.txt' INTO TABLE my_tbl
   LINES TERMINATED BY '\r\n';
```

К сожалению, обработка созданных под управлением ОС Windows файлов может закончиться записью ошибочной информации в базу данных, если создавшая этот файл программа использует устаревшее правило помещать в конце файла символ <Ctrl+Z>. Вам придется либо записать файл с помощью более современной программы, либо удалить ошибочную запись после загрузки файла.

В некоторых файлах в формате CSV (comma-separated values – значения разделены запятыми) значения разделены запятыми, а поля заключены в двойные ка-

вычки. Если предположить, что в конце строк такого файла размещаются символы перехода на новую строку, то оператор LOAD DATA для считывания такого файла должен выглядеть следующим образом:

```
LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl
   FIELDS TERMINATED BY ',' ENCLOSED BY '"';
```

Приведенный ниже оператор позволяет считать файл, в котором поля разделены символами <Ctrl+A> (ASCII 1), а строки заканчиваются символом <Ctrl+B> (ASCII 2).

```
LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl
   FIELDS TERMINATED BY 0x01 LINES TERMINATED BY 0x02;
```

LOAD ... FROM MASTER

```
LOAD DATA FROM MASTER
LOAD TABLE tbl_name FROM MASTER
```

Эти операторы используются на подчиненных серверах репликации для запроса данных из главного сервера. Оператор LOAD DATA FROM MASTER запрашивает все таблицы с главного сервера. Он также вносит коррективы в координаты подчиненного сервера, чтобы подчиненный сервер мог забирать с главного сервера только те обновленные данные, которые были добавлены после завершения операции LOAD DATA.

На передаваемые таблицы могут налагаться ограничения любым параметром `--replicate-xxx`, с которым запускается подчиненный сервер. Этот оператор сам по себе имеет множество ограничений. Подробнее об этом — в разделе "Установка взаимоотношений репликации главный–подчиненный" главы 11, "Общее администрирование MySQL". Если эти условия приемлемы для вас, данный оператор сможет обеспечить удобный метод инициализации подчиненного сервера.

Оператор LOAD TABLE ... FROM MASTER передает копию названных таблиц с главного сервера на подчиненный. Это используется чаще всего при отладке репликации.

LOAD TABLE ... FROM MASTER был добавлен в MySQL 3.23.19, а LOAD DATA FROM MASTER — в MySQL 4.0.0.

LOCK TABLE

```
LOCK TABLE
LOCK {TABLE | TABLES}
   tbl_name [AS alias_name] lock_type
   [, tbl_name [AS alias_name] lock_type] ...
```

Устанавливает блокировку на указанные таблицы. Параметр *lock_type* может иметь одно из следующих значений.

- READ. Блокировка чтения. Блокировка любой записи в таблицу другими клиентами, но чтение другими клиентами разрешено.
- READ LOCAL. Вариант блокировки READ, разработанный для ситуаций, при которых возможна параллельная запись. Она применима только к таблицам типа MyISAM, в которых при удалении записей не образуется никаких "дыр". Блокировка READ LOCAL позволяет блокировать таблицы явным образом, позволяя при этом параллельные вставки в таблицу MyISAM, в кото-

рой нет "дыр". (Если таблица и имеет "дыры", блокировка считается обычной блокировкой READ.)

- WRITE. Блокировка записи. Полностью блокирует доступ другим клиентам, независимо от того, хочет ли он производить чтение из таблицы или запись в таблицу.
- LOW_PRIORITY WRITE. Этот тип блокировки позволяет другим читателям производить чтение, если запрос другого клиента ожидает уже читающего таблицу клиента. Блокировка не устанавливается до тех пор, пока все клиенты не перестанут считывать данные таблицы. Если в момент запуска оператора с таким видом блокировки кто-то из клиентов считывает данные таблицы, то и остальные смогут к ней обратиться. Только по завершении считывания всех пользователей оператор будет выполнен.

Оператор LOCK TABLE позволяет задавать также и псевдоним для блокируемой таблицы, чтобы впоследствии использовать его в запросах. Если обращение к таблице в запросе выполняется несколько раз, необходимо заблокировать каждый образец таблицы.

При выполнении оператора LOCK TABLE снимаются также все имеющиеся в настоящее время блокировки. Следовательно, чтобы заблокировать несколько таблиц, необходимо все их указать в одном операторе LOCK TABLES. Все установленные клиентом блокировки автоматически снимаются по завершении сеанса его работы. Блокировка чтения LOW_PRIORITY была добавлена в MySQL 3.22.8, а READ LOCAL – в MySQL 3.23.11.

```
LOCK TABLES student READ, score WRITE, event READ;
LOCK TABLE member READ;
LOCK TABLES t AS t1 READ, t AS t2 READ;
```

OPTIMIZE TABLE

```
OPTIMIZE {TABLE | TABLES} tbl_name [, tbl_name] ...
```

Освобождает неиспользуемое свободное пространство таблицы. Результатом появления такого неиспользуемого пространства может быть выполнение операторов DELETE, REPLACE и UPDATE, особенно для таблиц со строками переменной длины. Оператор OPTIMIZE TABLE выполняет следующие действия:

- дефрагментирует таблицы для уменьшения неиспользуемого пространства и уменьшения размера таблицы;
- объединяет содержимое строк переменной длины, которые стали фрагментированными в процессе эксплуатации на отдельные фрагменты;
- по мере необходимости сортирует индексные страницы;
- обновляет внутреннюю табличную статистику.

Оператор OPTIMIZE TABLE может использоваться только с таблицами MyISAM и BDB (для таблиц BDB он в равносилен оператору ANALYZE TABLE). Для работы с этим оператором требуются привилегии SELECT и INSERT для каждой таблицы.

Действие оператора OPTIMIZE TABLE аналогично выполнению программы myisamchk с параметрами --check-only-changed, --quick, --sort-index и

--analyze. Однако при работе с программой myisamchk необходимо предпринять меры, которые смогли бы предотвратить доступ к таблице. При работе с оператором OPTIMIZE TABLE дела обстоят несколько иначе: сервер должен заблокировать доступ другим клиентам во время выполнения процедуры оптимизации.

Оператор OPTIMIZE TABLE выполняет вывод в формате оператора CHECK TABLE.

Оператор OPTIMIZE TABLE был добавлен в MySQL 3.22.7.

PURGE MASTER LOGS

```
PURGE MASTER LOGS TO 'log_name'
```

Удаляет двоичные журналы обновления на сервере, которые были созданы раньше указанного файла регистрации, и обновляет индексный файл, в котором должны быть перечислены только оставшиеся журналы. Обычно такая процедура осуществляется после отработки оператора SHOW SLAVE STATUS на каждом подчиненном сервере для определения того, какие регистрационные файлы по-прежнему используются. Работа с этим оператором требует привилегии SUPER (до MySQL 4.0.2 – привилегии PROCESS).

Для того чтобы удалить файлы binlog.001–binlog.009 (или те, которые из них существуют), делает файл binlog.010 первым из оставшихся журнальных файлов:

```
PURGE MASTER LOGS TO 'binlog.010';
```

Оператор PURGE MASTER LOGS был добавлен в MySQL 3.23.28.

RENAME TABLE

```
RENAME {TABLE | TABLES} tbl_name TO new_tbl_name [, ...]
```

Переименовывает одну и несколько таблиц. Оператор RENAME TABLE подобен оператору ALTER TABLE ... RENAME, за исключением того, что он может переименовать несколько таблиц одновременно, закрывая их во время операции переименования. Это предоставляет определенные преимущества, если необходимо заблокировать доступ к таблицам во время выполнения процедуры переименования.

Оператор RENAME TABLE был добавлен в MySQL 3.23.23.

REPAIR TABLE

```
REPAIR {TABLE | TABLES} tbl_name [, tbl_name] ... [options]
```

Этот оператор предназначен для выполнения операций восстановления. Он работает только для таблиц MyISAM, и для него требуются привилегии SELECT и INSERT для каждой таблицы. Необязательный параметр *options* представляет собой список с указанием одного или нескольких параметров (не разделяемых запятыми).

- EXTENDED. Выполняет расширенное восстановление с восстановлением индексов. Его функциональность аналогична запуску программы myisamchk --safe-recover по таблицам, за исключением того, что восстановление выполняется сервером, а не внешней программой.
- QUICK. Восстановление только индексов; данные не затрагиваются.

- USE_FRM. Использует файл .frm таблицы для определения того, как можно интерпретировать содержимое файла данных, а затем использует файл данных для перестройки индексного файла. Это может пригодиться в тех случаях, когда файл утерян или не подлежит восстановлению.

Оператор REPAIR TABLE без параметров выполняет процедуру восстановления таблицы аналогично myisamchk --recover.

Оператор REPAIR TABLE выполняет вывод в формате, описанном в записи CHECK TABLE.

Оператор REPAIR TABLE был добавлен в MySQL 3.23.14. Параметры QUICK и USE_FRM были добавлены в MySQL 3.23.16 и 4.0.2. В версиях 3.23.14–3.23.25 разрешено указание только одного параметра, и ему должен предшествовать квалификатор TYPE =; однако потом указание TYPE = было упразднено и разрешено использование сразу нескольких параметров.

REPLACE

```
REPLACE [LOW_PRIORITY | DELAYED] [INTO]
   tbl_name [(column_list)]
   VALUES (expr [, expr] ...) [, (...)] ...
REPLACE [LOW_PRIORITY | DELAYED] [INTO]
   tbl_name [(column_list)]
   SELECT ...
REPLACE [LOW_PRIORITY | DELAYED] [INTO]
   tbl_name SET col_name=expr [, col_name=expr] ...
```

Оператор REPLACE аналогичен оператору INSERT с той лишь разницей, что при вставке строки с дублирующим значением для индекса UNIQUE одной из уже существующих строк таблицы старое значение удаляется перед вставкой нового. Именно поэтому в синтаксисе оператора REPLACE отсутствует ключевое слово IGNORE. Более детально об элементах этого оператора рассказывается в описании оператора INSERT.

Оператор REPLACE может удалять больше одной строки, так как таблица может содержать несколько уникальных индексов. Это может произойти, если значению в новой строке соответствуют значения сразу в нескольких из уникальных индексов. В таком случае перед вставкой новой строки все соответствующие строки удаляются.

Для работы с оператором REPLACE требуются привилегии INSERT и DELETE. До MySQL 4.0.5 он также требовал привилегии UPDATE.

RESET

```
RESET option [, option] ...
```

По своему эффекту на информацию в журналах или кэше оператор RESET аналогичен оператору FLUSH. (И действительно, оператор RESET начинал свое существование как оператор FLUSH.) Ниже приведены значения, которые могут принимать его параметры.

- MASTER. Удалить существующий двоичный журнал обновления для репликации главного сервера, создать новый регистрационный журнал с последовательно-

стью нумерации, начинающейся с 001, и переустановить индекс двоичного журнала обновления, чтобы в нем отражались только новые файлы.

- QUERY CACHE. Очистка кэша запросов с удалением запросов, записанных в нем. (Для дефрагментации без очистки используйте FLUSH QUERY CACHE.)
- SLAVE. Если сервер работает как подчиненный сервер репликации, то этот параметр указывает ему "забыть" координаты репликации (т.е. имя файла его текущего двоичного журнала репликации и положение внутри этого файла).

Работа с оператором RESET требует привилегии RELOAD.

Оператор RESET был добавлен в MySQL 3.23.26. Параметр QUERY CACHE было добавлен в MySQL 4.0.1. В версиях 3.23.19–3.23.25 операторам RESET MASTER и RESET SLAVE соответствуют FLUSH MASTER и FLUSH SLAVE.

RESTORE TABLE

```
RESTORE {TABLE | TABLES} tbl_name [, tbl_name] ... FROM 'dir_name'
```

Восстанавливает заданную таблицу или таблицы на основании данных из файлов, содержащихся в каталоге резервных копий, созданных с помощью оператора BACKUP TABLE. Строка *'dir_name'* должна быть полным путем к компьютеру с сервером, содержащему файлы с резервными копиями. Восстанавливаемые таблицы должны уже существовать.

Оператор RESTORE TABLE работает для таблиц MyISAM и требует привилегий INSERT и FILE. При восстановлении используются только файлы определения структуры и файлы данных (файлы .frm и .MYD). Индексы перестраиваются в соответствии с информацией, содержащейся в этих файлах.

Оператор RESTORE TABLE был добавлен в MySQL 3.23.25.

REVOKE

```
REVOKE priv_type [(column_list)] [, priv_type [(column_list)] ...]
  ON {*.* | * | db_name.tbl_name | tbl_name}
  FROM account [,account] ...
```

Оператор REVOKE лишает пользователя (или пользователей) указанных привилегий. Предложения *priv_type*, *column_list* и *account* определяются так же, как и в операторе GRANT. Те же спецификаторы можно использовать и в предложении ON.

В процессе выполнения оператор REVOKE не удаляет пользователя из таблицы разрешений. Это означает, что пользователь все еще сможет подключиться к серверу MySQL. Чтобы полностью запретить пользователю подключение, необходимо вручную удалить его запись из таблицы разрешений. (Однако рассматривается возможность автоматического удаления записи из таблицы разрешений.)

Оператор REVOKE впервые появился в MySQL версии 3.22.11.

Рассмотрим несколько примеров. Чтобы лишить пользователя superduper@localhost всех привилегий, выполните следующий оператор:

```
REVOKE ALL ON *.* FROM 'superduper'@'localhost';
```

Теперь удалите привилегии, благодаря которым пользователь member_mgr мог изменять таблицу member базы данных sampdb:

```
REVOKE INSERT,DELETE,UPDATE ON sampdb.member FROM
'member_mgr'@'boa.snake.net';
```

И наконец, для удаления всех привилегий доступа к единственной таблице базы данных `menagerie` со стороны анонимного пользователя локального компьютера воспользуйтесь следующим оператором:

```
REVOKE ALL ON menagerie.pet FROM ''@'localhost';
```

ROLLBACK

```
ROLLBACK
```

Этот оператор отменяет все изменения, сделанные операторами, которые являются частью текущей транзакции, и, таким образом, "забывает" все внесенные изменения. Это имеет отношение только к тем типам таблиц, которые могут работать с транзакциями. (Для типов таблиц, не имеющих возможностей транзакций, все операторы выполняются сразу же, поэтому их действие не может быть отменено.)

ROLLBACK не делает ничего, если режим автовыполнения не был отключен оператором BEGIN или параметр AUTOCOMMIT установлен в 0.

Оператор ROLLBACK был добавлен в MySQL 3.23.14.

SELECT

```
SELECT
   [select_options]
   select_list
   [
     INTO OUTFILE 'file_name' export_option]
     | INTO DUMPFILE 'file_name'
     | INTO @var_name [, @var_name ] ...
   ]
   [FROM tbl_list
   [WHERE where_expr]
   [GROUP BY {unsigned_integer | col_name | formula} [ASC | DESC] ,
...]
   [HAVING where_expr]
   [ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC] , ...]
   [LIMIT [offset,] rows]
   [PROCEDURE procedure_name (arg_list) ]
     [FOR UPDATE | LOCK IN SHARE MODE] ]
```

Делает выборку строк из определенных таблиц. Практически все элементы синтаксиса этого оператора, за исключением ключевого слова SELECT и предложения *select_list*, являются необязательными, что позволяет использовать этот оператор для расчета выражений:

```
SELECT 'Один плюс один равно ', 1+1
```

Предложение *select_options* может содержать следующие ключевые слова.

- ALL; DISTINCT; DISTINCTROW. Определяют, будут ли возвращаться строки с дублирующими значениями. Слово ALL приводит к возврату всех строк и используется по умолчанию. Ключевые слова DISTINCT и DISTINCTROW определяют, что все дублирующие строки исключаются из результирующего набора.

- HIGH_PRIORITY. Добавление ключевого слова HIGH_PRIORITY присваивает оператору высокий приоритет. Если другие операторы, такие, например, как INSERT или UPDATE, ожидают перед записью в определенную оператором SELECT таблицу, пока пользователи не перестанут считывать данные таблицы, то слово HIGH_PRIORITY позволяет оператору SELECT получить более высокий по сравнению с ними приоритет. Это ключевое слово следует использовать только для операторов SELECT, которые исполняются быстро и должны быть выполнены немедленно, поскольку выполнение операторов записи или обновления значительно замедлится. Ключевое слово HIGH_PRIORITY впервые появилось в MySQL версии 3.22.9.
- SQL_BUFFER_RESULT. Позволяет серверу сохранить результат запроса в отдельной временной таблице, а не блокировать таблицу или таблицы, заданные в операторе SELECT, в ожидании того момента, когда весь результат запроса будет отослан клиенту. Это позволяет серверу быстрее снять блокировку и значительно быстрее получить доступ к таблице другим пользователям. (Однако использование этого параметра требует больше дискового пространства и памяти.) Параметр SQL_BUFFER_RESULT был добавлен в MySQL 3.23.13.
- SQL_CACHE; SQL_NO_CACHE. Если кэш запросов работает в режиме повышенной нагрузки, параметр SQL_CACHE вызывает кэширование результатов запросов. Параметр SQL_NO_CACHE отменяет любое кэширование результатов запросов. Эти параметры были добавлены в MySQL 4.0.1.
- SQL_CALC_FOUND_ROWS. Счетчик строк из запроса, включающего предложение LIMIT, отражает количество возвращенных запросом строк. Параметр SQL_CALC_FOUND_ROWS позволяет определить, насколько большим был бы возвращенный результат без предложения LIMIT. Этот счетчик можно получить с помощью оператора SELECT FOUND_ROWS() после оператора SELECT.
- SQL_BIG_RESULT; SQL_SMALL_RESULT. Эти ключевые слова определяют, что результирующий набор будет большим или, наоборот, малым, что позволяет средству оптимизации более эффективно обработать запрос. Ключевое слово SQL_SMALL_RESULT впервые появилось в MySQL версии 3.22.12, а SQL_BIG_RESULT — в MySQL 3.23.
- STRAIGHT_JOIN. Заставляет таблицы объединяться в порядке, определенном в предложении FROM. Это ключевое слово пользователи могут задавать в том случае, если, на их взгляд, средство оптимизации не в состоянии сделать это лучше.

В предложении *select_list* определяются имена возвращаемых столбцов, разделенные запятыми. Эти столбцы могут быть ссылками на столбцы таблицы или выражения. Каждому такому столбцу можно присвоить псевдоним, используя синтаксис AS *alias_name*. Этот псевдоним впоследствии становится именем столбца в выводимых данных и может использоваться для ссылки в предложениях GROUP BY, ORDER BY и HAVING. Нельзя использовать этот псевдоним в предложении WHERE.

Специальный символ "*" означает "все столбцы из таблицы, указанной в предложении FROM", а запись *tbl_name*.* — "все столбцы из таблицы *tbl_name*".

Результат выполнения оператора SELECT можно записать в файл *file_name* с помощью предложения INTO OUTFILE '*file_name*'. Синтаксис предложения *export_options* аналогичен синтаксису предложения *import_options* для оператора LOAD DATA.

Предложение INTO DUMPFILE '*file_name*' аналогично предложению INTO OUTFILE, но записывает только одну строку и вывод, не интерпретируя последний. Другими словами, он пишет значения без каких-либо ограничителей или кавычек, что может очень пригодиться при записи двоичных данных типа BLOB в двоичный файл. Параметр INTO DUMPFILE был добавлен в MySQL 3.23.5.

Как для параметра INTO OUTFILE, так и для INTO DUMPFILE, имя файла интерпретируется таким же образом, как при считывании нелокальных файлов оператором LOAD DATA. Для этого необходимо обладать привилегиями FILE, а файл с указанным именем существовать не должен.

Начиная с MySQL 4.1 результаты работы оператора SELECT могут сохраняться в наборе переменных, определенных пользователем в виде *@var_name*. Запрос выбирает одну строку значений и задает по одной переменной на выводимый столбец.

В предложении FROM приводятся имена одной или нескольких таблиц, из которых извлекаются строки. MySQL поддерживает следующие типы объединений в операторах SELECT:

```
tbl_list:
   tbl_name
   tbl_list, tbl_name
   tbl_list [CROSS] JOIN tbl_name
   tbl_list INNER JOIN tbl_name ON conditional_expr
   tbl_list INNER JOIN tbl_name USING (column_list)
   tbl_list STRAIGHT_JOIN tbl_name
   tbl_list LEFT [OUTER] JOIN tbl_name ON conditional_expr
   tbl_list LEFT [OUTER] JOIN tbl_name USING (column_list)
   tbl_list NATURAL [LEFT [OUTER]] JOIN tbl_name
   { oj tbl_list LEFT OUTER JOIN tbl_name ON conditional_expr }
   tbl_list RIGHT [OUTER] JOIN tbl_name ON conditional_expr
   tbl_list RIGHT [OUTER] JOIN tbl_name USING (column_list)
   tbl_list NATURAL [RIGHT [OUTER]] JOIN tbl_name
(tbl_list)
```

Имя таблицы может сопровождаться псевдонимом или определением индекса. А это значит, что полный синтаксис, относящийся непосредственно к таблице, выглядит следующим образом:

```
tbl_name
   [[AS] alias_name]
   [USE INDEX (index_list) | IGNORE INDEX (index_list)]
```

Кроме того, псевдонимы могут присваиваться таблицам в предложении FROM с использованием синтаксиса *tbl_name alias_name* или *tbl_name* AS *alias_name*. Псевдоним представляет собой альтернативное имя, по которому можно обращаться к столбцам таблицы в любом месте запроса.

Предложения USE INDEX и IGNORE INDEX могут использоваться с версии MySQL 3.23.12 для предоставления оптимизатору информации об индексах. Это имеет смысл в случаях, когда оптимизатор не может сделать правильный выбор индекса, используемого в объединении. (Параметры USE KEY и IGNORE KEY яв-

ляются синонимами параметров USE INDEX и IGNORE INDEX.) Параметр *index_list* дает перечень одного или нескольких индексов, разделенных запятыми. Каждый индекс в списке *index_list* может быть именем индекса таблицы или ключевым словом PRIMARY, чтобы указать на индекс PRIMARY KEY таблицы.

Типы объединения помогают извлекать строки из названных таблиц, как отражено в представленных описаниях. Хотя реально возвращаемые строки могут ограничиваться предложениями WHERE, HAVING или LIMIT.

- Если в операторе SELECT указана одна таблица, сервер извлекает строки только из нее.
- Если же через запятую определены имена сразу нескольких таблиц, оператор SELECT возвращает все возможные комбинации строк из них. В качестве альтернативы запятым могут выступать ключевые слова JOIN и CROSS JOIN. Подобную функцию выполняет и слово STRAIGHT_JOIN, однако оно заставляет средство оптимизации объединять таблицы в порядке их перечисления. Как уже отмечалось выше, эту особенность можно применять в случаях, когда, по мнению пользователя, средство оптимизации не помогает сделать это лучше.
- Объединение типа INNER JOIN аналогично объединению, которое задается с помощью запятой, но при этом требует указания предложений ON или USING() для наложения ограничений на правила соответствия между таблицами, аналогично тому, как это делает объединение LEFT JOIN. (Однако обратите внимание, что до MySQL 3.23.17 объединение типа INNER JOIN полностью аналогично запятой и не требует предложения ON или USING().)
- Ключевое слово LEFT JOIN также приводит к выборке строк из связанных таблиц, однако сопровождается генерированием строки для каждой строки левой таблицы, даже если в правой таблице соответствующие ей записи отсутствуют. Когда такие записи отсутствуют, столбцы из правой таблицы возвращаются со значениями NULL. Выборка строк осуществляется в соответствии с условиями, определенными в предложениях ON *conditional_expr* или USING (*column_list*). Форма выражения *conditional_expr* может использоваться и в предложении WHERE. Список *column_list* может содержать как одно имя столбца, так и несколько разделенных запятыми имен. Каждое определенное имя указывает на столбец, который есть во всех объединенных таблицах. Предложение LEFT OUTER JOIN аналогично предложению LEFT JOIN. То же можно сказать и о синтаксисе, начинающемся со слова oj (который добавлен для совместимости с ODBC). (Фигурные скобки, показанные для синтаксиса OJ, являются не метасимволами, а литеральными символами и обязательно должны присутствовать в операторе.)
- Предложение NATURAL LEFT JOIN является аналогом предложения LEFT JOIN USING (*column_list*), где список *column_name* содержит имена столбцов, входящих во все указанные таблицы.
- Предложение RIGHT JOIN аналогично предложению LEFT JOIN, но при этом таблицы меняются своими ролями. Предложение RIGHT JOIN появилось в MySQL 3.23.25.

Предложение WHERE определяет выражение, которое применяется к извлеченным из таблиц (заданных в предложении FROM) строкам. (В предложении WHERE нельзя оперировать псевдонимами.) Все строки, не удовлетворяющие заданному критерию, отвергаются. Впоследствии результирующий набор можно еще больше ограничить предложениями HAVING и LIMIT.

Предложение GROUP BY *column_list* группирует строки результирующего набора в соответствии с перечисленными в списке столбцами. Это предложение особенно полезно при определении таких функций, как COUNT() или MAX() в предложении *select_list*. Ссылаться на эти столбцы можно либо по имени, либо по псевдонимам, либо по позиции внутри списка *select_list*. Позиции столбцов нумеруются с 1. В версии 3.23.2 появилась возможность использовать выражения в предложениях GROUP BY. Это позволяет производить группировку по результатам выражений.

В MySQL предложение GROUP BY не только группирует строки, но и сортирует полученные результаты выборки. Начиная с MySQL 3.23.47 после спецификаторов столбцов GROUP BY также можно использовать ключевые слова ASC и DESC. Это позволяет непосредственно задавать порядок сортировки (что влияет и на порядок вывода). Порядок вывода, полученный с помощью предложения GROUP BY, перекрывается любым предложением ORDER BY, если таковое имеется в наличии.

Предложение HAVING задает вторичное выражение, ограничивающее результирующий набор после проверки на условия предложения WHERE. Строки, не соответствующие условиям предложения HAVING, также отбрасываются. Предложение HAVING эффективно используется в выражениях с агрегатными функциями, которые нельзя проверить с помощью предложения WHERE. Однако если какое-либо условие можно задать как в предложении WHERE, так и в предложении HAVING, лучше разместить его в первом. В этом случае условие будет проанализировано средством оптимизации.

Предложение ORDER BY определяет порядок сортировки результирующего набора. Так же как и в предложении GROUP BY, ссылаться на столбцы можно с помощью имени, псевдонима или позиции внутри списка столбцов. Выводимые столбцы по умолчанию сортируются в возрастающем порядке. Чтобы самостоятельно определить порядок сортировки столбцов, укажите ключевое слово ASC (восходящая) или DESC (нисходящая). В MySQL 3.23.2 можно также в качестве столбца сортировки задать выражение. Так, например, предложение ORDER BY RAND() приводит к сортировке строк случайным образом. Однако, в отличие от GROUP BY, выражения в предложении ORDER BY не могут ссылаться на суммирующие (агрегатные) функции.

Для выбора определенных строк из результирующего набора можно использовать предложение LIMIT. Оно принимает от одного до двух аргументов, которые должны быть целыми константами. Так, определение LIMIT *n* приводит к возврату первых *n* строк. Предложение LIMIT *n*, *m* возвращает *n* строк, начиная со строки *m*. Расчет строк начинается с нуля, а не с единицы. Формат с двумя аргументами позволяет задавать предложение LIMIT вида LIMIT m,-1 для пропуска первых *m* строк, а затем произвести выборку оставшихся строк, как бы много их не было.

После ключевого слова PROCEDURE указывается имя процедуры, которой будут посланы данные результирующего набора перед возвратом его клиенту. Список аргументов `arg_list` может быть пустым или списком аргументов, разделенных запятыми, которые должны быть переданы процедуре. В версии MySQL 3.23 предложение PROCEDURE ANALYSE() можно применять для получения информации о характеристиках данных столбцов, перечисленных в списке столбцов.

Предложения FOR UPDATE и LOCK IN SHARE MODE блокируют выбранные строки до тех пор, пока текущая транзакция не будет выполнена полностью или не будет произведен откат. Это может пригодиться при осуществлении транзакций. При использовании предложения FOR UPDATE с таблицей, для которой дескриптор использует блокировку страничного уровня или уровня строк (BDB или InnoDB), выбранные строки являются заблокированными для исключительного использования. Применение LOCK IN SHARE MODE устанавливает блокировки чтения на строки, позволяя другим клиентам только читать их, не модифицируя их при этом. Эти предложения, предназначенные для блокировки, были добавлены в MySQL 3.23.35.

Ниже представлены примеры использования оператора SELECT. Множество полезных примеров можно найти также в главе 1, "Знакомство с СУБД MySQL и SQL", и в главе 3, "Синтаксис и использование языка SQL".

Выборка всего содержимого таблицы:

```
SELECT * FROM president;
```

Выборка всего содержимого таблицы и сортировка по имени и фамилии:

```
SELECT * FROM president ORDER BY last_name, first_name;
```

Выборка записей для всех президентов, рожденных в день "1900-01-01" или позднее:

```
SELECT * FROM president WHERE birth >= '1900-01-01';
```

То же самое, но с сортировкой по датам рождения:

```
SELECT * FROM president WHERE birth >= '1990-01-01' ORDER BY birth;
```

Определение штатов, имеющихся в строках таблицы member:

```
SELECT DISTINCT state FROM member;
```

Выборка строк из таблицы member и запись столбцов в виде разделенных запятыми значений в файл:

```
SELECT * INTO OUTFILE '/tmp/member.txt'
  FIELDS TERMINATED BY ',' FROM member;
```

Выборка первых пяти результатов для определенного события:

```
SELECT * FROM score WHERE event_id = 9 ORDER BY score DESC LIMIT 5;
```

Поддержка встроенных запросов. В СУБД MySQL 4.1 добавлена поддержка, позволяющая вкладывать один оператор SELECT в тело другого запроса. Имеется несколько видов вложенной выборки.

- Вложенная выборка может применяться в операторе сравнения, и в этом случае вложенный оператор SELECT должен давать единственное значение.

  Выбрать запись из таблицы `president`, в которой имеется минимальное значение `birth`:

```
SELECT * FROM president
WHERE birth = (SELECT MIN(birth) FROM president);
```

Выбрать оценки по заданному экзамену, которые больше средней оценки за экзамен:

```
SELECT * FROM score WHERE event_id = 5
AND score > (SELECT AVG(score) FROM score WHERE event_id = 5);
```

- Вложенный оператор SELECT может предваряться операторами EXISTS или NOT EXISTS. В этом случае внутренний оператор SELECT может ссылаться на столбцы из внешнего оператора SELECT, а поэтому во избежание неоднозначности ссылки на столбцы должны быть квалифицированы именами таблиц.

  Выбрать учащихся, кто бы хотя бы один раз отсутствовал на занятиях:

  ```
  SELECT student_id, name FROM student WHERE EXISTS
  (SELECT * FROM absence WHERE absence.student_id =
  student.student_id);
  ```

  Выбрать учащихся, кто не отсутствовал ни разу:

  ```
  SELECT student_id, name FROM student WHERE NOT EXISTS
  (SELECT * FROM absence WHERE absence.student_id =
  student.student_id);
  ```

  Формат SELECT * используется в выборке, так как внутренний запрос должен выдать значение "true" либо "false", а не значения конкретных столбцов.

- Вложенный оператор SELECT может предваряться выражениями IN или NOT IN и в таком случае возвращает значение одного столбца. Предшествующие запросы EXISTS и NOT EXISTS могут быть переписаны для использования IN и NOT IN следующим образом:

  ```
  SELECT student_id, name FROM student
  WHERE student_id IN (SELECT student_id FROM absence);
  SELECT student_id, name FROM student
  WHERE student_id NOT IN (SELECT student_id FROM absence);
  ```

SET

```
SET [OPTION] option_setting [, option_setting] ...
```

Оператор SET используется для определения большого количества параметров, пользовательских переменных, глобальных или сеансовых переменных или уровня изоляции транзакции.

Оператор SET TRANSACTION ISOLATION LEVEL описан отдельно, а информация о пользовательских переменных содержится ниже, в разделе "Переменные языка SQL".

Рекомендуется избегать ключевого слова OPTION, поскольку в будущих версиях MySQL планируется его убрать.

Синтаксис присвоения значений таков: *name op val*, где *name* — параметр SQL или присваиваемая переменная; *op* — оператор присвоения; *val* — присваиваемое значение. До MySQL 3.23.6 в качестве оператора присвоения применялся только оператор "="; а начиная с 3.23.6 и выше уже может использоваться "=" или ":=".

Для присвоения значения любому из нескольких параметров SQL может использоваться оператор SET:

```
SET SQL_LOG_BIN = 1;
SET AUTOCOMMIT = 0;
```

или для установки пользовательских переменных:

```
SET @day = CURDATE(), @time = CURTIME();
```

Кроме того, начиная с MySQL 4.0.3 сервер уже поддерживает несколько динамических системных переменных, которые могут быть изменены во время работы сервера, и для изменения этих переменных используется оператор SET. (До версии 4.0.3 системные переменные устанавливались только во время запуска сервера. Таким образом, для актуализации новых значений переменных требовался перезапуск сервера.) Динамические переменные существуют на двух уровнях. Глобальные переменные распространяются на весь сервер и на всех клиентов.

Сеансовые переменные (также именуемые локальными переменными) являются специфическими только для текущего соединения клиента. Для переменных, которые существуют на двух уровнях, сеансовые переменные инициализируются для каждого нового подключения клиента на основании значений соответствующих глобальных переменных. Любой клиент имеет возможность внести свои коррективы в переменные своего сеанса подключения. Однако, для того, чтобы модифицировать глобальную переменную, требуется привилегия SUPER.

Для установки глобальной переменной (например, table_type) используется оператор, который имеет одну из следующих форм:

```
SET GLOBAL table_type = InnoDB;
SET @@GLOBAL.table_type = InnoDB;
```

Для установки сеансовой переменной замените слово GLOBAL словом SESSION:

```
SET SESSION table_type = InnoDB;
SET @@SESSION.table_type = InnoDB;
```

Как синоним SESSION может использоваться слово LOCAL:

```
SET LOCAL table_type = InnoDB;
SET @@LOCAL.table_type = InnoDB;
```

Если отсутствуют слова GLOBAL, SESSION и LOCAL, оператор SET изменяет по умолчанию переменную сеансового уровня:

```
SET table_type = InnoDB;
SET @@table_type = InnoDB;
```

Для того чтобы проверить значения системных переменных, используется оператор SHOW VARIABLES. В разделах, посвященных этому оператору, дается перечень имеющихся переменных и отображаются те из них, которые можно изменять динамически.

Следующий список описывает параметры SQL, управляемые оператором SET. В некоторых из описаний параметров упомянуто, что для их установки требуется привилегия SUPER; а до MySQL 4.0.2 необходима привилегия PROCESS.

- AUTOCOMMIT = {0 | 1}. Устанавливает режим автовыполнения для обработки транзакций. Установка значения 0 отключает режим автовыполнения, поэтому следующие операторы не выполняются до тех пор, пока не встретится оператор COMMIT или переменная AUTOCOMMIT не будет установлена в 1. Действие операторов, заключенных в транзакцию, можно отменить оператором ROLLBACK, если до этого не встретилась команда вы-

полнения. Установка AUTOCOMMIT в 1 восстанавливает режим автовыполнения (и выполняет все отложенные действия, если таковые имеются). При работе в режиме автовыполнения все операторы выполняются немедленно, т.е. каждый оператор выполняется как отдельная транзакция.

- CHARACTER SET [=] {*charset* | DEFAULT}. Определяет используемый клиентом набор символов. С учетом этого набора символов впоследствии обрабатываются все строки, принятые от клиента и переданные ему. В настоящее время доступен только набор cp1251_koi8. Использование имени набора DEFAULT приводит к восстановлению набора по умолчанию. Символ "=" появился в MySQL 4.0.3 и является необязательным; до этой версии его можно не указывать.
- FOREIGN_KEY_CHECKS = {0 | 1}. Установка этого параметра в 0 или 1 отключает или включает проверку внешних ключей. По умолчанию проверка производится. Отключение проверки ключей может очень пригодиться, например, при выполнении процедуры восстановления из файла резервной копии, которая восстанавливает и загружает таблицы совсем в другом порядке, чем это требуется в соответствии со взаимоотношениями, заданными внешними ключами. После завершения процедуры загрузки таблицы можно восстановить проверку внешних ключей. Этот параметр был добавлен в MySQL 3.23.52. Он действует только для таблиц типа InnoDB.
- INSERT_ID = *n*. Определяет значение, используемое следующим оператором INSERT при вставке столбца AUTO_INCREMENT. Этот параметр используется при обработке журнала обновлений.
- LAST_INSERT_ID = *n*. Определяет значение, возвращаемое функцией LAST_INSERT_ID(). Этот параметр используется при обработке журнала обновлений.
- PASSWORD [FOR *account*] = PASSWORD('*pass_val*'). Если предложение FOR отсутствует, оператор SET с этим параметром устанавливает пароль "*pass_val*" для текущего пользователя. Добавление предложения FOR позволяет задать пароль для определенного пользователя *account*. Пользователь, задающий новый пароль, должен обладать достаточными полномочиями для доступа к базе данных mysql. Пользователь *account* определяется в форме '*user_name*'@'*host_name*'. Значения *user_name* и *host_name* должны быть такого же типа, как и для оператора GRANT.

```
SET PASSWORD = PASSWORD('secret');
SET PASSWORD FOR 'paul' = PASSWORD('secret');
SET PASSWORD FOR 'paul'@'localhost' = PASSWORD('secret');
SET PASSWORD FOR 'bill'@'%.bigcorp.com' = PASSWORD('old-sneep');
```

- SQL_AUTO_IS_NULL = {0 | 1}. Если этот параметр имеет значение 1, последняя строка, вставленная со значением AUTO_INCREMENT, может быть извлечена с помощью предложения WHERE, заданного в форме WHERE *col_name* IS NULL. Такая возможность используется некоторыми ODBC-программами, например СУБД Access. Этот параметр впервые появился в MySQL 3.23.5 и по умолчанию имеет значение 1.
- SQL_BIG_SELECTS = {0 | 1}. Этот параметр используется в совокупности с параметром MAX_JOIN_SIZE:

- Если этот параметр имеет значение 1, оператор SELECT будет возвращать больше строк, чем разрешено параметром MAX_JOIN_SIZE.
- Если параметр SQL_BIG_SELECTS имеет значение 0, то запросы, достигнув значения MAX_JOIN_SIZE, будут завершать свою работу. В этом случае при выполнении оператора типа "объединение" применяется параметр MAX_JOIN_SIZE. Сервер производит оценку количества комбинаций строк. Если это значение превышает значение MAX_JOIN_SIZE, сервер возвращает ошибку и не выполняет оператор.

Установка значения MAX_JOIN_SIZE, отличного от значения DEFAULT, автоматически устанавливает параметр SQL_BIG_SELECTS в 0.

- SQL_BIG_TABLES = {0 | 1} BIG_TABLES = {0 | 1}. Если этот параметр имеет значение 1, все временные таблицы сохраняются на диске, а не в памяти. Это, конечно, отражается на производительности, однако операторы SELECT, создающие большие временные таблицы, в процессе выполнения не диагностируют переполнение таблиц. По умолчанию этот параметр имеет значение 0 (т.е. временные таблицы хранятся в памяти). В MySQL версии 3.23 и выше этот параметр не нужен. Начиная с MySQL 4 более предпочтительным является использование параметра BIG_TABLES.
- SQL_BUFFER_RESULT = *n*. Если этот параметр имеет значение 1, сервер использует временные таблицы для хранения результатов работы оператора SELECT. В результате этого сервер может быстрее снимать блокировку с обрабатываемых таблиц. Этот параметр был добавлен в MySQL 3.23.13.
- SQL_LOG_BIN = {0 | 1}. Если этот параметр имеет значение 1, то активизируется двоичное протоколирование для текущего клиента, установка этого параметра в 0 отключает протоколирование. Для выполнения этого оператора клиент должен иметь привилегию SUPER. Этот параметр был добавлен в MySQL 3.23.16.
- SQL_LOG_OFF = {0 | 1}. Если этот параметр имеет значение 1, запросы текущего клиента не регистрируются в общем журнале. Установка значения 0 приводит к активизации регистрации. Чтобы изменить значения этого параметра, клиент должен обладать привилегией SUPER.
- SQL_LOG_UPDATE = {0 | 1}. Если этот параметр имеет значение 1, запросы текущего клиента регистрируются в журнале обновлений. Установка значения 0 приводит к отключению регистрации. Чтобы изменить значения этого параметра, клиент должен обладать привилегией SUPER.

  Параметр SQL_LOG_UPDATE впервые появился в MySQL 3.22.5.
- SQL_LOW_PRIORITY_UPDATES = {0 | 1} LOW_PRIORITY_UPDATES = {0 | 1}. Если этот параметр имеет значение 1, операторы изменения содержимого таблицы (DELETE, INSERT, REPLACE и UPDATE) не приступают к работе до тех пор, пока остальные пользователи не завершат выборку записей из таблиц. Вновь "прибывающие" операторы SELECT начинают исполняться немедленно, не ожидая выполнения операторов изменения, обладающих более низким приоритетом.

  Параметр SQL_LOW_PRORITY_UPDATES впервые появился в MySQL 3.22.5.

Начиная с MySQL 4 рекомендуется использовать параметр LOW_PRIORITY_UPDATES.

- SQL_MAX_JOIN_SIZE = {*n* | DEFAULT} MAX_JOIN_SIZE = {*n* | DEFAULT}

  Как говорилось в описании параметра SQL_BIG_SELECTS, этот параметр используется вместе с параметром SQL_BIG_SELECTS.

  Начиная с MySQL 4 рекомендуется использовать параметр MAX_JOIN_SIZE.

- SQL_QUERY_CACHE_TYPE = {0 | 1 | 2 | OFF | ON | DEMAND}
  QUERY_CACHE_TYPE = {0 | 1 | 2 | OFF | ON | DEMAND}

  Устанавливает режим кэширования для текущего клиента.

Режим	Значение
0, OFF	Не использовать кэширование
1, ON	Кэшировать все запросы, за исключением тех, которые начинаются со слов SELECT SQL_NO_CACHE
2, DEMAND	Кэшировать по требованию только те запросы, которые начинаются со слов SELECT SQL_CACHE

  Этот параметр был добавлен в MySQL 4.0.1. Предпочтительнее использовать параметр QUERY_CACHE_TYPE.

- SQL_QUOTE_SHOW_CREATE = {0 | 1}. Этот параметр контролирует использование обратных одинарных кавычек в именах таблиц, столбцов и индексов в распечатках команды SHOW CREATE TABLE. Значение по умолчанию — 1 (использовать кавычки). Отключение этого параметра установкой его в 0 может пригодиться при применении с другими СУБД или серверами MySQL ниже версии 3.23.6, а также оператора CREATE TABLE, которые не "понимают" символы обратных кавычек. Однако в этом случае необходимо быть полностью уверенным в том, что в именах таблиц не используются зарезервированные слова или они не содержат специальных символов. Этот параметр был добавлен в MySQL 3.23.26.
- SQL_SAFE_UPDATES = {0 | 1}. Если этот параметр установлен в 1, использование операторов UPDATE и DELETE разрешено только в том случае, если изменение записей производится по ключевым полям или используется предложение LIMIT. Этот параметр был добавлен в MySQL 3.22.32.
- SQL_SELECT_LIMIT = {*n* | DEFAULT}. Определяет максимальное число записей, возвращаемых оператором SELECT. Определение этого значения в операторе явным образом задается предложением LIMIT и имеет больший приоритет, чем значение, заданное этим параметром. По умолчанию устанавливается значение "без ограничений". Определение ключевого слова DEFAULT приводит к восстановлению значения по умолчанию, если до этого оно было изменено.
- SQL_SLAVE_SKIP_COUNTER = *n*. Указывает подчиненному серверу пропустить следующие *n* событий на главном сервере. Этот параметр был добавлен в MySQL 3.23.33. Начиная с MySQL 4.0.3 для изменения этого параметра необходимо использовать команду SET GLOBAL.

- SQL_WARNINGS = {0 | 1}. Если этот параметр имеет значение 1, MySQL выдает предупреждение даже для однострочных вставок. Значение по умолчанию — 0; таким образом, обычно счетчик добавленных строк сообщается только для операторов INSERT, которые добавляют сразу несколько строк. Параметр SQL_WARNINGS впервые появился в MySQL 3.22.11 и по умолчанию имеет значение 0.
- TIMESTAMP = {*timestamp_value* | DEFAULT}. Определяет значение TIMESTAMP, которое используется при обработке журнала обновлений.
- UNIQUE_CHECKS = {0 | 1}. Установка этого параметра в 0 или 1 отключает или включает проверку уникальности для вторичных уникальных индексов для таблиц типа InnoDB. Этот параметр был добавлен в MySQL 3.23.52.

SET TRANSACTION ISOLATION LEVEL

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL level
```

Этот оператор устанавливает глобальный (общесерверный) или сеансовый (специфический для клиента) уровень изоляции транзакции или уровень для следующей транзакции в пределах текущего сеанса. Если не задано ни ключевое слово GLOBAL, ни SESSION, оператор устанавливает уровень следующей транзакции. Для установки глобального уровня изоляции требуется привилегия SUPER (до 4.0.2 — это привилегия PROCESS). Глобальное изменение воздействует на клиентов, которые подключаются после сделанного изменения, на тех, которые уже были подключены, изменение никак не влияет.

Уровень транзакции, обозначенный значением *level*, должен иметь одно из следующих значений: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ или SERIALIZABLE.

Этот параметр был добавлен в MySQL 3.23.36. Он действует на таблицы типа InnoDB только начиная с MySQL 3.23.50; до этого таблицы InnoDB постоянно работали в режиме REPEATABLE READ. До MySQL 4.0.5 другим допустимым значением для InnoDB было SERIALIZABLE, и установка любого другого значения *level* приводит к использованию режима REPEATABLE READ. Начиная с MySQL 4.0.5 дескриптор InnoDB также поддерживает уровни изоляции READ COMMITTED и READ UNCOMMITTED.

Этот оператор не воздействует на таблицы типа BDB, так как дескриптор BDB всегда работает на уровне SERIALIZABLE.

SHOW

```
SHOW BINLOG EVENTS [IN 'file_name'] [FROM n]
   [LIMIT [skip_count,] show_count]
SHOW CHARACTER SET
SHOW COLUMN TYPES
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
SHOW CREATE DATABASE db_name
SHOW CREATE TABLE tbl_name
SHOW DATABASES [LIKE 'pattern']
SHOW GRANTS FOR account
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW INNODB STATUS
```

```
SHOW LOGS
SHOW MASTER LOGS
SHOW MASTER STATUS
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW SLAVE HOSTS
SHOW SLAVE STATUS
SHOW STATUS [LIKE 'pattern']
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
SHOW TABLE TYPES
SHOW [OPEN] TABLES [FROM db_name] [LIKE 'pattern']
SHOW [GLOBAL | SESSION ] VARIABLES [LIKE 'pattern']
```

Различные формы оператора SHOW выводят всевозможную информацию о базах данных, таблицах, столбцах, индексах и общем функциональном состоянии сервера. Некоторые его формы включают предложение FROM *db_name*, позволяющее определить имя интересующей базы данных. Если это предложение отсутствует, для извлечения информации используется база данных, установленная по умолчанию. (Начиная с MySQL 4.0 слово IN является синонимом FROM в каждом из этих операторов.)

Некоторые формы содержат также необязательное предложение LIKE '*pattern*', что позволяет с помощью условий ограничить спектр выводимых значений. Задаваемый образец должен быть строкой или SQL-образцом с групповыми символами "%" или "_".

SHOW BINLOG EVENTS

Этот оператор используется на главных серверах репликации для отображения событий в файле двоичного журнала обновления. Его вывод включает следующие столбцы.

- Log_name. Имя файла двоичного журнала обновления.
- Pos. Положение события внутри файла регистрации.
- Event_type. Тип события, например Query для операторов, которые будут выполняться.
- Server_id. Идентификатор сервера, который зарегистрировал это событие.
- Orig_log_pos. Положение события в исходном регистрационном файле на главном сервере.
- Info. Информация о событии, такая как текст оператора для события Query.

Оператор SHOW BINLOG EVENTS был добавлен в MySQL 4.0.

SHOW CHARACTER SET

Отображает список кодировок, которые поддерживаются сервером.

Результаты работы оператора SHOW CHARACTER SET сейчас включают следующие типы информации о каждой кодировке, хотя вполне вероятно, что этот формат в будущем будет расширяться.

- Name. Имя кодировки.
- Id. Внутренний идентификатор кодировки.

- strx_maxlen. Фактор стоимости, имеющий отношение к объему информации, которая будет отведена под внутренние операции преобразования строк при сортировке значений в данной кодировке.
- mb_maxlen. Длина в байтах "самого широкого" символа в кодировке.

Для многобайтовых кодировок это значение будет больше единицы. Для обычных кодировок все символы занимают по одному байту, поэтому для них это значение будет всегда равно единице.

Оператор SHOW CHARACTER SET был добавлен в MySQL 4.1.

SHOW COLUMN TYPES

Этот оператор дает перечень информации о типах столбцов, которые использовались при создании таблиц MySQL.

Выводимые оператором SHOW COLUMN TYPES данные включают следующие столбцы.

- Type. Тип столбца.
- Size. Длина хранимых этим типом данных в байтах.
- Min_Value. Минимальное значение диапазона для типа.
- Max_Value. Максимальное значение диапазона для типа.
- Prec. Точность типа.
- Scale. Коэффициент диапазона типа.
- Nullable. Разрешение значений NULL.
- Auto_Increment. Может ли тип использоваться в последовательностях AUTO_INCREMENT.
- Unsigned. Имеет ли тип атрибут UNSIGNED.
- Zerofill. Имеет ли тип атрибут ZEROFILL.
- Searchable. Может ли данный тип участвовать в операциях поиска.
- Case_Sensitive. Чувствителен ли данный тип к регистру.
- Default. Значение по умолчанию для данного типа. Может задаваться больше одного значения, так как значение по умолчанию зависит от того, был ли столбец объявлен как принимающий значения NULL.
- Comment. Описательный комментарий о типе столбца.

Оператор SHOW COLUMN TYPES был добавлен в MySQL 4.1.

SHOW [FULL] COLUMNS

Оператор SHOW COLUMNS выводит список столбцов в данной таблице. Его синонимом является оператор SHOW FIELDS.

```
SHOW COLUMNS FROM president;
SHOW FIELDS FROM president;
SHOW COLUMNS FROM president FROM sampdb;
SHOW COLUMNS FROM tables_priv FROM mysql LIKE '%priv';
```

Выводимые оператором SHOW COLUMNS данные включают следующие столбцы.

- `Field`. Имя столбца таблицы.
- `Type`. Тип столбца.
- `Null`. Показывает, может ли данный столбец содержать значения NULL. Может иметь значение YES (да) или пустое значение (нет).
- `Key`. Показывает, является ли столбец индексируемым.
- `Default`. Значение по умолчанию столбца.
- `Extra`. Дополнительная информация о столбце.
- `Privileges`. Привилегии, выделенные для столбца. Эта информация указывается только в версиях MySQL 3.23 и выше. Она всегда отображается начиная с 3.23.0 и до 3.23.32, а начиная с версии 3.23.32 она отображается только при указании ключевого слова FULL.
- `Comment`. Значение любого предложения COMMENT, которое было задано в объявлении столбца. Эта информация отображается только начиная с MySQL 4.1 и при указании ключевого слова FULL.

SHOW CREATE DATABASE

Действие этого оператора заключается в отображении оператора CREATE DATABASE, необходимого для создания указанной таблицы.

```
SHOW CREATE DATABASE sampdb;
```

Этот параметр был добавлен в MySQL 4.1.

SHOW CREATE TABLE

Действие этого оператора заключается в отображении оператора CREATE TABLE, который соответствует структуре заданной таблицы.

```
SHOW CREATE TABLE absence;
```

Имена таблиц, столбцов и индексов, полученные с помощью оператора SHOW CREATE TABLE, по умолчанию берутся в кавычки. Добавление кавычек управляется установкой параметра SQL_QUOTE_SHOW_CREATE. (Этот параметр был описан в операторе SET.)

Оператор SHOW CREATE TABLE был добавлен в MySQL 3.23.20.

SHOW DATABASES

Оператор SHOW DATABASES выводит список баз данных, доступных на компьютере сервера.

```
SHOW DATABASES;
SHOW DATABASES LIKE 'test%';
```

Без привилегии SHOW DATABASES пользователь увидит только те базы данных, к которым у пользователя имеется привилегия доступа.

SHOW GRANTS

Оператор SHOW GRANTS отображает информацию о привилегиях, предоставленных определенному пользователю. Для задания пользователя применяется формат '*user_name*'@'*host_name*', где значения *user_name* и *host_name* со-

ответствуют задаваемым для оператора GRANT. Оператор SHOW GRANTS впервые появился в MySQL версии 3.23.4.

```
SHOW GRANTS FOR 'root'@'localhost';
SHOW GRANTS FOR ''@'cobra'.snake.net;
```

Вывод осуществляется в форме операторов GRANT, необходимых для восстановления привилегий учетной записи.

Оператор SHOW GRANTS был добавлен в MySQL 3.23.4.

SHOW INDEX

Оператор SHOW INDEX отображает информацию об индексах таблицы. Его аналогом является оператор SHOW KEYS.

```
SHOW INDEX FROM score;
SHOW KEYS FROM score;
SHOW INDEX FROM sampdb.score;
SHOW INDEX FROM score FROM sampdb;
```

Выводимый результат этого оператора содержит следующие столбцы.

- Table. Имя таблицы, содержащей индексы.
- Non_unique. Этот столбец имеет значение 1, если индекс может содержать дублирующие значения, и 0, если это не допускается.
- Key_name. Имя индекса.
- Seq_in_index. Число столбцов внутри индекса, которые нумеруются с 1.
- Column_name. Имя столбца.
- Collation. Порядок сортировки столбцов внутри индекса. Возможными значениями этого столбца могут быть A (восходящая сортировка), D (нисходящая сортировка) или NULL (столбцы не отсортированы). Нисходящая сортировка еще пока недоступна, но ее планируется ввести в будущем.
- Cardinality. Количество уникальных значений в индексе. Значение этого параметра обновляется посредством запуска программ myisamchk и isamchk с параметром --analyze. Оператор ANALYZE TABLE обновляет это значение для таблиц MyISAM или BDB.
- Sub_part. Длина префикса, если только индексируется префикс столбца. Этот параметр имеет значение NULL, если индексируется целый столбец.
- Packed. Способ упаковки ключа (NULL, если ключ не упаковывается). Этот столбец имеется только начиная с MySQL 3.23.
- Null. Значение YES, если столбец может содержать значения NULL, в противном случае — пустое.
- Index_type. Метод, используемый для индексирования столбца, такой как BTREE или FULLTEXT.
- Comment. Зарезервировано для внутренних комментариев об индексе. Этот столбец имеется только в MySQL 3.23 и выше.

Столбцы Packed и Comment были добавлены в MySQL 3.23.0. Столбцы Null и Index_type добавлены в 4.0.2.

SHOW INNODB STATUS

Этот оператор отображает информацию о внутренней работе дескриптора таблиц InnoDB; добавлен в MySQL 3.23.52.

SHOW LOGS

Этот оператор отображает информацию о регистрационных файлах сервера. Сейчас он используется только для регистрационных журналов BDB.

Выводимый результат этого оператора содержит следующие столбцы.

- `File`. Имя файла журнала регистрации.
- `Type`. Тип журнала регистрации.
- `Status`. Статус журнала регистрации, например параметр `IN USE` `SHOW LOGS` был добавлен в MySQL 3.23.29.

SHOW MASTER LOGS

Этот оператор используется на главных серверах репликации. Он отображает имена двоичных журналов, имеющихся в данный момент на главном сервере. Он может пригодиться перед оператором `PURGE MASTER LOGS` после запуска `SHOW SLAVE STATUS` на каждом из запускаемых подчиненных серверах для определения двоичных журналов, на которые они в данный момент позиционированы.

Этот параметр был добавлен в MySQL 3.23.28.

SHOW MASTER STATUS

Этот оператор используется на главных серверах репликации. Он обеспечивает информацию о состоянии двоичных журналов обновления главных серверов.

Выводимый результат этого оператора содержит следующие столбцы.

- `File`. Имя файла двоичного журнала обновлений.
- `Position`. Текущая позиция, начиная с которой сервер ведет запись журнала.
- `Binlog_do_db`. Разделенный запятыми перечень баз данных, которые реплицируются явным образом в двоичный журнал с помощью параметров `--binlog-do-db`, и пустой, если такие параметры отсутствуют.
- `Binlog_ignore_db`. Разделенный запятыми перечень баз данных, исключенных из двоичного журнала, и пустой, если такие базы данных не заданы.

Оператор `SHOW MASTER STATUS` был добавлен в MySQL 3.23.22.

SHOW PRIVILEGES

Оператор `SHOW PRIVILEGES` отображает присваиваемые привилегии и информацию о предназначении каждой их них.

Выводимый результат этого оператора содержит следующие столбцы.

- `Privilege`. Наименование привилегии.
- `Context`. Применимость привилегии к администрированию сервера, базам данных или таблицам.
- `Comment`. Описание предназначения привилегии.

Привилегия `SHOW PRIVILEGES` была добавлена в MySQL 4.1.

SHOW [FULL] PROCESSLIST

Оператор SHOW PROCESSLIST отображает информацию о потоках, выполняемых внутри сервера. Его вывод содержит следующие столбцы.

- Id. Идентификатор потока для клиента.
- User. Имя клиента, связанного с потоком.
- Host. Компьютер, с которого подключается клиент.
- db. База данных по умолчанию для клиента.
- Command. Команда, исполняемая потоком.
- Time. Количество времени, в течение которого команда выполняется в потоке (в секундах).
- State. Информация об операциях, которые MySQL выполняет в процессе обработки оператора SQL. Значение этого столбца может оказаться полезным для определения проблем в работе MySQL или для вопросов для списка рассылки.
- Info. Исполняемый запрос. Начиная с MySQL 3.23.7 параметр FULL может быть добавлен для просмотра текста запросов в поле Info. Без него отображаются только первые 100 символов.

SHOW SLAVE HOSTS

Этот оператор используется при репликации главного сервера. Он отображает информацию о подчиненных серверах, которые в настоящее время зарегистрированы главным сервером.

Его вывод содержит следующие столбцы.

- Server_id. Идентификатор подчиненного сервера.
- Host. Подчиненный узел.
- User. Учетная запись, которая используется подчиненным сервером для подключения.
- Password. Пароль для учетной записи, который использует подчиненный сервер для подключения.
- Port. Порт, к которому подключается подчиненный сервер.
- Rpl_recovery_rank. Ранг восстановления репликации.
- Master_id. Идентификатор главного сервера.

Столбцы User и Password отображаются, только если главный сервер был запущен с параметром --show-slave-auth-info.

Оператор SHOW SLAVE HOSTS был добавлен в MySQL 4.0.

SHOW SLAVE STATUS

Этот оператор используется на подчиненных серверах и обеспечивает информацию о состоянии сервера репликации.

Его вывод содержит следующие столбцы.

- Master_Host. Имя главного узла или его IP-адрес.
- Master_User. Пользователь для подключения в главному серверу.

- Master_Port. Номер порта для подключения к главному серверу.
- Connect_retry. Количество попыток подключения к главному серверу.
- Master_Log_File. Имя текущего файла двоичного журнала обновления на главном сервере.
- Read_Master_Log_Pos. Текущее положение в двоичном файле регистрации обновлений главного сервера, откуда считывается подчиненный поток ввода-вывода.
- Relay_Log_File. Имя текущего файла регистрации переключений.
- Relay_Log_Pos. Текущее положение в файле регистрации переключений.
- Relay_Master_Log_File. Имя текущего файла регистрации переключений главного сервера.
- Slave_IO_Running. Активность потока ввода-вывода подчиненного сервера.
- Slave_SQL_Running. Активность потока SQL подчиненного сервера.
- Replicate_do_db. Разделенный запятыми список баз данных, которые явным образом реплицируются с помощью параметра --replicate_do_db, или пустое значение, если такой параметр не был задан.
- Replicate_ignore_db. Разделенный запятыми список баз данных, которые явным образом были исключены из схемы репликации с помощью параметра --replicate-ignore-db, или пустое значение, если такой параметр не был задан.
- Last_errno. Номер самого последнего сообщения об ошибке или значение 0, если такого сообщения нет.
- Last_error. Самое последнее сообщение об ошибке или пустое значение, если такого сообщения нет.
- Skip_counter. Число пропускаемых подчиненным сервером событий с главного сервера.
- Exec_master_log_pos. Текущее положение внутри файла двоичного журнала регистрации обновлений главного сервера, где выполняется поток подчиненного SQL.
- Relay_log_space. Полный размер файлов регистрации переключений.

Оператор SHOW SLAVE STATUS был добавлен в MySQL 3.23.22.

SHOW STATUS

Оператор SHOW STATUS выводит на экран значения переменных состояния сервера. В MySQL 3.23.0 появилась возможность добавить предложение LIKE '*pattern*' для отображения переменных, имена которых соответствуют шаблону.

Ниже дается перечень наиболее общеупотребительных переменных. Переменные счетчиков операторов, кэша запросов и протокола SSL даны в отдельных группах после этого.

- Aborted_clients. Счетчик клиентских соединений, прерванных в результате неправильного закрытия соединения клиентом.

- Aborted_connects. Счетчик неудачных попыток подключения к серверу.
- Bytes_received. Счетчик принятых байтов от всех клиентов. Этот параметр был добавлен в MySQL 3.23.7.
- Bytes_sent. Счетчик отосланных байтов всем клиентам. Этот параметр был добавлен в MySQL 3.23.7.
- Connections. Счетчик общих попыток подключения к серверу (удачных и неудачных). Если это значение достаточно велико, возможно, есть смысл рассмотреть возможность установления для данного клиента устойчивого соединения.
- Created_tmp_disk_tables. Счетчик временных таблиц, созданных на диске в процессе обработки запроса. Эта переменная была добавлена в MySQL 3.23.24.
- Created_tmp_files. Счетчик временных файлов, созданных сервером. Эта переменная была добавлена в MySQL 3.23.28.
- Created_tmp_tables. Счетчик временных таблиц, созданных в памяти в процессе обработки запроса.
- Delayed_errors. Счетчик ошибок, возникших в процессе обработки строк INSERT DELAYED.
- Delayed_insert_threads. Счетчик обрабатываемых запросов INSERT DELAYED.
- Delayed_writes. Счетчик строк, записанных с помощью оператора INSERT DELAYED.
- Flush_commands. Счетчик выполненных команд FLUSH.
- Handler_commit. Счетчик запросов, выполняемых транзакцией. Эта переменная была добавлена в MySQL 4.0.2.
- Handler_delete. Счетчик запросов на удаление строк из таблицы.
- Handler_read_first. Счетчик запросов на чтение первой строки таблицы.
- Handler_read_key. Счетчик запросов на чтение строк, построенных на значении индекса.
- Handler_read_next. Счетчик запросов на чтение следующей строки в порядке индексов.
- Handler_read_prev. Счетчик запросов для предыдущей строки при ниспадающем порядке индексирования. Эта переменная была добавлена в MySQL 3.23.6.
- Handler_read_rnd. Счетчик запросов на чтение строки, основанных на ее позиции.
- Handler_read_rnd_next. Количество запросов, необходимых для чтения следующей строки. Если это значение превышает сумму значений всех предыдущих параметров, следовательно, выполняется большое число запросов, требующих полного сканирования таблицы или неправильно использующих индексы. Эта переменная была добавлена в MySQL 3.23.6.

- `Handler_rollback`. Счетчик запросов, необходимых для отмены транзакции. Эта переменная была добавлена в MySQL 4.0.2.
- `Handler_update`. Счетчик запросов на обновление строк таблицы.
- `Handler_write`. Счетчик запросов на вставку строк в таблицу.
- `Key_block_used`. Счетчик используемых в кэше индексов блоков.
- `Key_read_requests`. Счетчик запросов на чтение блока из кэша индексов.
- `Key_reads`. Счетчик физических операций чтения блоков индексов с диска.
- `Key_write_requests`. Счетчик запросов на запись блока в кэш индексов.
- `Key_writes`. Счетчик физических операций записи блоков индексов на диск.
- `Max_used_connections`. Максимальное число соединений, которые были открыты одновременно.
- `Not_flushed_delayed_rows`. Счетчик строк, ожидающих вставки в запросах `INSERT DELAYED`.
- `Not_flushed_key_blocks`. Счетчик блоков в кэше ключей, которые были изменены, но еще не записаны на диск.
- `Opened_tables`. Счетчик открытых таблиц. При большом значении этого числа рекомендуется увеличить размер кэша таблиц.
- `Open_files`. Счетчик открытых файлов.
- `Open_streams`. Счетчик открытых потоков. Потоком называется файл, открытый с помощью функции `fopen()` (применяется только к файлам журналов).
- `Open_tables`. Счетчик открытых таблиц.
- `Questions`. Счетчик запросов, полученных сервером (как успешно обработанных, так и отвергнутых).
- `Rpl_status`. Статус безопасности репликации. Эта переменная была добавлена в MySQL 4.0.0, но до сих пор не используется.
- `Select_full_join`. Счетчик запросов-объединений, выполненных без использования индексов. Эта переменная была добавлена в MySQL 3.23.25.
- `Select_full_range_join`. Счетчик запросов-объединений, выполненных с использованием поиска по диапазону в таблице, на которую делается ссылка. Эта переменная была добавлена в MySQL 3.23.25.
- `Select_range`. Счетчик запросов-объединений, выполненных с использованием поиска по диапазону в первой таблице. Эта переменная была добавлена в MySQL 3.23.25.
- `Select_range_check`. Счетчик запросов-объединений, выполненных с использованием поиска по диапазону для выборки строк из вторичной таблицы. Эта переменная была добавлена в MySQL 3.23.25.
- `Select_scan`. Счетчик запросов-объединений, выполненных с использованием полного поиска по первой таблице. Эта переменная была добавлена в MySQL 3.23.25.

- *Slave_open_temp_tables. Счетчик временных таблиц, открытых подчиненным потоком.* Эта переменная была добавлена в MySQL 3.23.29.
- Slave_running. Функционирует ли данный сервер в качестве подчиненного сервера, который в данный момент подключен к серверу. Эта переменная была добавлена в MySQL 3.23.16.
- Slow_launch_threads. Счетчик потоков, на создание которых потребовалось более чем slow_launch_time секунд. Эта переменная была добавлена в MySQL 3.23.15.
- Slow_queries. Счетчик запросов, которые будут обрабатываться дольше, чем long_query_time секунд.
- Sort_merge_passes. Счетчик проходов, сделанных алгоритмом сортировки. Эта переменная была добавлена в MySQL 3.23.28.
- Sort_range. Счетчик операций сортировки, выполненных с использованием диапазона. Эта переменная была добавлена в MySQL 3.23.25.
- Sort_rows. Счетчик отсортированных строк. Эта переменная была добавлена в MySQL 3.23.25.
- Sort_scan. Счетчик операций, выполненных с использованием полного сканирования таблицы. Эта переменная была добавлена в MySQL 3.23.25.
- Table_locks_immediate. Счетчик запросов на блокировку таблицы, которые можно удовлетворить немедленно. Эта переменная была добавлена в MySQL 3.23.33.
- Table_locks_waited. Счетчик запросов на блокировку таблицы, которые могут быть удовлетворены только после определенного периода ожидания. Эта переменная была добавлена в MySQL 3.23.33.
- Threads_connected. Счетчик открытых текущих соединений.
- Threads_created. Полное количество потоков, созданных для поддержания соединений с клиентом. Эта переменная была добавлена в MySQL 3.23.31.
- Threads_running. Счетчик процессов, находящихся в активном состоянии.
- Uptime. Время работы сервера в секундах.

Переменные состояния счетчиков операторов

Начиная с MySQL 3.23.47 сервер обрабатывает множество переменных состояния, служащих счетчиками, предназначенными для хранения информации о том, сколько раз выполнялись определенные виды операторов (команд). Существуют десятки таких переменных, и у них всех имеются достаточно информативные имена, поэтому отдельно они здесь не упоминаются. Имя каждой переменной счетчиков операторов начинается с Com_ и имеет суффикс, отображающий тип оператора, к которому относится счетчик. Например, Com_select и Com_drop_table отображают количество выполненных сервером операторов SELECT и DROP TABLE.

Переменные состояния кэша запросов

Вниманию читателя предлагается информация о действии переменных кэша запросов. Эти переменные были добавлены в MySQL 4.0.1, одновременно с добавлением самого кэша запросов.

- `Qcache_free_blocks`. Число свободных блоков памяти в кэше запросов.
- `Qcache_free_memory`. Объем свободной памяти для кэша запросов.
- `Qcache_hits`. Количество "попаданий" в кэш запросов, т.е. сколько запросов было удовлетворено запросами, находящимися в кэше.
- `Qcache_inserts`. Количество запросов, зарегистрированных в кэше запросов.
- `Qcache_not_cached`. Количество запросов, которые оказались некэшируемыми или для которых кэширование было подавлено с помощью ключевого слова `SQL_NO_CACHE`.
- `Qcache_queries_in_cache`. Количество запросов, зарегистрированных в кэше на данный момент.
- `Qcache_total_blocks`. Суммарное количество блоков памяти, отведенных под кэш запросов.

Кроме того, оператор `SHOW VARIABLES` также выводит значение нескольких переменных запросов, относящихся к кэшу; их имена начинаются с `query_cache`.

Переменные состояния SSL-соединений

Следующие переменные предоставляют информацию об управляющем коде протокола SSL. Многие из них отражают состояние текущего соединения и будут пустыми, если данное соединение является небезопасным. Эти переменные были добавлены в MySQL 4.0, когда, собственно, и была добавлена поддержка SSL. Однако он не будут доступны, если поддержка SSL не встроена в сервер.

- `Ssl_accept_renegotiates`. Счетчик повторных стартов соединений в серверном режиме.
- `Ssl_accepts`. Счетчик "хендшейков" SSL/TLS в серверном режиме.
- `Ssl_callback_cache_hits`. Счетчик успешно найденных во внешнем кэше сеансов в серверном режиме.
- `Ssl_cipher`. Шифр протокола SSL для текущего соединения (пустая, если нет никакого шифра). Эту переменную можно использовать для того, чтобы определить, является ли данное соединение закодированным.
- `Ssl_cipher_list`. Перечень имеющихся шифров SSL.
- `Ssl_client_connects`. Счетчик стартов "хендшейков" SSL/TLS в клиентском режиме.
- `Ssl_connect_renegotiates`. Счетчик повторных стартов соединений в клиентском режиме.
- `Ssl_ctx_verify_depth`. Глубина проверки контекста SSL.

- Ssl_ctx_verify_mode. Режим проверки контекста SSL.
- Ssl_default_timeout. Стандартный тайм-аут сеанса SSL.
- Ssl_finished_accepts. Счетчик успешных сеансов SSL/TLS в серверном режиме.
- Ssl_finished_connects. Счетчик успешных сеансов SSL/TLS в клиентском режиме.
- Ssl_session_cache_hits. Счетчик сеансов SSL, обнаруженных в кэше сеанса.
- Ssl_session_cache_misses. Счетчик сеансов SSL, не обнаруженных в кэше сеанса.
- Ssl_session_cache_mode. Тип кэширования SSL, используемого сервером.
- Ssl_session_cache_overflows. Счетчик сеансов SSL, удаленных из кэша из-за его переполнения.
- Ssl_session_cache_size. Счетчик сеансов SSL, которые могут храниться в кэше сеанса SSL.
- Ssl_session_cache_timeouts. Счетчик сеансов SSL, завершенных по тайм-ауту.
- Ssl_sessions_reused. Возможно ли повторное использование сеанса из раннего сеанса.
- Ssl_used_session_cache_entries. Счетчик сеансов в текущем кэше сеансов.
- Ssl_verify_depth. Глубина проверки SSL.
- Ssl_verify_mode. Режим проверки SSL.
- Ssl_version. Версия протокола соединения.

SHOW TABLE STATUS

Оператор SHOW TABLE STATUS выводит описательную информацию о таблицах базы данных. Этот оператор впервые появился в MySQL 3.23.

```
SHOW TABLE STATUS;
SHOW TABLE STATUS FROM sampdb;
SHOW TABLE STATUS FROM mysql LIKE '%priv';
```

Выводимые данные оператора SHOW TABLE STATUS содержат следующие столбцы.

- Name. Имя таблицы.
- Type. Тип таблицы. Возможными значениями этого столбца могут быть NISAM (ISAM), MyISAM, HEAP или InnoDB.
- Row_format. Формат хранения строк. Возможными значениями этого столбца могут быть Fixed (фиксированная длина), Dynamic (переменная длина) или Compressed.
- Rows. Число строк в таблице. Для таблиц некоторых типов, таких как BDB и InnoDB, это только приблизительный счетчик.

- Avg_row_length. Среднее число байт, занимаемых строками таблицы.
- Data_length. Реальный размер файла данных таблиц (в байтах).
- Max_data_length. Максимальный размер (в байтах), до которого может увеличиваться файл данных таблицы.
- Index_length. Реальный размер файла индекса (в байтах).
- Data_free. Число неиспользуемых байт файла данных. Если этот столбец имеет очень большое значение, рекомендуется запустить для таблицы оператор OPTIMIZE TABLE.
- Auto_increment. Следующее значение, которое будет сгенерировано для столбца AUTO_INCREMENT.
- Create_time. Время создания таблицы.
- Update_time. Время последнего изменения таблицы.
- Check_time. Время последней проверки и отладки таблицы MyISAM с помощью программы myisamchk. Если таблица никогда не проверялась или не исправлялась, этот столбец имеет значение NULL.
- Charset. Кодировка таблицы.
- Create_options. Дополнительные параметры, определенные в предложении *table_options* оператора CREATE TABLE при создании таблицы.
- Comment. Текст всех определенных в момент создания комментариев. Для таблиц InnoDB этот столбец задает определения внешнего ключа, а также отображает объем пустого пространства в табличном пространстве InnoDB.

Оператор SHOW TABLE STATUS был добавлен в MySQL 3.23.0, а столбец harset – в MySQL 4.1.

HOW TABLE TYPES

Оператор SHOW TABLE TYPES дает перечень дескрипторов, поддерживаемых ервером.

Его вывод содержит следующие столбцы.

- Type. Тип таблицы (MyISAM, InnoDB и т.д.).
- Support. Уровень поддержки: YES – для поддерживаемых, NO – для неподдерживаемых, DISABLED – для поддерживаемых, но отключенных при работе, и DEFAULT – для отображения типа таблицы, являющегося стандартным типом.
- Comment. Описательный текст о типе таблицы.

Оператор SHOW TABLE TYPES был добавлен в MySQL 4.1.

HOW [OPEN] TABLES

Оператор SHOW TABLES выводит список таблиц базы данных, не являющихся ременными.

```
HOW TABLES;
HOW TABLES FROM sampdb;
HOW TABLES FROM mysql LIKE '%priv'
```

Начиная с MySQL 3.23.33 может использоваться оператор SHOW OPEN TABLES. Он отображает перечень открытых таблиц, зарегистрированных в кэше таблиц. Предложения FROM и LIKE для этой формы оператора игнорируются.

```
SHOW OPEN TABLES;
```

SHOW [GLOBAL | SESSION] VARIABLES

Оператор SHOW VARIABLES выводит список переменных сервера и их значения.

```
SHOW VARIABLES;
SHOW VARIABLES LIKE '%thread%';
```

Начиная с MySQL 4.0.3 сервер может отображать значения переменных на глобальном (общесерверном) и сеансовом (специфическом для клиента) уровнях. По умолчанию оператор SHOW отображает сеансовое значение для любого заданного или глобального значения, если сеансового значения не существует. Для отображения глобальных или сеансовых значений явным образом можно воспользоваться соответствующим индикатором:

```
SHOW GLOBAL VARIABLES [LIKE 'pattern'];
SHOW SESSION VARIABLES [LIKE 'pattern'];
```

SESSION является синонимом LOCAL. Кроме того, возможно определить значения динамических переменных с использованием SELECT:

```
SELECT @@GLOBAL.table_type, @@SESSION.table_type, @@LOCAL.table_type;
```

Использование SELECT имеет определенное преимущество, позволяющее в определенной мере упростить манипулирование результатом запросов.

Вывод SHOW VARIABLES включает переменные, перечень которых приводится в следующем списке. Если не указано другого, перечисленные здесь переменные уже использовались начиная с MySQL 3.22.0. (Многие переменные InnoDB были добавлены в MySQL 3.23.29 с характерными именами, которые начинались с innobase_, а затем были переименованы в 3.23.37, чтобы начинаться с innodb_.) Некоторые переменные представлены только при определенных настройках. Например, многие из тех, которые начинаются с bdb_, присутствуют только тогда, когда установлен дескриптор BDB.

Начиная с MySQL 4.0.3 появилась возможность динамического изменения нескольких серверных переменных. (Как это можно сделать, описано в описании оператора SET.) Для переменных, которые можно изменять таким образом, за именем следует слово global или session, заключенное в кавычки. Это делается для того, чтобы показать уровень на котором модифицируется переменная. (До MySQL 4.0.3 эти индикаторы не имеют значения.)

- back_log. Максимальное количество задержанных запросов на соединение, которые могут быть поставлены в очередь во время обработки параллельных запросов.
- ansi_mode. Был ли сервер запущен с параметром --ansi. Эта переменная была добавлена в 3.23.6 и удалена в 3.23.41, одновременно с добавлением переменной sql_mode.
- basedir. Путь к корневому каталогу инсталляции MySQL.

- bdb_cache_size. Размер буфера, используемого дескриптором таблиц BDB для кэширования строк данных и индексов. Запуск MySQL --skip-bdb с целью отключения дескриптора BDB устанавливает значение этой переменной в 0, уменьшая тем самым использование памяти. Эта переменная была добавлена в MySQL 3.23.14.
- bdb_home. Корневой каталог для таблиц типа BDB; обычно это то же значение, что и datadir. Эта переменная была добавлена в MySQL 3.23.14.
- bdb_log_buffer_size. Размер буфера, используемого для журнала регистрации транзакций BDB. Эта переменная была добавлена в MySQL 3.23.21.
- bdb_logdir. Путь к каталогу, в котором дескриптор BDB записывает регистрационные файлы. Эта переменная была добавлена в MySQL 3.23.14.
- bdb_max_lock. Максимальное количество одновременных блокировок, которые могут устанавливаться на таблицах типа BDB. Эта переменная была добавлена в MySQL 3.23.29.
- bdb_shared_data. Отображает, был ли дескриптор BDB запущен в мультипроцессном режиме. Эта переменная была добавлена в MySQL 3.23.29.
- bdb_tmpdir. Каталог, в котором дескриптор BDB создает временные файлы. Эта переменная была добавлена в MySQL 3.23.14.
- bdb_version. Номер версии дескриптора BDB. Эта переменная была добавлена в MySQL 3.23.31.
- binlog_cache_size (*глобальный*). Размер кэша, который используется для хранения операторов SQL, являющихся частью транзакции, до того, как они были сохранены в двоичном журнале. (Так происходит при выполнении транзакции. При откате транзакции эти операторы отменяются.) Эта переменная была добавлена в MySQL 3.23.29.
- bulk_insert_buffer_size (*глобальный, сеансовый*). Размер кэша, используемого для оптимизации поблочных добавлений в таблицы типа MyISAM. Включает операторы LOAD DATA, многострочные операторы INSERT и операторы INSERT INTO ... SELECT. Установка этого значения в 0 отключает оптимизацию.

  Эта переменная была добавлена как переменная myisam_bulk_insert_tree_size в MySQL 4.0.0 и переименована в bulk_insert_buffer_size в 4.0.3.
- character_set. Имя стандартной кодировки. Эта переменная была добавлена в MySQL 3.23.3.
- character_sets. Перечень поддерживаемых кодировок. Значение представляет собой набор названий кодировок, разделенных запятыми. Эта переменная была добавлена в MySQL 3.23.15.
- concurrent_insert (*глобальный*). Позволяет ли сервер производить запросы INSERT на таблицах MyISAM, пока запросы SELECT активны. Активизирована по умолчанию, но отключается с помощью параметра --skip-concurrent-insert. Эта переменная была добавлена в MySQL 3.23.7.

- connect_timeout (*глобальный*). Время в секундах, на протяжении которого mysqld будет ожидать пакеты во время первоначального "хендшейка" соединения.
- convert_character_set (*сеансовый*). Набор символов, который используется для преобразования символов, полученных от клиентов. Эта переменная была добавлена в MySQL 4.0.3.
- datadir. Путь к каталогу данных MySQL.
- delayed_insert_limit (*глобальный*). Количество строк из операторов INSERT DELAYED, которые будут добавлены в таблицу до того, как будет сделана проверка поступления новых операторов SELECT. Если таковые есть, операция добавления будет задержана до завершения выборки.
- delayed_insert_timeout (*глобальный*). Когда дескриптор операций INSERT DELAYED завершает вставку очередных строк, он ожидает столько времени появились ли какие-либо строки INSERT DELAYED. Если да — он обрабатывает их; в другом случае — его работа на этом завершается.
- delayed_queue_size (*глобальный*). Количество строк, которые могут быть поставлены в очередь операторами INSERT DELAYED. При переполнении очереди операторы INSERT DELAYED блокируются до тех пор, пока в очереди не появится свободное место.
- delay_key_write (*глобальный*). Поддержка сервером задержки записи ключей для таблиц MyISAM, созданных с помощью параметра DELAY_KEY_WRITE. Эта переменная может принимать три значения. При значении ON (по умолчанию) сервер поддерживает параметр DELAY_KEY_WRITE для таблиц, объявленных с таким параметром. (Запись ключей задерживается при значении DELAY_KEY_WRITE=1, а для таблиц, объявленных с DELAY_KEY_WRITE=0, не задерживается.) Значение OFF означает, что запись ключей не задерживается ни для одной из таблиц, независимо от того, как она была объявлена. Значение ALL приводит к задержке записи ключей для всех таблиц, независимо от того, как она была объявлена.

  Эта переменная была добавлена как delayed_key_write в MySQL 3.23.5 и переименована в delay_key_write в 3.23.8.
- flush (*глобальный*). Происходит ли сброс таблиц сервером после каждого обновления.

  Эта переменная была добавлена в MySQL 3.22.9, но использовалась в ОС Windows 3.22.9–3.22.11.
- flush_time (*глобальный*). При ненулевом значении этой переменной таблицы закрываются для сброса на диск изменений, ожидающих в очереди, через каждые flush_time секунд. По умолчанию принимается значение 0 для UNIX и 1800 (30 минут) для Windows; для изменения этого значения используется параметр --flush. Эта переменная была добавлена в MySQL 3.22.18.
- ft_boolean_syntax. Перечень операторов, поддерживающих поиск FULLTEXT и использующих режим IN BOOLEAN MODE. Эта переменная была добавлена в MySQL 4.0.1.

- ft_max_word_len. Максимальная длина слов, которые могут быть включены в индекс FULLTEXT. Более длинные слова игнорируются. При изменении значения этой переменной потребуется перестроить все индексы FULLTEXT, представленные в таблицах, обрабатываемых сервером. Эта переменная была добавлена в MySQL 4.0.0.
- ft_max_word_len_for_sort. Максимальная длина слов, которые можно считать достаточно короткими для того, чтобы воспользоваться быстрым методом создания индексов, при использовании операторами ALTER TABLE, CREATE INDEX и REPAIR TABLE для создания индексов FULLTEXT. Более длинные слова, вставляются с использованием более медленного метода. Эта переменная была добавлена в MySQL 4.0.0.
- ft_min_word_len. Минимальная длина слов, которые могут быть включены в индекс FULLTEXT. Слова, длина которых короче этого значения, игнорируются. После изменения этого значения необходимо перестраивать все индексы FULLTEXT, которые обрабатываются сервером. Эта переменная была добавлена в MySQL 4.0.0.
- have_bdb. Имеет значение YES, если присутствует и активизирован дескриптор таблиц BDB; значение DISABLED — если дескриптор имеется, но отключен; значение NO — если дескриптор отсутствует. Эта переменная была добавлена в MySQL 3.23.30.
- have_innodb. Имеет значение YES, если присутствует и активизирован дескриптор таблиц InnoDB; значение DISABLED — если дескриптор имеется, но отключен; значение NO — если дескриптор отсутствует.

  Эта переменная была добавлена в MySQL 3.23.30 как have_innobase и переименована в have_innodb в MySQL 3.23.37.
- have_isam. Имеет значение YES, если присутствует и активизирован дескриптор таблиц ISAM; значение DISABLED — если дескриптор имеется, но отключен; значение NO — если дескриптор отсутствует. Эта переменная была добавлена в MySQL 3.23.30.
- have_openssl. Имеет значения YES или NO для отражения поддержки сервером зашифрованных клиентских соединений с использованием SSL. Эта переменная была добавлена в MySQL 3.23.43, но не несла никакой функциональной нагрузки до версии 4.0.0, когда была добавлен поддержка SSL.
- have_query_cache. Имеется ли кэш запросов. Эта переменная была добавлено в MySQL 4.0.2.
- have_raid. Отражает, поддерживаются ли параметры CREATE TABLE, работающие с массивами RAID. Эта переменная была добавлена в MySQL 3.23.30.
- have_symlink. Активизирована ли поддержка символических ссылок. Эта переменная была добавлена в MySQL 4.0.0.
- init_file. Имя файла, содержащего операторы SQL, которые должны выполняться при запуске сервера (пустой, если ничего). Использование этого файла определяется использованием параметра --init-file. Эта переменная была добавлена в MySQL 3.23.2.

- `innodb_additional_mem_pool_size`. Размер буфера памяти, отведенного для хранения внутренних структур данных InnoDB.

  Эта переменная была добавлена в MySQL 3.23.37.

- `innodb_buffer_pool_size`. Размер кэша InnoDB для буферизации табличных данных и индексов. Эта переменная была добавлена в MySQL 3.23.37.
- `innodb_data_file_path`. Спецификации для файлов, составляющих табличное пространство. Эта переменная была добавлена в MySQL 3.23.37.
- `innodb_data_home_dir`. Путь к каталогу, в котором расположены компоненты табличного пространства InnoDB. Эта переменная была добавлена в MySQL 3.23.37.
- `innodb_fast_shutdown`. Будет ли использовать дескриптор InnoDB быстрый метод отключения, который пропускает некоторые операции, обычно им выполняемые. Эта переменная была добавлена в MySQL 3.23.44.
- `innodb_file_io_threads`. Количество потоков ввода-вывода, используемых InnoDB. Эта переменная была добавлена в MySQL 3.23.37.
- `innodb_flush_log_at_trx_commit`. Этот параметр управляет обновлением журнала InnoDB после завершения транзакции.

Режим	Значение
0	Запись в журнал с периодичностью один раз в секунду и сброс на диск
1	Запись в журнал после выполнения каждого оператора `commit` и сброс на диск
2	Запись в журнал после выполнения каждого оператора `commit`; сброс на диск производить один раз в секунду

  Эта переменная была добавлена в MySQL 3.23.37.

- `innodb_flush_method`. Метод, используемый InnoDB для регистрации обновления. Эта переменная была добавлена в MySQL 3.23.39.
- `innodb_force_recovery`. Нормальное значение 0, но может принимать значение от 1 до 6, того, чтобы сервер запускался после краха, даже если восстановление таблиц типа InnoDB закончилось неуспешно. Описание использования этой переменной приводится в приложении B, "Операторы и функции". Эта переменная была добавлена в MySQL 3.23.44.
- `innodb_lock_wait_timeout`. Время ожидания дескриптором InnoDB завершения транзакции. Если завершение невозможно, дескриптор InnoDB делает откат транзакции. Эта переменная была добавлена в MySQL 3.23.37.
- `innodb_log_arch_dir`. Эта переменная не используется. Она была добавлена в MySQL 3.23.37.
- `innodb_log_archive`. Эта переменная не используется. Она была добавлена в MySQL 3.23.37.
- `innodb_log_buffer_size`. Размер буфера журнала регистрации транзакций. Эта переменная была добавлена MySQL 3.23.37.
- `innodb_log_files_in_group`. Количество журналов регистрации, сопровождаемых дескриптором InnoDB. Полный размер журнала InnoDB опреде-

ляется произведением переменных innodb_log_files_in_group и innodb_log_file_size. Эта переменная была добавлена в MySQL 3.23.37.

- innodb_log_file_size. Размер каждого файла регистрации InnoDB. Полный размер журнала InnoDB определяется произведением переменных innodb_log_files_in_group и innodb_log_file_size. Эта переменная была добавлена в MySQL 3.23.37.
- innodb_log_group_home_dir. Каталог, в котором записываются файлы журналов InnoDB. Эта переменная была добавлена в MySQL 3.23.37.
- innodb_mirrored_log_groups. Количество сопровождаемых групп регистрационных файлов InnoDB. Значение этой переменной всегда должно равняться 1. Эта переменная была добавлена в MySQL 3.23.37.
- innodb_thread_concurrency. Предельное значение количества потоков, которые дескриптор InnoDB пытается обслуживать. Эта переменная была добавлена в MySQL 3.23.44.
- interactive_timeout (*глобальный, сеансовый*). Время простоя соединения интерактивного клиента (в секундах), после которого сервер имеет право закрыть такое соединение. Для неинтерактивных клиентов вместо этой переменной используется значение переменной wait_timeout. Эта переменная была добавлена в MySQL 3.23.7.
- join_buffer_size (*глобальный, сеансовый*). Размер буфера полного объединения (т.е. буфер для объединений, которые выполняются без использования индексов).

  Эта переменная до MySQL 3.23 называлась join_buffer.
- key_buffer_size (*глобальный*). Размер буфера, используемого для хранения индексных блоков. Этот буфер распределяется между потоками обработки соединений.

  До MySQL 3.23 эта переменная носила название key_buffer.
- language. Язык вывода сообщений об ошибках. Значение может быть именем языка или путем к каталогу, содержащему файлы языков.
- large_files_support. Был ли сервер построен с поддержкой обработки файлов большого размера. Эта переменная была добавлена в MySQL 3.23.28.
- local_infile (*глобальный*). Разрешение режима LOCAL для операторов LOAD DATA. Эта переменная была добавлена в MySQL 4.0.3.
- locked_in_memory. Заблокирован ли сервер в памяти. Эта переменная была добавлена в MySQL 3.23.25.
- log. Активизирована ли регистрация запросов в журнал.
- log_bin. Активизирован ли двоичный журнал обновлений. Эта переменная была добавлена в MySQL 3.23.14.
- log_slave_updates. Активизирована ли регистрация обновлений, полученных от главного сервера, на подчиненном сервере репликации. Регистрация обновлений на подчиненном сервере позволяет подчиненному серверу служить главным сервером для другого подчиненного сервера в конфи-

гурации цепной репликации. Эта переменная была добавлена в MySQL 3.23.17.

- `log_slow_queries`. Активизирована ли регистрация медленных обновлений в журнал. Эта переменная была добавлена в MySQL 4.0.2.
- `log_update`. Активизирована ли регистрация обновлений в журнал регистрации ошибок. Эта переменная была добавлена в MySQL 3.22.18.
- `log_warnings` (*глобальный, сеансовый*). Активизирована ли регистрация некритических предупреждений в журнал регистрации ошибок. Эта переменная была добавлена в MySQL 4.0.3.
- `long_query_time` (*глобальный, сеансовый*). Время в секундах, определяющее параметры "медленного" запроса. Любой запрос, на выполнение которого требуется больше времени, приводит к приращению счетчика `Slow_queries`. Кроме того, если ведется журнал медленных запросов, такой запрос будет записан в журнал.
- `lower_case_table_names`. Переводятся ли все имена таблиц, хранимые на диске, в нижний регистр (и, начиная с MySQL 4.0.2, имена баз данных). Эта переменная была добавлена в MySQL 3.23.6.
- `low_priority_updates` (*глобальный, сеансовый*). Был ли сервер запущен с параметром `--low-priority-updates` для того, чтобы обновления получили меньший приоритет, чем выборки. Эта переменная была добавлена в MySQL 3.22.5.
- `max_allowed_packet` (*глобальный, сеансовый*). Максимальный размер буфера, используемого для обмена данными между сервером и клиентом. Первоначально под буфер распределяется `net_buffer_length` байт, но в случае необходимости он может увеличиваться до `max_allowed_packet` байт. Начиная с MySQL 4 максимальное значение для `max_allowed_packet` составляет 1 Гбайт, а до MySQL 4 — 16 Мбайт.
- `max_binlog_cache_size` (*глобальный*). Максимальный размер кэша двоичного журнала. Таким образом, ограничивается суммарный размер операторов, составляющих транзакцию. Эта переменная была добавлена в MySQL 3.23.29.
- `max_binlog_size` (*глобальный*). Максимальный размер двоичного журнала. При достижении этого размера журнал обновляется. Допустимые значения лежат в диапазоне 1 Кбайт–1 Гбайт. Эта переменная была добавлена в MySQL 3.23.
- `max_connections` (*глобальный*). Максимальное количество разрешенных подключений клиентов.
- `max_connect_errors` (*глобальный*). Количество разрешенных неудачных соединений с узла перед тем, как узел будет заблокирован от дальнейших попыток соединения. Это делается в предположении, что с такого узла возможна попытка доступа со стороны взломщика. Оператор `FLUSH HOSTS` или команда программы `mysqladmin flush-hosts` может использоваться для очистки кэша данного узла для повторной актуализации заблокированного узла.

- max_delayed_threads (*глобальный*). Максимальное количество чтений, которые создаются для обработки операторов INSERT DELAYED. Любой такой оператор, полученный в то время, когда максимальное количество дескрипторов уже задействовано, будет рассмотрен как оператор, не относящийся к разряду операторов DELAYED.

  Этот параметр был добавлен как max_delayed_insert_threads в MySQL 3.22.15 и переименован в max_delayed_threads в 3.23.0.

- max_heap_table_size (*глобальный, сеансовый*). Максимально разрешенный размер таблиц HEAP. Эта переменная может использоваться для предотвращения использования сервером излишней памяти. Эта переменная была добавлена в MySQL 3.23.0.

- max_join_size (*глобальный, сеансовый*). При выполнении объединения оптимизатор MySQL оценивает, сколько комбинаций строк ему придется перебрать. Если полученная оценка превышает max_join_size строк, возвращается ошибка. Этой переменной можно воспользоваться, если среди пользователей обнаружилась тенденция создавать непродуманные запросы SELECT, возвращающие огромное количество строк.

- max_sort_length (*глобальный, сеансовый*). Значения BLOB или TEXT сортируются с учетом первых max_sort_length байтов каждого значения.

- max_tmp_tables (*глобальный, сеансовый*). Максимальное число временных таблиц, которые может одновременно открыть клиент. Эта переменная была добавлена в MySQL 3.23.0, но сейчас она не используется.

- max_user_connections (*глобальный*). Максимальное количество одновременно открытых подключений, разрешенных для одной учетной записи. Значение по умолчанию – 0, что обозначает "без ограничений". Число соединений в расчете на одно соединение привязано к значению max_connections. Эта переменная была добавлена в MySQL 3.23.34.

  Для задания ограничений на соединение для учетных записей используется оператор GRANT.

- max_write_lock_count (*глобальный*). Достигнув такого количество блокировок записей, сервер начинает пропускать только те запросы, которым требуется блокировка записей с большим приоритетом, чем обычно. Эта переменная была добавлена в MySQL 3.23.7.

- myisam_bulk_insert_tree_size. См. описание переменной bulk_insert_buffer_size.

- myisam_max_extra_sort_file_size (*глобальный, сеансовый*). Это значение используется дескриптором MyISAM для помощи в использовании более медленного, но и более надежного метода создания индекса.

  Эта переменная была добавлена в MySQL 3.23.37. Начиная MySQL 4.0.3 это значение задается в байтах, а до этого – в мегабайтах.

- myisam_max_sort_file_size (*глобальный, сеансовый*). Перестройка индексов таблиц MyISAM при выполнении таких операторов, как REPAIR TABLE, ALTER TABLE или LOAD DATA, может использовать временный файл или

кэш ключа. Значение этой переменной определяет используемый метод; если временный файл превышает этот размер, используется кэш ключа.

Эта переменная была добавлена в MySQL 3.23.37. Начиная MySQL 4.0.3 это значение задается в байтах, а до этого — в мегабайтах.

- `myisam_recover_options`. Значение параметра `--myisam-recover`, с которым был запущен сервер для задания режима автопроверки MyISAM. Эта переменная была добавлена в MySQL 3.23.36.
- `myisam_sort_buffer_size` (*глобальный, сеансовый*). Размер буфера, который распределяется при сортировке индекса таблиц MyISAM во время выполнения операций `ALTER TABLE`, `CREATE INDEX` и `REPAIR TABLE`. Эта переменная была добавлена в MySQL 3.23.16.
- `named_pipe`. Включена ли поддержка именованных каналов. Эта переменная используется для серверов в Windows NT. Она была добавлена в MySQL 3.23.50.

  (До 3.23.50 поддержка именованных каналов активизировалась по умолчанию для серверов, которые поддерживают именованные каналы.)
- `net_buffer_length` (*глобальный, сеансовый*). Первоначальный размер буфера, используемого для обмена данными между сервером и клиентом. Этот буфер может расширяться до значения `max_allowed_packet` байт.
- `net_read_timeout` (*глобальный, сеансовый*). Время ожидания поступления данных по клиентскому соединению (в секундах) до наступления тайм-аута. Эта переменная была добавлена в MySQL 3.23.20.
- `net_retry_count` (*глобальный, сеансовый*). Количество повторных попыток прерванной операции чтения. Эта переменная была добавлена в MySQL 3.23.7.
- `net_write_timeout` (*глобальный, сеансовый*). Время ожидания (в секундах) наступления тайм-аута при блочной записи. Эта переменная была добавлена в MySQL 3.23.20.
- `open_files_limit`. При ненулевом значении эта переменная содержит количество дескрипторов файлов, которые сервер будет резервировать. Если его значение равно 0, сервер использует количество резервируемых дескрипторов как наибольшее значение из значений `max_connections*5` и `max_connections + table_cache*2`. Эта переменная была добавлена в MySQL 3.23.30.
- `pid_file`. Путь к файлу, куда сервер записывает идентификатор своего процесса.

  Эта переменная была добавлена в MySQL 3.22.23.
- `port`. Номер порта TCP/IP, который прослушивает сервер для соединения с клиентом.
- `protocol_version`. Номер версии протокола "клиент/сервер", используемого сервером. Эта переменная была добавлена в MySQL 3.22.18.
- `query_cache_limit` (*глобальный*). Максимальный размер кэшируемых результатов запроса; большие результаты не кэшируются.

  Эта переменная была добавлена в MySQL 4.0.1.

- query_cache_size (*глобальный*). Объем памяти, который используется для кэширования результата запроса. Для отключения режима кэширования эта переменная должна быть равна 0. Эта переменная была добавлена в MySQL 4.0.1.
- query_cache_type (*глобальный, сеансовый*). Режим работы кэша запросов.

Режим	Значение
OFF	Не кэшировать
ON	Кэшировать все запросы за исключением тех, которые начинаются с SELECT SQL_NO_CACHE
DEMAND	Кэшировать по требованию только те запросы, которые начинаются с SELECT SQL_CACHE

  Эта переменная была добавлена как query_cache_startup_type в MySQL 4.0.1 и переименована в query_cache_type в 4.0.3.
- read_buffer_size (*глобальный, сеансовый*). Размер буфера, используемого потоками, выполняющими последовательный просмотр таблиц. Этот буфер распределяется по мере необходимости между клиентами. До MySQL 4.0.3 эта переменная называлась record_buffer.
- read_rnd_buffer_size (*глобальный, сеансовый*). Размер буфера, который используется для упорядоченного считывания строк после сортировки. Данный буфер распределяется между клиентами по мере необходимости. Эта переменная была добавлена как record_rnd_buffer в MySQL 3.23.41 и переименована в read_rnd_buffer_size в 4.0.3.
- record_buffer. См. описание переменной read_buffer_size.
- record_rnd_buffer. См. описание переменной read_rnd_buffer_size.
- rpl_recovery_rank (*глобальный*). Ранг восстановления сервера при репликации. Сейчас эта переменная не используется; ее назначение в будущем заключается в предоставлении возможности подчиненному серверу выбрать новый главный сервер при потере старого. Эта переменная была добавлена в MySQL 4.0.0.
- safe_show_database. Разрешение безусловного отображения имен баз данных. При значении ON пользователи могут просматривать только те базы данных, на просмотр которых у них есть право. При значении OFF все имена баз данных отображаются всем пользователям. Эта переменная была добавлена в MySQL 3.23.30. Начиная с 4.0.3 пользователям, желающим просматривать все базы данных, необходимо предоставить привилегию SHOW DATABASES. Эта переменная была добавлена в MySQL 4.0.5.
- server_id (*глобальный*). Идентификатор репликации сервера. Эта переменная была добавлена в MySQL 3.23.26.
- skip_external_locking. Запрещено ли использование внешней блокировки (блокировка файловой системы). До MySQL 4.0.3 эта переменная называлась skip_locking.
- skip_locking. См. описание для переменной skip_external_locking.

- skip_networking. Значение OFF — для разрешения TCP/IP соединений, значение ON — для запрещения их. В последнем случае клиенты могут подключиться только с локального узла, используя соединения через разъемы в ОС UNIX или именованные каналы в ОС Windows. Эта переменная была добавлена в 3.22.23.
- skip_show_database. Значение ON — для отображения имен баз данных только пользователям, обладающим привилегией SHOW DATABASES (до MySQL 4.0.2 — PROCESS). Значение OFF — при отсутствии такого требования. Эта переменная была добавлена в MySQL 3.23.4.
- slave_net_timeout (*глобальный*). Время ожидания данных от главного сервера (в секундах) до наступления тайм-аута. Эта переменная была добавлена в MySQL 3.23.40.
- slow_launch_time (*глобальный*). Время (в секундах), определяющее поток как "медленный". Любой поток, для создания которого требуется больше времени, приводит к увеличению счетчика статуса Slow_launch_threads. Эта переменная была добавлена в MySQL 3.23.15.
- socket. Путь к доменному разъему ОС UNIX или имя именованного канала под ОС Windows.
- sort_buffer_size (*глобальный*). Размер буфера, который используется потоками для выполнения операций сортировки (GROUP BY или ORDER BY). Этот буфер распределяется между клиентами по мере необходимости. Если имеется сразу несколько клиентов, производящих сортировку одновременно, делать это значение слишком большим нецелесообразно (больше 1 Мбайт). До MySQL 4.0.3 эта переменная называлась sort_buffer.
- sql_mode. Значение параметра --sql-mode. Эта переменная была добавлена в MySQL 3.23.41; она замещает переменную ansi_mode.
- table_cache (*глобальный*). Максимальное количество открываемых таблиц. Этот кэш распределяется между потоками.
- table_type (*глобальный, сеансовый*). Тип таблиц по умолчанию, используемый для таблиц, которые созданы без параметра TYPE = *type_name* или с использованием неподдерживаемого значения *type_name*. Эта переменная была добавлена в MySQL 3.23.0.
- thread_cache_size (*глобальный*). Максимальное количество потоков, которые могут храниться в кэше. Эта переменная была добавлена в MySQL 3.23.16.
- thread_concurrency. Эта переменная применима только для работы СУБД MySQL под управлением ОС Solaris. Это значение передается функции thr_concurrency() для передачи информации диспетчеру потоков о том, сколько потоков работает одновременно. Эта переменная была добавлена в MySQL 3.23.7.
- thread_stack. Размер стека для каждого потока.
- timezone. Установка часового пояса сервера. Эта переменная была добавлена в MySQL 3.23.15.

- `tmpdir`. Путь к каталогу, в котором сервер создает временные файлы.

  Эта переменная была добавлена в MySQL 3.22.4. В версиях 3.22.0–3.22.3 она называлась `tmp_dir`.
- `tmp_table_size` (*глобальный, сеансовый*). Максимальный размер (в байтах) для временных таблиц. До MySQL 3.23, если размер таблицы превышал этот размер, программа `mysqld` возвращала клиенту сообщение об ошибке. Начиная с 3.23, если размер таблицы превышает этот размер, сервер преобразует тип этой таблицы в тип MyISAM. При наличии свободной памяти большие значения этой переменной позволяют серверу обрабатывать временные файлы большего размера.
- `transaction_isolation`. См. описание `tx_isolation`.
- `tx_isolation` (*глобальный, сеансовый*). Уровень изоляции транзакции по умолчанию. Эта переменная была добавлена в MySQL 3.23.36 и переименована в `tx_isolation` в версии 4.0.3.
- `version`. Версия сервера. Это значение состоит из номера версии с последующим одним или несколькими суффиксами. Значения суффиксов см. в описании функции `VERSION()` приложения В, "Операторы и функции".
- `wait_timeout` (*глобальный, сеансовый*). Время подключения, на протяжении которого неинтерактивный клиент может оставаться неактивным, пока сервер не решит, что это соединение можно закрыть. Для интерактивных клиентов вместо этого используется значение `interactive_timeout`.

SLAVE

```
SLAVE {START | STOP} [slave_options];
```

Этот оператор управляет работой подчиненного сервера репликации. Оператор `SLAVE START` запускает подчиненный поток, оператор `SLAVE STOP` останавливает его.

Оператор `SLAVE` был добавлен в MySQL 3.23.16. Начиная с MySQL 4.0.2 появилась возможность задавать необязательное предложение *slave_options*. Оно может состоять из одного или нескольких следующих параметров, разделенных запятыми.

- `IO_THREAD`. Запуск или остановка потока ввода-вывода, который получает запросы от главного сервера и сохраняет их в журнале переключений.
- `SQL_THREAD`. Запуск или остановка потока SQL, считывающего запросы из журнала переключений и выполняющего их.

TRUNCATE

```
TRUNCATE [TABLE] tbl_name;
```

Оператор `TRUNCATE TABLE` выполняет быстрое обновление содержимого таблицы с удалением и последующим восстановлением таблицы. Эта процедура работает значительно быстрее, чем построчное удаление таблиц. Данный оператор не является безопасным для транзакций, и при задании оператора `TRUNCATE TABLE` посередине активной транзакции или при наличии любых явных блокировок таблиц происходит ошибка.

Оператор TRUNCATE TABLE был добавлен в MySQL 3.23.28, но до версии 3.23.33 ключевое слово TABLE необходимо опускать.

UNION

```
select_statement
   UNION [ALL] select_statement
   [UNION select_statement] ...
```

Оператор UNION не является самостоятельным оператором. Он формирует объединение отдельных операторов SELECT, объединяя результаты работы этих операторов SELECT. Каждый оператор SELECT должен иметь в результирующем наборе одинаковое число столбцов. Имена и типы столбцов в результирующем наборе определяются именами и типами столбцов первого оператора SELECT. Если типы столбцов не соответствуют, будет произведено неявное преобразование типов в строках, полученных от второй таблицы и всех последующих.

По умолчанию оператор UNION не возвращает повторений. Его действие аналогично действию оператора DISTINCT, распространяемому на весь результирующий набор. Оператор UNION ALL сохраняет повторения, возвращая все строки.

Предложения ORDER BY или LIMIT могут применяться к отдельному оператору SELECT в пределах оператора UNION посредством размещения оператора внутри скобок. Предложения ORDER BY или LIMIT, расположенные в конце оператора UNION, а не в скобках, применимы ко всему результату работы оператора UNION. В этом случае любые столбцы, названные в предложении ORDER BY, относятся к именам столбцов, указанным в первом операторе SELECT.

Оператор UNION был добавлен в MySQL 4.0.0.

UNLOCK TABLE

```
UNLOCK {TABLES | TABLES}
```

Этот оператор снимает блокировки со всех таблиц, установленные текущим клиентом.

UPDATE

```
UPDATE [LOW_PRIORITY] tbl_name
   SET col_name=expr [, col_name=expr ] ...
   [WHERE where_expr] [ORDER BY ... ] [LIMIT n]
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name, tbl_name ...
   SET col_name=expr [, col_name=expr ] ...
   [WHERE where_expr] [ORDER BY ... ] [LIMIT n]
```

В первом варианте оператор UPDATE изменяет содержимое строк таблицы *tbl_name*. Изменению подлежат строки, выделенные с помощью выражения, определенного в предложении WHERE. В этих строках значения столбцов, перечисленных в предложении SET, устанавливаются равными заданному выражению *expr*.

```
UPDATE member SET expiration = NULL, phone = '197-602-4832'
WHERE member_id = 14;
```

Если предложение WHERE не определено, *изменяются все строки таблицы*.

По завершении выполнения оператор UPDATE возвращает число обновленных строк. Однако обновленной строка считается только в том случае, если значение

столбца действительно изменилось. Установка в столбце значения, аналогичного текущему, не приводит к изменению строки. Если пользователю необходимо знать, сколько строк соответствовало определенным в предложении WHERE условиям, независимо от обновления строки оператором UPDATE, при подключении к серверу установите флаг CLIENT_FOUND_ROWS. Детально об этом рассказывается в описании функции mysql_real_connect() в приложении E, "Программный интерфейс приложений C". При установке ключевого слова LOW_PRIORITY оператор не выполняется до тех пор, пока все клиенты не прекратят чтение таблицы. Это ключевое слово впервые появилось в MySQL версии 3.22.5.

При появлении в результате процедуры обновления дублирующего ключевого значения оператор UPDATE завершится ошибкой. Добавление предложения IGNORE приводит к тому, что такие записи не обновляются, но и ошибка при этом не происходит. Предложение IGNORE было добавлено в MySQL 3.23.16.

Предложение ORDER BY приводит к тому, что записи будут модифицированы в соответствии с порядком сортировки.

Это предложение было добавлено в MySQL 4.0.0 и имеет тот же синтаксис, что и SELECT.

Значение *n* предложения LIMIT определяет максимальное число строк, которое будет обновлено. Предложение LIMIT впервые появилось в MySQL версии 3.23.3.

Вторая форма синтаксиса оператора UPDATE подобна первой, но она позволяет производить обновление одновременно в нескольких таблицах. В этом случае предложение WHERE задает условия на основании объединения таблиц, предложение SET позволяет обновлять столбцы сразу в нескольких таблицах. Например, следующий оператор обновляет строки в таблице t1, имеющие значения столбца id, совпадающие со значениями одноименного столбца в таблице t2, копируя значения quantity из таблицы t2 в таблицу t1:

```
UPDATE t, t2 SET t.quantity = t2.quantity WHERE t.id = t2.id;
```

Возможность обновления сразу нескольких таблиц была добавлена в MySQL 4.0.2.

USE

```
USE db_name
```

Устанавливает базу данных *db_name* в качестве текущей (эта база данных по умолчанию выбирается для операторов, в которых имя обрабатываемой базы данных не указана явным образом). Если такая база данных не существует или пользователь не имеет достаточно полномочий на доступ к ней, оператор USE прерывает свою работу.

Переменные языка SQL

Начиная с MySQL 3.23.6 переменным SQL, заданным пользователями, могут присваиваться значения, которыми можно манипулировать позднее в других операторах.

Имена пользовательских переменных начинаются с '@' и могут состоять из алфавитно-цифровых символов их текущей кодировки и символов '_', '$' и '.'.

Имена таких переменных чувствительны к регистру.

Значения переменным могут присваиваться оператором = или := в операторах SET и оператором := в других операторах, таких как SELECT. Один оператор может включать в себя сразу несколько операторов.

```
mysql> SET @x = 0, @y = 2;
mysql> SET @color := 'red', @size := 'large';
mysql> SELECT @x, @y, @color, @size;
+------+------+--------+-------+
| @x   | @y   | @color | @size |
+------+------+--------+-------+
|    0 |    2 | red    | large |
+------+------+--------+-------+
mysql> SELECT @count := COUNT(*) FROM president;
+--------------------+
| @count := COUNT(*) |
+--------------------+
|                 42 |
+--------------------+
```

Переменным присваиваются цифровые, строковые значения или значения NULL. Кроме того, им могут присваиваться произвольные выражения, включая и такие, которые ссылаются на другие переменные.

Переменные имеют значение NULL до того момента, пока им не будет присвоено определенное значение. Значения переменных между сеансами не передаются. Это значит, что после завершения соединения значения будут утеряны.

В операторах SELECT, возвращающих больше одной строки, присвоение переменных производится для каждой строки. Окончательным является значение, которое присваивается последней строке.

В MySQL 4.1 и последующих версиях переменным присваивается кодировка в соответствии с присвоенным значением:

```
mysql> SET @s = _latin1_de 'abc'; SELECT CHARSET(@s);
+-------------+
| CHARSET(@s) |
+-------------+
| latin1_de   |
+-------------+
```

Синтаксис комментариев

В этом разделе рассказывается, как писать комментарии в коде SQL, а также описываются особенности интерпретации комментариев клиентской программой mysql. Комментарии довольно часто используются в файлах запросов, выполняемых в пакетном режиме, что следует учитывать при их написании.

Сервер MySQL различает три вида комментариев.

- Строка, начинающаяся с символа "#", рассматривается как комментарий. Этот же синтаксис применяется в большинстве оболочек и в языке Perl.
- Выражение, заключенное между символами "/*" и "*/", рассматривается как комментарий. Записанный в такой форме комментарий может занимать несколько строк и используется аналогично языку C.
- В MySQL 3.23.3 и более поздних версиях комментарии можно начинать с символов "-- " (два тире и пробел). При этом весь текст, расположенный

между этими символами и концом строки, рассматривается как комментарий. Подобный стиль комментирования применяется в некоторых других системах управления базами данных, но без пробела. В MySQL пробел необходим во избежание недоразумений в выражениях вида *value1-value2*, где *value2* — отрицательное значение (в этом случае второе значение может рассматриваться как комментарий).

При обработке запросов сервер пропускает комментарии. Исключение составляют лишь комментарии, которые начинаются символами "/*!" и трактуются особенным образом. В этом случае текст должен содержать ключевые слова SQL, которые и будут восприниматься сервером как часть оператора, в котором расположены комментарии. Так, например, сервер одинаково воспримет следующие строки:

```
INSERT LOW_PRIORITY INTO mytbl  SET ...;
INSERT /*! LOW_PRIORITY */ INTO mytbl SET ...;
```

Эта форма комментария предназначается специально для определения расширений и ключевых слов MySQL. MySQL распознает заданные ключевые слова, а другие серверы SQL игнорируют их. Это упрощает написание запросов, которые позволяют воспользоваться расширенными возможностями MySQL и в то же время работают с другими СУБД. Стиль комментирования с символами "/*!" впервые появился в MySQL 3.22.7.

В MySQL 3.22.26 и более поздних версиях после символов "/*!" можно указывать номер версии. Это укажет MySQL пропускать комментарии, если номер версии сервера меньше указанного. Представленные ниже комментарии к оператору UPDATE, например, будут проигнорированы, если только не используется сервер версии 3.23.3 или более поздней.

```
UPDATE mytable SET my_col = 100 WHERE mycol < 100 /*!32303 LIMIT 100 */;
```

Клиент MySQL обладает более ограниченными, чем сервер, возможностями распознавания комментариев. Программа mysql может неправильно интерпретировать представленную в форме C-комментариев конструкцию, поскольку использует менее совершенный, чем сервер, синтаксический анализатор. Так, например, появившийся внутри такого комментария символ кавычек заставит эту программу полагать, что строку необходимо проанализировать. Соответственно, программа продолжит просматривать строку до конца, пытаясь найти второй символ кавычек. Это видно на примере следующего оператора:

```
mysql> SELECT /* I have no quote */ 1;
+---+
| 1 |
+---+
| 1 |
+---+
mysql> SELECT /* I've got a quote */ 1;
    '>
```

Первый оператор проанализирован программой mysql правильно и послан на сервер для выполнения, после чего снова появляется подсказка "mysql>". Второй оператор содержит комментарии с одним символом кавычек. В результате программа mysql входит в режим анализа строки и продолжает находиться в нем по-

сле запуска строки, что видно по подсказке ">". Чтобы выйти из этого режима, введите второй символ кавычки с последующей командой "/c" для отмены запроса. Более детальное описание предложений программы mysql представлено в приложении Д, "Программы MySQL".

Еще одним символом, который может "сбить с толку" mysql при появлении в С-комментарии, является точка с запятой.

Д

Программы MySQL

В этом приложении...

Определение параметров программ
Библиотека libmysqld
Программы isamchk и myisamchk
Программы myisampack и pack_isam
Программа mysql
Сценарий mysqlaccess
Программа mysqladmin
Программа mysqlbinlog
Сценарий mysqlbug
Программа mysqlcheck
Программа mysql_config
Программа mysqld
Сценарий mysqld_multi
Сценарий mysqld_safe
Программа mysqldump
Сценарий mysqlhotcopy
Программа mysqlimport
Сценарий mysql_install_db
Сценарий mysql.server
Программа mysqlshow
Сценарий safe_mysqld

Это приложение посвящено описанию перечисленных ниже программ MySQL. Далее в приложении каждая из указанных программ рассматривается более детально. В частности, описывается ее назначение, синтаксис вызова, поддерживаемые параметры и используемые переменные. (Заметим, что программы упорядочены без учета символов '_' и '.', имеющихся в их именах.) Если не указано другое, параметры и переменные программ имелись как минимум в MySQL 3.22.0.

- `libmysqld`. Встроенный MySQL-сервер. Это не программа в полном смысле этого слова. Это библиотека, которая подключается к другим программам для создания автономных приложений, имеющих встроенный сервер.
- `isamchk` и `myisamchk`. Программы проверки и отладки таблиц, выполняющие также анализ распределения ключей, деактивизацию и активизацию индексов.
- `myisampack` и `pack_isam`. Программы создания сжатых, доступных только для чтения таблиц.
- `mysql`. Интерактивная программа, включающая средства редактирования строк и используемая для отправки запросов к серверу MySQL. При исполнении запросов, записанных в файле, эта программа может работать в пакетном режиме.
- `mysqlaccess`. Сценарий проверки привилегий доступа.
- `mysqladmin`. Программа, предназначенная для выполнения административных операций.
- `mysqlbinlog`. Программа, предназначенная для отображения журналов двоичных обновлений в формате ASCII.
- `mysqlbug`. Сценарий создания отчетов об ошибках.
- `mysqlcheck`. Программа, предназначенная для проверки таблиц, их восстановления, оптимизации и анализа.
- `mysql_config`. Программа, предназначенная для отображения соответствующих флагов компиляции программ, работающих с СУБД MySQL.
- `mysqld`. Сервер MySQL. Это основная программа, обеспечивающая клиентам доступ к базам данных.
- `mysqld_multi`. Сценарий запуска и остановки множественных серверов.
- `mysqld_safe`. Сценарий запуска и управления сервером MySQL. (До MySQL 4 этот сценарий носил название `safe_mysqld`.)
- `mysqldump`. Программа резервирования содержимого таблиц баз данных в текстовых файлах.
- `mysqlhotcopy`. Программа резервного копирования.
- `mysqlimport`. Программа загрузки данных в таблицы из текстовых файлов.
- `mysql_install_db`. Сценарий инициализации каталога данных сервера и таблиц разрешения.
- `mysql.server`. Сценарий запуска и завершения работы сервера MySQL.

- `mysqlshow`. Программа, обеспечивающая сбор информации о базах данных и таблицах.
- `safe_mysqld`. Сценарий запуска и мониторинга функционирования сервера MySQL. (После MySQL 4 этот сценарий носил название `mysqld_safe`.)

В представленном далее описании синтаксиса вся необязательная информация заключается в квадратные скобки (`[]`).

Определение параметров программ

С помощью параметров пользователь имеет возможность влиять на функционирование большинства программ MySQL. Эти параметры можно задавать в командной строке или конфигурационном файле. Кроме того, некоторые параметры определяются посредством установки значений переменных среды. Заданные в командной строке параметры обладают бо́льшим приоритетом по сравнению с параметрами, заданными другим способом. Аналогичным образом, установки, сделанные из конфигурационных файлов, имеют больший приоритет, чем значения переменных среды. Большинство программ MySQL понимает параметр `--help`, который обеспечивает быстрый способ получения справочной информации от самой программы. Например, если пользователь не имеет навыков работы с программой `mysqlimport`, для получения инструкций можно вызвать справку:

```
% mysqlimport --help
```

Параметр `-?` аналогичен параметру `help`, хотя в некоторых оболочках символ "?" интерпретируется как специальный символ в имени файла:

```
% mysqlimport -?
mysqlimport: No match.*
```

В этом случае можно попытаться запустить следующую команду:

```
% mysqlimport -\?
```

Некоторые параметры появляются в справочных сообщениях только при определенных условиях. Например, параметры `--debug` или параметры, относящиеся к протоколу SSL, появляются, только если MySQL компилировался с поддержкой отладки или поддержкой протокола SSL, а такие параметры, как `-pipe`, присущие только для ОС Windows, отображаются только в системах Windows.

Большинство параметров имеет как короткую (состоящую из одного символа), так и длинную (состоящую из полного слова) форму. Отличным примером являются два варианта параметра справки: `-?` и `--help`. Параметры, за которыми следует значение, должны задаваться в формате `--`*`name`*`=`*`val`*, где *name* — имя параметра, а *val* — его значение. В большинстве случаев, если за короткой формой следует значение, параметр и значение разделяются пробелом. Например, последовательности `-usampadm` и `-u sampadm`, определяющие имя пользователя, являются эквивалентными. Исключением из этого правила является лишь параметр `-p`, задающий пароль. Значение пароля здесь должно задаваться без промежуточных пробелов.

В описании каждой программы приводится список всех поддерживаемых параметров. Если какой-либо из параметров, описанных здесь, не поддерживается

программой, следовательно, используется устаревшая версия программы, которую необходимо обновить. (Однако сначала рекомендуется очень внимательно проверить синтаксис параметров.)

"Великое разделение" процедуры обработки параметров в MySQL 4.0.2

В MySQL 4.0.2 было сделано существенное нововведение, была также пересмотрена обработка параметров с целью создания более унифицированного формата, предназначенного для обработки логических параметров, имеющих только два значения – on/off. Такие параметры теперь имеют основную форму и несколько распознаваемых связанных форм, приведенных в следующей таблице.

Параметр	Описание
`--name`	Основная форма параметра; разрешение параметра
`--enable-name`	Префикс `-enable-`; параметр разрешен
`--disable-name`	Префикс `-disable-`; параметр не разрешен
`--skip-name`	Префикс `--skip-`; параметр не разрешен
`--name=1`	Суффикс `=1`; параметр разрешен
`--name=0`	Суффикс `=0`; параметр не разрешен

Например, многие команды MySQL для включения сжатия протокола "клиент/сервер" используют параметр `--compress`. До MySQL 4.0.2 этот параметр указывался для включения режима сжатия, а при отсутствии сжатия он не указывался.

Такая форма работы с параметром по-прежнему возможна, но начиная с MySQL 4.0.2 для включения и отключения режима сжатия также правомочны такие формы, как `--enable-compress` и `--compress=1`, для включения режима сжатия и `--disable-compress`, `--skip-compress` и `--compress=0` для отключения режима сжатия.

Все последующие описания программ, которые можно будет встретить в этом приложении, используют атрибут "(*boolean*)" для обозначения параметров, которые являются предметом такой интерпретации, т.е. параметры, для которых префиксы и суффиксы, приведенные в таблице, поддерживаются. Присутствие этого аргумента показывает, что сейчас предпочтительнее пользоваться префиксами `--enable-`, `--disable-` и `--skip-`. Однако при работе с программами до версии 4.0.2 можно использовать `=0`, или `=1`, или даже другой суффикс. Например, более старые версии `mysql` поддерживают использование параметра `--no-named-columns` для подавления имен столбцов в результатах запросов.

Начиная с MySQL 4.0.2 основной параметр имеет вид `--named-columns` (что также является установкой по умолчанию), а подавление или использование заголовков задается с помощью `--disable-named-columns` или `--skip-named-columns`.

Принимая во внимание, что большая часть инсталляций MySQL, используемых в данный момент, является инсталляциями MySQL до версии 4.0.2, сейчас может быть не совсем ясно, какой вариант можно использовать при запуске определенной программы MySQL. (Большинство параметров остались теми же и не зависят

от используемой версии MySQL, но синтаксис логических параметров в некоторых случаях поменялся.) Но по мере смещения версии инсталляции в сторону MySQL 4 эта проблема будет возрастать. При возникновении малейших сомнений можно вызвать параметр `--help` и посмотреть, какая форма параметра поддерживается в данном конкретном случае.

Другие нововведения в обработке параметров, появившиеся в версии 4.0.2, заключаются в следующем.

- Имена параметров были сокращены до неоднозначных префиксов, что упрощает определение параметров, имеющих очень длинные имена.
- Установку переменных программы можно производить из командной строки или из конфигурационного файла, рассматривая имена переменных как имена параметров. Таким образом, больше нет необходимости в использовании параметров `--set-variable` или `-O`. Подробнее об этом можно узнать в разделе "Установка программных переменных".
- Программа `mysqld` поддерживает префикс `--maximum-` для определения максимального значения установки переменных, задаваемых пользователем. Например, сервер позволяет пользователям устанавливать размер буфера сортировки, изменяя переменную `sort_buffer_size`. По мере необходимости можно задать значению этой переменной ограничение, равное 64 Мбайт, и запустить сервер с параметром `--maximum-sort_buffer_size=64M`.
- Префикс `--loose-` поддерживается для упрощения использования разных версий программы, которые могут не распознавать определенные параметры. Например, сервер версии 4.1 и выше может распознавать параметр `--old-passwords`, но более старые версии не могут. Задав этот параметр в виде `--loose-old-passwords`, любая версия сервера начиная с 4.0.2 использует или проигнорирует параметр, в зависимости от того, понимает она или нет `--old-passwords`.

Стандартные параметры программ MySQL

Некоторые параметры имеют одинаковое значение во всех программах MySQL. Именно поэтому было принято решение описать все их в начале, а затем дать краткие рекомендации в разделах, посвященных стандартным параметрам. (В этом разделе описаны только длинные названия параметров, хотя программы отлично понимают и их короткие формы, если не указано другое.)

- `--character-sets-dir=`*`dir_name`*. Каталог, в котором хранятся файлы кодировки.
- `--compress`, `-C` (*`boolean`*). Активизирует средства сжатия протокола, используемого для взаимодействия клиента и сервера (если, конечно, протокол имеет такие средства). Этот параметр используется только клиентскими программами. Он впервые появился в MySQL 3.22.3.
- `--debug=`*`debug_options`*, `-#` *`debug_options`*. Активизирует вывод средств отладки. Этот параметр приведет к желаемому эффекту только в том случае, если в MySQL была предусмотрена и активизирована поддержка средств отладки. Разделенные запятой параметры задаются в строке

debug_options. Чаще всего эта строка содержит значение `d:t:o,`*file_name*, которое активизирует отладку, включает элемент трассировки входа и выхода из функций, а также отправляет выводимые данные в файл *file_name*.

Пользователям, которые собираются серьезно заняться отладкой своих программ, рекомендуется прочитать файл dbug/dbug.c исходного дистрибутива MySQL. Этот файл содержит описание всех необходимых для отладки параметров.

- `--default-character-set=`*charset*. Имя стандартной кодировки.
- `--help, -?`. Указывает программе вывести справочную информацию и завершить работу.
- `--host=`*host_name*, `-h` *host_name*. Определяет главный компьютер, с которым устанавливается соединение (и на котором запущен сервер). Этот параметр используется только клиентскими программами.
- `--password[=`*pas_val*`], -p[`*pas_val*`]`. Определяет пароль, используемый при подключении к серверу. Если после ключевого слова этого параметра не определен пароль *pas_val*, программа попросит пользователя ввести его. Задаваемый в командной строке пароль должен размещаться сразу за именем параметра без промежуточных пробелов. (Это означает, что короткая форма этого параметра имеет вид `-p`*pass_val*, а не `-p` *pass_val*.) Этот параметр используется только клиентскими программами.
- `--pipe, -W`. Используется для подключения к серверу через именованный канал. Этот параметр используется только клиентскими программами, работающими под управлением ОС Windows.
- `--port=`*port_num*, `-P` *port_num*. Клиентским программам этот параметр указывает номер порта, который следует использовать при подключении к серверу. Этот порт применяется при TCP/IP-соединениях (соединениях, в которых узлом не является `localhost` в ОС UNIX или '.' в Windows). Серверу `mysqld` этот параметр сообщает номер порта, по которому следует ожидать запросы на установление соединения TCP/IP.
- `--set-variable` *var=value*, `-O` *var=value*. В некоторых программах пользователь имеет возможность сам устанавливать значения переменных (операционных параметров). Для этих целей используется параметр `--set-variable`. Достаточно задать имя переменной *var* и ее новое значение *value*. Этот параметр потерял значение начиная с MySQL 4.0.2, так как появилась возможность устанавливать переменные с использованием их имен. Старый и новый синтаксис описаны ниже, в разделе "Установка программных переменных".
- `--silent, -s`. Определяет "тихий" режим работы. Это не означает, что программа вовсе не обменивается данными, просто она генерирует меньше выводимых данных, чем обычно. Некоторые программы позволяют задавать этот параметр несколько раз, добиваясь последовательного снижения объема выводимых данных (подобные возможности имеются и в конфигурационных файлах).

- `--socket=path_name`, `-S path_name`. Для клиентских программ этот параметр определяет полный путь и имя файла разъема, используемого для подключения к серверу. Файл разъема применяется для установления соединения с компьютером `localhost`. Для клиентских программ под Windows это имя именованного канала, если в качестве имени узла используется '.'.
- `--user=user_name`, `-u user_name`. Клиентским программам этот параметр сообщает имя пользователя при подключении к серверу. Если этот параметр не указан, в качестве имени пользователя по умолчанию используется регистрационное имя в UNIX и ODBC в Windows. Серверу `mysqld` этот параметр указывает имя учетной записи UNIX, под управлением которой следует запустить сервер. (Для достижения наибольшей эффективности можно запустить сервер с помощью этого параметра с полномочиями пользователя `root`.)
- `--verbose`, `-v`. Определяет "шумный" режим, при котором программа генерирует больше выводимых данных, чем обычно. Некоторые программы позволяют задавать этот параметр несколько раз, добиваясь последовательного повышения объема вывода (подобные возможности предусмотрены и в конфигурационных файлах).
- `--version`, `-V`. Указывает программе вывести строку с информацией о текущей версии и завершить работу.

Стандартные параметры SSL

Следующие параметры используются для установки безопасных соединений. Они появились в MySQL 4.0.0, но только для сервера MySQL, скомпилированного с поддержкой протокола SSL. Подробно о безопасном соединении см. в главе 12, "Безопасность".

- `--ssl`. Разрешить SSL-соединения (или запретить их, если указан параметр `--disable-ssl`). Параметр `--ssl` можно заменить любым другим параметром SSL.
- `--ssl-ca=file_name`. Путь к файлу сертифицирующей организации.
- `--ssl-capath=dir_name`. Путь к каталогу доверительных сертификатов, которые будут использоваться для проверки сертификата.
- `--ssl-cert=file_name`. Путь к файлу сертификата.
- `--ssl-cipher=str`. Строка распечатки шифров SSL, которые используются для шифрования трафика через данное соединение. Это значение может быть представлено одним или несколькими типами шифров, разделенных запятыми.
- `--ssl-key=file_name`. Путь к файлу ключей.

Установка программных переменных

Некоторые программы MySQL имеют устанавливаемые переменные (рабочие параметры). В MySQL 4.0.2 появилась возможность задавать значения переменных, рассматривая их имена в качестве параметров. Например, вызвать `mysql` с переменной `connect_timeout`, равной 10, можно следующей командой:

```
% mysql --connect_timeout=10
```

Этот синтаксис также позволяет вместо символов подчеркивания в именах переменных использовать тире. Это сделает установку переменных похожей на задание других параметров:

```
% mysql --connect-timeout=10
```

До MySQL 4.0.2 для установки переменных использовался параметр `--set-variable` (или его короткий эквивалент `-O`). Таким образом, устанавливать переменную `connect_timeout` можно следующим образом:

```
% mysql --set-variable=connect_timeout=10
% mysql -O connect_timeout=10
```

Параметры `--set-variable` и `-O` по-прежнему используются в MySQL 4.0.2 и выше.

Значения переменных, представляющих размеры буферов или длины, задаются в байтах. Поэтому подобные значения можно задавать как отдельным числом, так и с указанием суффиксов "K" и "M", обозначающих соответственно Кбайт и Мбайт. Регистр символов в таких суффиксах не играет роли, поэтому суффиксы "k" и "m" эквивалентны суффиксам "K" и "M". Начиная с MySQL 4.0.2 также можно использовать 'G' или 'g' для обозначения гигабайтов.

Список переменных для каждой программы можно найти ниже в этом приложении, а также получить с помощью параметра `--help`.

Конфигурационные файлы

Конфигурационные файлы впервые появились в MySQL 3.22. Они применяются для записи и хранения параметров программ MySQL, что исключает необходимость ввода этих параметров в командной строке при каждом вызове программы. Пример конфигурационного файла `my-example.cnf` можно найти в подкаталоге `/share/mysql` каталога инсталляции MySQL или в каталоге `support-files` исходного дистрибутива.

Все определенные в конфигурационном файле параметры могут перекрываться параметрами, заданными в командной строке.

MySQL просматривает в поисках параметров несколько файлов. Отсутствие конфигурационного файла не приводит к возникновению ошибки в работе программы. В операционных системах UNIX программа просматривает следующие файлы (в указанном порядке).

Имя файла	Содержимое
`/etc/my.cnf`	Глобальные параметры
DATADIR`/my.cnf`	Параметры сервера
`~/.my.cnf`	Параметры, заданные пользователями

В операционных системах Windows считываются следующие файлы.

Имя файла	Содержимое
SYSTEMDIR`\my.ini`	Глобальные параметры
`C:\my.cnf`	Параметры сервера
DATADIR`\my.cnf`	Параметры, заданные пользователями

Часть *DATADIR* представляет путь к каталогу данных на рабочем компьютере. (Этот путь используется также сервером и не меняется с параметром `--datadir`.) На компьютерах, работающих под управлением Windows, часть *DATADIR* заменяется каталогом `C:\mysql\data`. Часть *SYSTEMDIR* указывает путь к системному каталогу Windows (обычно это `C:\Windows` или `C:\WinNT`).

Глобальные конфигурационные файлы принимаются программами, параметры которым можно задавать через файлы. Расположенный в каталоге данных сервера конфигурационный файл предназначен только для отдельной программы-сервера. Аналогичным образом, пользовательские файлы применяются программами, запускаемыми отдельными пользователями.

- Несмотря на то что для разделения компонентов пути в Windows используется символ "\", внутри конфигурационных файлов для подобных целей применяется символ "/". Если все же в качестве разделителя необходимо использовать "\", введите подряд два символа "\\", поскольку один "\" рассматривается как управляющий символ.
- Часто ОС Windows скрывает расширения имен файлов. Конфигурационный файл, созданный с именем `my.cnf`, Windows может отображать как `my`. При этом попытка исправить имя в Проводнике Windows может привести к тому, что конфигурационный файл больше работать не будет. А это происходит потому, что в действительности файл был переименован из `my.cnf` в `my.cnf.cnf`!
- Конфигурационные файлы — это простые текстовые файлы. Конфигурационный файл можно создавать с помощью текстового редактора, но при этом необходимо обязательно удостовериться, что он будет сохранен именно в текстовом формате, а не в каком-то специфическом формате, присущем данному редактору.

Четыре параметра, имеющих отношение к обработке конфигурационных файлов, являются стандартными для большинства программ MySQL и имеют значение, при задании которого этот параметр должен указываться первым в командной строке.

- `--defaults-extra-file=`*`file_name`*. Считывать дополнительные параметры из этого файла. Он должен считываться после глобальных конфигурационных файлов и конфигурационных файлов сервера, но перед пользовательскими файлами. Этот параметр был добавлен в MySQL 3.23.26, за исключением `safe_mysqld` (версия 3.23.27) и `mysql_install_db` (версия 3.23.29).
- `--defaults-file=`*`file_name`*. Определяет путь и имя конфигурационного файла. Обычно программы ищут конфигурационный файл в нескольких заранее предопределенных местах, однако если задан параметр `--defaults-file`, считывается только указанный в нем файл. Этот параметр необходимо размещать в командной строке первым. Параметр `--defaults-file` впервые появился в MySQL 3.22.23.
- `--no-defaults`. Запрет на использование любых конфигурационных файлов. Если этот параметр задается, его необходимо размещать в командной строке первым.

- `--print-defaults`. Выводит значения параметров, которые используются при вызове программы, если в ее командной строке вовсе отсутствуют параметры. В выводимый список входят значения, которые извлекаются из конфигурационных файлов (и значений переменных среды). Параметр `--print-defaults` отлично подходит для проверки правильности настройки конфигурационного файла. Он также оказывается весьма кстати, если MySQL использует параметры, которые сам пользователь никогда прежде не использовал. Кроме того, с помощью параметра `--print-defaults` можно проверить, из какого файла извлекаются параметры. Если этот параметр задан, его следует разместить в командной строке первым.

При запуске программы с параметром `--help` одна часть сообщения об ошибке покажет конфигурационные файлы, которые программа обычно пытается прочесть. (Набор считываемых файлов зависит от параметров `--defaults-file`, `--defaults-extra-file` или `--no-defaults`.)

Внутри конфигурационных файлов параметры распределены по группам. Например:

```
[client]
user=sampadm
password=secret

[mysql]
no-auto-rehash

[mysqlshow]
status
```

Названия групп заключаются в квадратные скобки и обычно соответствуют именам программ. В предыдущем примере группа `[mysql]` содержит параметры для клиента `mysql`, а `[mysqlshow]` — для программы `mysqlshow`. Заданные в специальной группе `[client]` параметры используются всеми клиентскими программами. Вообще, стандартные клиентские программы MySQL ищут как группу `[client]`, так и группу с собственным названием. Например, программа `mysql` просматривает параметры групп `[client]` и `[mysql]`, а программа `mysqlshow` — `[client]` и `[mysqlshow]` соответственно.

Все следующие за названием группы параметров относятся к данной группе. Конфигурационный файл может содержать любое число групп, причем параметры групп, указанных позднее, обладают бо́льшим приоритетом, чем параметры групп, приведенных первыми. Это означает, что если один параметр определяется в просматриваемых программой группах несколько раз, при запуске применяется значение, указанное последним.

Параметры определяются отдельной строкой. Первое слово в такой строке является именем параметра, которое может быть представлено в длинном формате без начальных дефисов. (Другими словами, для определения сжатия в командной строке можно использовать последовательности `-C` или `--compress`. В конфигурационном файле достаточно указать слово `compress`.) В группах конфигурационного файла можно задавать любые поддерживаемые программой параметры. Если необходимо задать значение для какого-либо параметра, отделите его от названия параметра символом равенства "=".

Рассмотрим для примера следующую командную строку:

```
% mysql --compress --user=sampadm -set-variable max_allowed_packet=16M
```

Чтобы представить ту же информацию в группе [mysql] конфигурационного файла, достаточно ввести следующие строки:

```
[mysql]
compress
user=sampadm
set-variable=max_allowed_packet=16M
```

Обратите внимание, что в конфигурационном файле помимо основного знака равенства между названием параметра и его значением знак равенства установлен также после предложения set-variable.

Начиная с MySQL 4.0.2, где разрешается рассматривать имена переменных непосредственно как параметры, можно использовать

```
% mysql --compress --user=sampadm --max_allowed_packet=16M
```

или

```
[mysql]
compress
user=sampadm
max_allowed_packet=16M
```

Пустые строки конфигурационного файла, а также строки, начинающиеся с символов "#" или ";", рассматриваются как комментарии и пропускаются программой. Ведущие пробелы в строках с параметрическими группами игнорируются.

Для представления специальных символов в значениях параметров можно применять следующие последовательности выхода.

Последовательность	Значение
\b	<Backspace>
\n	Переход на новую строку
\r	Возврат каретки
\s	Пробел
\t	Табуляция
\\	Обратная косая черта

При работе с конфигурационным файлом следует проявлять максимальную осторожность, чтобы случайно не задать в группе [client] параметры, понятные только одной клиентской программе. Если, например, поместить в эту группу параметр skip-line-numbers, понятный только программе mysql, другие клиентские программы типа mysqlimport перестанут работать. (Попытка их запуска приведет к появлению сообщения об ошибке с последующей справочной информацией.) В такой ситуации необходимо перенести параметр skip-line-numbers в группу [mysql].

В группе [client] нужно определять параметры, понятные всем программам MySQL, а именно:

```
character-sets-dir=charset_directory_path
compress
connect-timeout=seconds
database=db_name
debug
```

```
default-character-set=charset_name
disable-local-infile
host=host_name
init-command=query
interactive-timeout=seconds
local-infile
password=your_pass
pipe
port=port_num
return-found-rows
socket=socket_name
ssl-ca=ssl_certificate_authority file
ssl-capath=ssl_certificate_authority_directory_path
ssl-cert=ssl_certificate_file
ssl-key=ssl_key_file
timeout=seconds
user=user_name
```

Поддержка конфигурационных файлов была добавлена в MySQL 3.22.10, но некоторые из имеющихся значений параметров были добавлены позднее.

- `debug`. Добавлен в MySQL 3.22.11.
- `return-found-rows`. Добавлен в MySQL 3.22.15.
- `character-sets-dir` и `default-character-set`. Добавлены в MySQL 3.23.14.
- `interactive-timeout`. Добавлен в MySQL 3.23.28.
- `connect-timeout`. Добавлен в MySQL 3.23.49. Является синонимом `timeout` и сейчас не используется.
- `disable-local-infile` и `local-infile`. Добавлены в MySQL 3.23.49.
- `ssl-ca`, `ssl-capath`, `ssl-cert` и `ssl-key`. Добавлены в MySQL 4.0.0.

Использование программы `my_print_defaults` для проверки параметров

Программа `my_print_defaults` применяется для определения параметров, которые данная программа может найти в конфигурационном файле. Она считывает конфигурационный файл и отображает, какие параметры были там обнаружены для одной или нескольких параметрических групп. Например, программа `mysql` использует параметры из групп `[client]` и `[mysql]`. Для того чтобы определить, какие параметры из вашего конфигурационного файла имеют отношение к `mysql`, необходимо запустить программу `my_print_defaults` следующим образом:

```
% my_print_defaults client mysql
```

Аналогично, сервер `mysqld` использует параметры, заданные в группах `[mysqld]` и `[server]`. Для того чтобы определить, какие параметры имеются в конфигурационных файлах, можно задать следующую команду:

```
% my_print_defaults mysqld server
```

Реализация программы `my_print_defaults` для UNIX появилась в MySQL 3.23.19, а для Windows — в 4.0.4.

Пользовательские конфигурационные файлы следует хранить закрытыми

Администратор должен обязательно убедиться, что файл с пользовательскими параметрами принадлежит "правильному" пользователю и что для него установлен режим 600 или 400, закрывающий доступ к файлу другим пользователям. Мало кто из пользователей захочет, чтобы его имя и пароль стали доступны другим лицам.

```
% chmod 600 .my.cnf
% chmod go-rwx .my.cnf
```

Переменные среды

Для установки значений параметров программы MySQL используют значения переменных среды. Эти переменные обладают самым низким приоритетом. Другими словами, параметры, определенные с помощью переменных среды, могут перекрываться параметрами, установленными в конфигурационном файле или в командной строке.

В частности, программы MySQL проверяют значения следующих переменных.

- `MYSQL_DEBUG`. Параметры, используемые для отладки. Эта переменная может эффективно использоваться только в том случае, когда MySQL имеет встроенную поддержку средств отладки. Установка переменной `MYSQL_DEBUG` аналогична определению параметра `--debug`.
- `MYSQL_PWD`. Пароль, используемый для подключения к серверу MySQL. Установка переменной `MYSQL_PWD` равносильна определению параметра `--password`.

  Использование переменной `MYSQL_PWD` для хранения пароля представляет определенный риск с точки зрения безопасности, так как другие пользователи системы могут очень просто получить ее значение. Например, с помощью утилиты `ps` можно просмотреть установки переменных среды для других пользователей.
- `MYSQL_TCP_PORT`. Для клиентских программ эта переменная определяет номер порта, используемого для установления TCP/IP-соединения с сервером. Серверу `mysqld` эта переменная указывает порт, от которого следует ожидать запрос на установление TCP/IP-соединения. Установка переменной `MYSQL_TCP_PORT` равносильна определению параметра `--port`.
- `MYSQL_UNIX_PORT`. Для клиентских программ эта переменная определяет файл разъема, используемый для установления соединения через разъем домена UNIX с сервером, работающим на компьютере `localhost`. Серверу `mysqld` эта переменная указывает разъем, по которому следует ожидать локального подключения. Установка переменной `MYSQL_UNIX_PORT` аналогична определению параметра `--socket`.
- `TMPDIR`. Путь и имя каталога, в котором создаются временные файлы. Установка этой переменной аналогична использованию параметра `--tmpdir`.
- `USER`. Эта переменная определяет имя пользователя MySQL, используемое при подключении к серверу. Она используется только клиентскими программами, работающим под управлением Windows, и равносильна параметру `--user`.

Клиентская программа `mysql` проверяет также значение следующих переменных.

- `MYSQL_HISTFILE`. Эта переменная определяет имя файла, используемого для хранения журнала командных строк. По умолчанию для нее устанавливается значение `$HOME/.mysql_history`, где `$HOME` — месторасположение корневого каталога.
- `MYSQL_HOST`. Эта переменная определяет имя компьютера, с которым необходимо установить соединение для подключения к серверу. Установка этой переменной равносильна использованию параметра `--host`.

- MYSQL_PS1. Строка, которая будет использоваться вместо подсказки "mysql>". Эта строка может содержать специальные последовательности, необходимые для входа в mysql.

Библиотека libmysqld

Библиотека libmysqld является встроенным сервером MySQL. Это библиотека, которая подключается к главной программе для создания автономного приложения, включающего сервер MySQL. Библиотека libmysqld не включает поддержку некоторых возможностей, которые являются частью mysqld. Например, для экономии пространства отключается поддержка ISAM, отсутствует поддержка протокола SSL, так как связь с главной программой не подразумевает использования сетевого соединения с сервером и к нему невозможно подключение с помощью других программ. Последнее ограничение не позволяет использовать репликации со встроенным сервером.

Однако, из-за того, что встроенный сервер не нуждается в сетевых возможностях, и из-за того, что от него не требуется создание каких-либо выводных файлов, одним из применений libmysqld является создание приложений, которые используются на носителях, предназначенных только для чтения, таких как CD-ROM, и на машинах, которые используются в автономном режиме работы (например, в справочных системах).

Библиотека libmysqld сама не производит никакого чтения из командной строки, но есть возможность передачи параметров и интерпретации их, как это делается и в любой другой программе MySQL. В основном, libmysqld принимает те же параметры, что и mysqld. Параметры, которые имеют отношение к функциям сервера, не поддерживаемых libmysqld (например, параметры репликации), игнорируются.

Информацию о встроенном сервере можно найти в приложении В, "Операторы и функции", в котором описано, как создаются главные программы, работающие со встроенным сервером, обсуждаются проблемы администрирования, являющиеся общими для серверов всех типов; а также в приложении Д, "Программы MySQL", в котором описаны специфические функции API-интерфейса языка программирования C, имеющие отношение к встроенному серверу.

Программы isamchk и myisamchk

Эти программы применяются для проверки и восстановления поврежденных таблиц, вывода информации о таблицах, проведения анализа распределения значений индексных ключей, а также их включения/отключения. Подробнее об анализе ключей и работе с индексами можно узнать в главе 4, "Оптимизация запросов". В главе 13, "Поддержка и восстановление баз данных", подробно рассказывается о процедурах проверки и восстановления таблиц.

Программа myisamchk используется для работы с таблицами, представленными в формате MyISAM. Имена файлов данных и индексов таких таблиц имеют расширения MYD и MYI соответственно. Для обработки таблиц, записанных в более старом формате хранения ISAM, используется программа isamchk. Имена

файлов данных и индексов таких таблиц имеют расширения ISD и ISM. Если запустить любую из этих программ для таблицы неверного типа, программа выдаст сообщение об ошибке и таблицу обрабатывать не будет.

Важно также помнить, что нельзя выполнять проверку или восстановление таблицы, пока с ней работает сервер MySQL. В главе 13, "Поддержка и восстановление баз данных", рассказывается, как заблокировать использование таблицы на время работы программ myisamchk и isamchk.

Использование

```
myisamchk [options] tbl_name[.MYI]
isamchk [options] tbl_name[.ISM]
```

Если при вызове программ не заданы параметры, эти программы просто проверяют таблицы на наличие ошибок. Во всех остальных случаях таблицы обрабатываются в соответствии со значением заданных параметров. При выполнении операции, обновляющей содержимое таблицы, лучше всего предварительно ее скопировать.

В качестве аргумента *tbl_name* можно задать как имя таблицы, так и имя ее индексного файла (с расширением ISM или MYI). Использование названий индексных файлов предпочтительней, поскольку позволяет с помощью специальных символов задать проверку (или отладку) всех таблиц одного формата одновременно. Так, например, для проверки всех таблиц в текущем каталоге достаточно ввести следующие команды:

```
% myisamchk *.MYI
% isamchk *.ISM
```

Эти программы самостоятельно не ищут заданные таблицы, поэтому если файлы таблиц не располагаются в текущем каталоге, их расположение следует задать явным образом. Так как для проверки программами isamchk и myisamchk файлы таблиц необязательно должны размещаться в каталоге данных сервера, их можно легко скопировать в другой каталог и выполнить проверку копий.

Некоторые параметры этих программ связаны с номерами индексов. Нумерация индексов начинается с 1. Для определения нумерации индексов определенной таблицы можно запустить оператор SHOW INDEX или воспользоваться командой mysqlshow --keys. В столбце Key_name индексы указаны в таком же порядке, в котором они используются программами myisamchk и isamchk.

Стандартные параметры, поддерживаемые программами isamchk и myisanchk

```
--character-sets-dir --help                    --verbose
--debug                        --set-variable   --version
--default-character-set  --silent
```

Параметр --default-character-set имеется только для isamchk. При задании параметра --silent на экран выводятся только сообщения об ошибках. Параметр --verbose, наоборот, выводит на экран, помимо основной, еще и дополнительную информацию, особенно если задается вместе с параметрами

--description и --extend-check (для обеих программ) или с параметрами --check (для программы myisamchk). Для усиления эффекта параметры --silent и --verbose можно дублировать и задавать несколько раз.

Общие параметры программ isamchk и myisamchk

Программы isamchk и myisamchk имеют множество общих параметров.

- --analyse, -a. Выполняет анализ распределения ключей. Это позволяет серверу быстрей выполнять поиск с помощью индексов и процедуру объединения таблиц. Чтобы после выполнения анализа получить информацию о распределении ключей, достаточно еще раз запустить эти программы с параметрами --description и --verbose.
- --block-search=*n*, -b *n*. Выводит начало строки таблицы, которая содержит начало блока *n*. Этот параметр используется только в целях отладки.
- --description, -d. Выводит описательную информацию о таблице.
- --extend-check, -e. Выполняет расширенную проверку таблицы. Этот параметр, как правило, используется редко, поскольку программы myisamchk и isamchk находят все ошибки и при работе в обычном режиме проверки.
- --force, -f. Запускает проверку или восстановление таблицы, даже если временный файл для нее уже существует. Как правило, если программы myisamchk и isamchk находят файл *tbl_name*.TMD, то выводят сообщение об ошибке и завершают свою работу. Это объясняется тем, что существование такого файла может указывать на проверку данной таблицы другой программой. Хотя временный файл будет также существовать и в том случае, если программа проверки внезапно завершила свою работу и не успела корректно удалить файл. Если администратор уверен, что наличие файла объясняется именно последней причиной, то можно воспользоваться параметром --force. (Либо же можно самостоятельно удалить временный файл.)

  Если параметр --force используется для проверки таблиц, программа автоматически перезапускается с параметром --recover для каждой таблицы, в которой обнаружены ошибки.
- --information, -i. Выводит статистическую информацию о содержимом таблицы.
- --keys-used=n, -k n. Этот параметр используется вместе с параметром --recover и указывает MySQL обновить только первые n индексов. Другими словами, он отключает все индексы с номером больше n. Параметр --key-used можно использовать для повышения эффективности работы операторов INSERT, DELETE и UPDATE. При определении номера индекса 0 отключаются все индексы. Аналогичным образом, определение номера индекса, равного наибольшему номеру индекса таблицы, приводит к включению всех индексов.
- --no-symlinks, -l. Если в качестве аргумента таблицы выступает символическая ссылка, обработке, как правило, подлежит таблица, на которую эта ссылка указывает. Если определен этот параметр, символические ссылки не используются, а *заменяются* новыми (исправленными) версиями файлов таблиц.

Начиная с MySQL 4 для программы myisamchk этот параметр отсутствует, так как она не удаляет символические ссылки при операциях восстановления.

- `--quick, -q (boolean)`. Этот параметр используется вместе с параметром `--recover` для ускорения восстановления. Если заданы сразу оба параметра, файл данных не изменяется. Чтобы заставить программу изменять файл данных даже при нахождении дублирующих значений ключей, определите параметр `--quick` дважды. В этом случае не забудьте предварительно создать резервную копию файла данных.
- `--recover, -r`. Выполняет обычную процедуру восстановления. Этот параметр позволяет устранить практически все ошибки за исключением дублирования значений в уникальных индексах.
- `--safe_recover, -o`. Использует более медленный режим восстановления, чем с параметром `--recover`. Это позволяет устранить некоторые проблемы, которые обычный режим преодолеть не в состоянии. Параметр `--safe-recover` также требует меньше пространства, чем `--recover`.
- `--sort-index, -S`. Сортирует блоки индексов для ускорения последовательного считывания блоков.
- `--sort-records=n, -R n`. Сортирует записи данных в соответствии с порядком их расположения в индексе n. В результате ускоряется процедура поиска, основанная на этом индексе. Если данная операция запускается для таблицы впервые, она займет много времени, поскольку записи еще не упорядочены. Для MySQL 3.23.28 или выше оператор `ALTER TABLE ... ORDER BY` выполняет ту же операцию, что и `--sort-records`, и обычно работает быстрее.
- `--unpack, -u`. Распаковывает заархивированный файл. Программа myisamchk предназначена для распаковки MyISAM-файлов, запакованных программой myisampack. Программа isamchk предназначена для распаковки ISAM-файлов, запакованных программой pack_isam. Этот параметр можно также использовать для преобразования сжатых и доступных только для чтения таблиц в изменяемый формат. Его нельзя использовать с параметрами `--quick` или `--sort-records`.
- `--wait, -W`. Если указанная таблица заблокирована, запущенные с этим параметром программы не приступают к своей работе до тех пор, пока блокировка не будет снята. Если же параметр `--wait` не определен, программа ожидает 10 секунд и, если блокировка не снимается, выдает сообщение об ошибке.

Специальные параметры программы myisamchk

Если программа isamchk не содержит каких-либо специальных параметров, то у программы myisamchk они имеются.

- `--backup, -B`. Для параметров, модифицирующих файлы данных (.MYD), создается резервная копия *tbl_name-time*.BAK, где *time* — число, представляющее временную метку. Файл резервной копии записывается в каталог, содержащий табличные файлы. Этот параметр был добавлен в СУБД MySQL 3.23.25, в которой появился оператор `BACKUP TABLE`, но этот параметр не имеет всех функциональных возможностей оператора.

- `--check`, `-c`. Проверяет таблицы на наличие ошибок. Если параметры не определены, эта операция выполняется по умолчанию.
- `--check-only-changed`, `-C`. Проверить таблицы только в том случае, если они подвергались изменениям со времени последней проверки. Этот параметр был добавлен в MySQL 3.23.22. До этого необходимо использовать `--fast`.
- `--correct-checksum`. Для таблиц, созданных с параметром `CHECKSUM = 1`, убедитесь в том, что информация о контрольной сумме правильна. Этот параметр был добавлен в MySQL 4.0.0.
- `--data-file-length=`*n*, `-D` *n*. Определяет максимально возможный размер файла, до которого может увеличиваться файл данных при заполнении. (Это происходит при достижении файлом лимита, установленного MySQL, или ограничениях файловой системы. Кроме того, число строк может также ограничиваться внутренней структурой данных таблицы.) Данное значение задается в байтах. Этот параметр имеет эффект только при совместном использовании с параметрами `--recover` или `--safe-recover`.
- `--fast`, `-F`. Проверять таблицы только в случае, если они не были правильно закрыты. До MySQL 3.23.22 этот параметр действовал как `--check-only-changed`; т.е. таблицы проверяются только в том случае, если они не изменялись с момента последней проверки.
- `--medium-check`, `-m`. Использует более быстрый метод проверки, чем расширенный, но в то же время менее детализированный. (Согласно справочной информации об программе `myisamchk`, этот метод в состоянии обнаружить "только" 99,99% всех ошибок.) В большинстве случаев данного режима такой проверки оказывается более чем достаточно. Данный метод проверки основан на расчете значений CRC для ключей индекса и их сравнения со значениями CRC, рассчитанными на основании индексированных столбцов файла данных.
- `--parallel-recover`, `-p`. Выполняет восстановление как и `--recover`, но с параллельной перестройкой индексов с использованием нескольких процессов. Это может быть быстрее, чем непараллельная перестройка, но этот параметр следует рассматривать как экспериментальный. Этот параметр был добавлен в MySQL 4.0.2.
- `--read-only`, `-T`. Не помечает таблицы как проверенные.
- `--set-auto-increment[=`*n*`]`, `-A[`*n*`]`. Определяет отсчет значений `AUTO_INCREMENT` начиная с числа *n* (или с еще большего значения, если таблица уже содержит записи со значениями `AUTO_INCREMENT`, большими, чем *n*). Если значение *n* не задано, этот параметр устанавливает следующее значение `AUTO_INCREMENT` на единицу большим текущего максимального.
- `--set-character-set=charset`. При перестройке индексов использовать порядок сортировки, принятый для данной кодировки. Эта возможность может пригодиться при переупорядочивании индексов таблиц MyISAM после изменения стандартной кодировки сервера и обычно используется совместно с параметрами `--recover` и `--quick`. Этот параметр был добавлен в MySQL 3.23.14.

- `--sort-recover`, `-n`. Восстановление с принудительной сортировкой, даже если временный файл, необходимый для выполнения этой операции, примет слишком большой размер. Этот параметр был добавлен в MySQL 3.23.22.
- `--start-check-pos=n`. Начать чтение файла данных с позиции *n*. Этот параметр был добавлен в MySQL 3.23.25 и используется только при отладке.
- `--tmpdir=dir_name`, `-t dir_name`. Определяет путь к каталогу, используемому для хранения временных файлов. Если эта переменная не установлена, по умолчанию используется значение переменной среды `TMPDIR` или каталог `/tmp`. Начиная с MySQL 4.1 значение параметра может задаваться в виде списка каталогов, которые могут использоваться для круговой диспетчеризации. В ОС UNIX имена каталогов разделяются двоеточиями, а в ОС Windows точками с запятой.
- `--update-state`, `-U`. `Update` является внутренним флагом, хранящемся в таблице и отражающим его состояние. Помечаются неповрежденные таблицы, а таблицы, в которых произошли ошибки, помечаются как нуждающиеся в восстановлении. Использование этой возможности позволяет повысить эффективность последовательного вызова программы `myisamchk` с заданием параметра `--check-only-changed` для неповрежденных таблиц. Этот параметр был добавлен в MySQL 3.23.14.

Переменные программ `isamchk` и `myisamchk`

Следующие переменные программ `isamchk` и `myisamchk` можно установить с помощью команд, описанных выше, в разделе "Установка программных переменных".

- `key_buffer_size`. Размер буфера, используемого для хранения блоков индексов.
- `read_buffer_size`. Размер буфера чтения.
- `write_buffer_size`. Размер буфера записи.
- `sort_buffer_size`. Размер буфера, используемого для операций сортировки значений ключей. (Используется с параметром `--recover`, но не с `--safe_recover`.)
- `sort_key_blocks`. Эта переменная связана с глубиной структуры B-дерева, используемого для индексов. Ее значение изменять нет необходимости.
- `decode_bits`. Число разрядов, используемых для шифрования сжатых таблиц. Установка больших значений приведет к ускорению выполнения операции, однако потребует большего объема памяти. В большинстве случаев достаточно значения, установленного по умолчанию.

Следующие переменные могут устанавливаться только для `myisamchk`. Они были добавлены в MySQL 4.0.0.

- `ft_max_word_len`. Маскимальная длина слов, которые можно включать в индекс `FULLTEXT`. Более длинные слова игнорируются.
- `ft_max_word_len_for_sort`. Длина слов, слишком коротких для включения в индекс `FULLTEXT` с применением более эффективного алгоритма оптимизации. Более длинные слова вставляются медленнее.

- ft_min_word_len. Минимальная длина слов, которые можно включать в индекс FULLTEXT. Более короткие слова игнорируются.
- myisam_block_size. Размер индексных блоков.

Программы myisampack и pack_isam

Программы myisampack и pack_isam используются для создания сжатых и предназначенных исключительно для чтения таблиц. Они позволяют уменьшить размеры таблиц от 40 до 70% и в то же время обеспечивают высокую скорость доступа. Программа myisampack сжимает таблицы MyISAM и может работать со столбцами всех типов. Программа pack_isam работает с таблицами ISAM, но способна обрабатывать только таблицы, не содержащие столбцов типа BLOB и TEXT.

Программы myisampack и pack_isam появились в MySQL 3.23.19, первая версия появилась под общедоступной лицензией GNU. До этого они предоставлялись только для общей поддержки потребителей, поэтому лучше всего воспользоваться версией СУБД MySQL не ниже 3.23.19.

Для считывания таблиц, сжатых описываемыми здесь программами, не требуется какая-либо специальная версия MySQL. Это делает программы myisampack и pack_isam исключительно полезными для приложений, с помощью которых обеспечивается доступ к таблицам с архивной или энциклопедической информацией. Эта информация, конечно же, доступна только для чтения и не требует обновления. Например, использование сжатых MyISAM-таблиц позволит сохранить больше информации на обычном CD-ROM.

Чтобы преобразовать сжатый файл в обычный изменяемый формат, достаточно запустить одну из команд myisamchk --unpack (для таблиц MyISAM) или isamchk --unpack (для таблиц ISAM).

Программы myisampack и pack_isam сжимают файлы данных, но не затрагивают файлы индексов. Для обновления последних необходимо по завершении работы программы myisampack запустить команду myisamchk --recover --quick, а по завершении работы программы pack_isam — команду isamchk -recover --quick.

Использование

```
myisampack [options] tbl_name ...
pack_isam [options] tbl_name ...
```

В качестве аргумента *tbl_name* может выступать как имя таблицы, так и имя ее файла индекса (MYI-файл для MyISAM-таблицы и ISM-файл для таблицы ISAM). Если файлы таблиц размещаются не в текущем каталоге, необходимо определить полный путь к ним.

Стандартные параметры, поддерживаемые программами myisampack и pack_isam

```
--debug  --silent          --version
--character-sets-dir --help   --verbose
```

Параметр `--character-sets-dir` с MySQL 3.23.33 поддерживается только `myisampack`.

Общие параметры программ `myisampack` и `pack_isam`

- `--backup`, `-b`. Указывает программам `myisampack` и `pack_isam` перед сжатием сделать резервную копию каждой заданной аргументом *`tbl_name`* таблицы. Файл резервной копии имеет имя *`tbl_name`*`.OLD`.
- `--force`, `-f`. Указывает выполнять сжатие даже в том случае, если результирующий файл больше исходного или если для таблицы временный файл уже создан. Как правило, если программы `myisampack` и `pack_isam` находят файл *`tbl_name`*`.TMD`, то выводят сообщение об ошибке и завершают свою работу. Это объясняется тем, что существование такого файла может указывать на проверку данной таблицы другой программой. Хотя временный файл будет также существовать и в том случае, если программа проверки внезапно завершила свою работу и не успела корректно удалить файл. Если администратор уверен, что существование файла объясняется именно последней причиной, можно воспользоваться параметром `--force`.
- `--join=`*`join_tbl`*, `-j` *`join_tbl`*. Объединяет все указанные в командной строке таблицы в одну сжатую таблицу с именем *`join_tbl`*. Для этого все объединяемые таблицы должны иметь одинаковую структуру (а именно: названия столбцов, их типы и индексы).
- `--test`, `-t`. Работает в режиме тестирования. В этом режиме выводится информация о результатах, которые можно было бы получить в случае реального сжатия таблицы.
- `--tmpdir=`*`path_name`*, `-T` *`path_name`*. Определяет путь к каталогу, используемому для хранения временных файлов.
- `--wait`, `-W`. Ожидает некоторое время и предпринимает повторную попытку сжатия, если на момент запуска программы команда кем-либо используется. Не рекомендуется обрабатывать таблицу, которая в процессе сжатия изменяется другими пользователями.

Специальные параметры программы `pack_isam`

Если программа `myisampack` не содержит каких-либо специальных параметров, то для программы `pack_isam` один такой параметр существует.

- `--packlength=`*n*, `-p` *n*. С помощью этого параметра можно задать объем, выделяемый для хранения длины записи, равным *n* байт, где *n* может быть целочисленным значением в диапазоне от 1 до 3. Программа `pack_isam` автоматически пытается определить число байт, необходимое для записи длины каждой строки сжимаемой таблицы. В некоторых случаях она может определить, что для хранения необходим даже меньший объем, чем был задан. Если это действительно так, программа `pack_isam` выдаст сообщение о данном факте. Пользователь имеет возможность позже запустить программу `pack_isam` снова и задать параметром `--packlength` меньшую длину. Это позволит сэкономить дисковое пространство.

Программа mysql

Клиент mysql представляет собой интерактивную программу, используемую для подключения к серверу, отправки запросов и просмотра результатов их выполнения. Программа mysql может работать в пакетном режиме, что позволяет исполнять записанные в файле запросы, перенаправив командный вход на чтение данного файла. Это можно осуществить, например, следующим образом:

```
% mysql -u sampadm -p -h cobra.snake.net sampdb < my_query_file
```

При запуске в интерактивном режиме программа mysql выводит на экран строку подсказки "mysql>", показывая, что ожидает ввода команды. Для запуска запроса введите его текст (используя, если необходимо, несколько строк), завершив ввод символом ";" (точка с запятой) или "\g". Программа отправит запрос к серверу, отобразит результаты выполнения этого запроса и снова выведет строку подсказки, демонстрируя свою готовность к выполнению следующего запроса.

Строка подсказки mysql (см. ниже) может изменяться в зависимости от содержания, которое программа ожидает от пользователя.

mysql> Ожидается первая строка нового запроса.
-> Ожидается следующая строка текущего запроса.
'> Ожидается завершение выделенной одинарными кавычками строки текущего запроса.
"> Ожидается завершение выделенной двойными кавычками строки текущего запроса.

Строки подсказки "'>" и "">" отражают, что пользователь начал выделенную одинарными или двойными кавычками строку и не ввел в конце завершающие кавычки. Как правило, это происходит из-за забывчивости пользователей. Для исправления подобной оплошности и выхода из строкового режима введите завершающий символ и затем последовательность "\c", чтобы отменить текущий запрос.

В интерактивном режиме работы mysql все запросы сохраняются в файле журнала. По умолчанию таковым является файл $HOME/.mysql_history, однако с помощью переменной MYSQL_HISTORY его можно изменить. Из созданного журнала запросы можно повторно вызывать и запускать, предварительно редактируя или оставляя их неизменными. В расположенном ниже списке представлены некоторые комбинации клавиш команд редактирования, а также их описание.

Комбинация клавиш	Значение
<↑>, <Ctrl+P>	Повторный вызов предыдущей строки
<↓>, <Ctrl+N>	Повторный вызов следующей строки
<←>, <Ctrl+B>	Перемещение курсора влево (назад)
<→>, <Ctrl+F>	Перемещение курсора вправо (вперед)
<Esc+Ctrl+B>	Перемещение назад на одно слово
<Esc+Ctrl+F>	Перемещение вперед на одно слово
<Ctrl+A>	Перемещение курсора в начало строки
<Ctrl+E>	Перемещение курсора в конец строки
<Ctrl+D>	Удаление символа, выделенного курсором

Комбинация клавиш	Значение
<Delete>	Удаление символа слева от курсора
<Esc+D>	Удаление слова
<Esc+Backspace>	Удаление слова слева от курсора
<Ctrl+K>	Удаление части строки от курсора до конца строки
<Ctrl+_>	Отмена последнего изменения

Некоторые параметры запрещают использование файла журнала. В частности, к таким относятся параметры, определяющие не интерактивное использование `mysql`, например `--batch`, `--html` и `--quick`.

Под управлением Windows редактирование командной строки в `mysql` может не работать. В этом случае можно попробовать работать с клиентом `mysqlc`, равнозначным клиенту `mysql`, но для него функция редактирования командной строки компилируется отдельно. (Это большое преимущество, однако есть и недостатки, заключающиеся в том, что программа `mysqlc` до сих пор не модифицирована и "не понимает" новейших параметров и внутренних команд.)

Программе `mysqlc` для работы требуется библиотека `cygwinb19.dll`. Если эта библиотека еще не установлена в том же каталоге, что и `mysqlc` (каталог `bin` для нашего дистрибутива MySQL), ее можно найти в каталоге `lib` дистрибутива и скопировать в каталог `bin` или в системный каталог Windows.

Использование

```
mysql [options] [db_name]
```

Определение аргумента *db_name* делает указанную базу данных текущей (базой данных по умолчанию) на время данного сеанса работы. Если же этот аргумент не задан, `mysql` запускается без текущей базы данных и в дальнейшем пользователю придется либо определять ссылки на все таблицы с именем базы данных, либо установить базу данных по умолчанию с помощью оператора `USE db_name`.

Стандартные параметры, поддерживаемые программой `mysql`

```
--character-sets-dir          --host       --silent
--compress                               --password   --socket
--debug                                  --pipe       --user
--default-character-set       --port       --verbose
--help       --set-variable   --version
```

Параметры `--character-sets-dir` и `--default-character-set` можно использовать начиная с MySQL 3.23.15. Начиная с MySQL 4 `mysql` также поддерживает стандартные параметры протокола SSL.

Для усиления эффекта параметры `--silent` и `--verbose` можно определять несколько раз.

Специальные параметры `mysql`

- `--auto-rehash` (*boolean*). В процессе запуска программа `mysql` упорядочивает названия баз данных, таблиц и столбцов для создания информаци-

онной структуры, позволяющей быстро завершать эти названия при вводе пользователем. (Создавая запрос, введите начальную часть уникального названия таблицы или столбца и нажмите клавишу <Tab>. Программа `mysql` завершит ввод этого имени.) Этот параметр отменяет сложный процесс упорядочения, благодаря чему `mysql` запускается значительно быстрее, особенно если обрабатывается большое число таблиц. Чтобы активизировать возможность завершения имен уже после запуска `mysql`, можно запустить команду `rehash` из строки подсказки "`mysql>`".

Этот параметр был добавлен в MySQL 4.0.2. До версии 4.0.2 для отключения хэшинга имен использовался параметр `--no-auto-rehash`.

- `--batch`, `B`. Определяет запуск `mysql` в пакетном режиме. Результаты выполнения запросов выводятся в специальном формате (каждая запись в отдельной строке, а значения столбцов разделены символами табуляции). Этот формат оказывается исключительно удобным для создания отчетов, переносимых в другие программы, например электронные таблицы. Выводимые результаты выполнения запросов по умолчанию включают первую строку с заголовками столбцов. Для исключения этой строки используется параметр `--skip-column-names`.
- `--column-names` (*boolean*). Отображать имена столбцов вместе с результатами запросов. Этот параметр был добавлен в MySQL 4.0.2.
- `--database=`*db_name*, `-D` *db_name*. База данных по умолчанию. Этот параметр был добавлен в MySQL 3.22.29.
- `--debug-info`, `-T` (*boolean*). Выводит отладочную информацию по завершении работы программы.
- `--execute=`*query*, `-e` *query*. Выполняет запрос и завершает работу. Текст запроса необходимо заключить в кавычки, чтобы рабочая оболочка не интерпретировала его как несколько аргументов командной строки. Несколько исполняемых запросов следует разделять в строке *query* с помощью точек с запятыми.
- `--force`, `-f` (*boolean*). Обычно при выполнении программой `mysql` запросов из файла работа завершается в случае возникновения ошибки. Этот параметр заставляет `mysql` продолжить работу, независимо от появления ошибок.
- `--html`, `-H` (*boolean*). Выводит данные в виде кода HTML. Возможность использования этого параметра появилась в MySQL 3.22.26.
- `--i-am-a-dummy` (*boolean*). Этот параметр является синонимом параметра `--safe-updates`. Этот параметр был добавлен в MySQL 3.23.11.
- `--ignore-space`, `-i`. Указывает серверу пропускать пробелы между названиями функций и символом "`(`", отражающим начало списка аргументов. Как правило, между названием функции и открывающей круглой скобкой пробелы использовать нельзя. Этот параметр приводит к рассмотрению имен функций в качестве резервных слов.

  Этот параметр был добавлен в MySQL 3.22.21.

- `--line-numbers` (*boolean*). Отображать номера строк в диагностических сообщениях. Этот режим задается по умолчанию; для подавления вывода номеров строк используется параметр `skip-line-numbers`. Этот параметр был добавлен в MySQL 4.0.2. В версиях 3.22.5–4.0.2 номера строк можно было отключить параметром `--skip-line-numbers` или `-L`.
- `--local-infile` (*boolean*). Разрешение или запрет режима LOAD DATA LOCAL. Начиная с MySQL 3.23.49 режим LOCAL может иметься, но быть отключенным по умолчанию. Если LOAD DATA LOCAL приводит к ошибке, попробуйте еще вызвать `mysql` с параметром `--local-infile`. Этот параметр может использоваться для отключения режима LOCAL, если он включен: начиная с MySQL 4.0.2 можно воспользоваться параметром `--disable-local-infile`, а до этой версии — `--local-infile=0`.

  Этот параметр неэффективен, если сервер был настроен с отключением использования режима LOCAL.
- `--named-commands`, `-G` (*boolean*). Разрешение длинной формы задания внутренних команд `mysql` в начале командной строки. Если эта возможность отключена параметром `--disable-named-commands`, длинная форма команд разрешена только для первичной подсказки и запрещена на вторичных подсказках (т.е. они отключены на второй и всех последующих строках многострочных операторов).

  Этот параметр имеет сложную историю создания. Именованные команды были разрешены по умолчанию до MySQL 3.23.22, но потом были запрещены. Их можно отключить параметром `--no-named-commands` в версиях MySQL 3.23.11–3.23.22, включить с помощью параметра `--enable-named-commands` в версиях 3.23.23–4.0.1 и управлять любым способом с помощью логического параметра `--named-commands` после этого.
- `--no-auto-rehash`, `-A`. См. описание параметра `--auto-rehash`; начиная с MySQL 4.0.2 параметр не используется.
- `--no-beep`, `-b` (*boolean*). Не подавать звуковой сигнал при возникновении ошибок. Этот параметр был добавлен в MySQL 4.0.2.
- `--no-named-commands`, `-g`. См. описание параметра `--named-commands`; начиная с MySQL 4.0.2 параметр не используется.
- `--no-pager`. См. описание параметра `--pager`; этот параметр был добавлен в MySQL 3.23.28, но не используется начиная с MySQL 4.0.2 в пользу параметра `--disable-pager`.
- `--no-tee`. См. описание параметра `-tee`. Этот параметр был добавлен в MySQL 3.23.28, но не используется в пользу параметра `--disable-tee` начиная с MySQL 4.0.2.
- `--one-database`, `-o`. Этот параметр был использован для восстановления баз данных по контексту регистрационного журнала восстановления. Он сообщает программе `mysql` для модификации базы данных по умолчанию (база данных, указанная в командной строке). Восстановление других баз данных игнорируется. Если база данных задана в командной строке, восстановление не выполняется.

- `--pager[=program]`. Использование программы постраничного вывода (например, `/bin/more` или `/bin/less`) для отображения объемного результата работы запроса в постраничном режиме. При отсутствии программы постраничного вывода имя этой программы берется по значению переменной среды PAGER. Постраничный вывод отсутствует при пакетном режиме работы и не работает под управлением ОС Windows. Постраничный вывод отключается параметром `--disable-pager`.

  Параметр `--pager` был добавлен в MySQL 3.23.28. До версии 4.0.2 отключение постраничного вывода производилось параметром `--no-pager`, а не `--disable-pager`.

- `--prompt=str`. Изменить первоначальную строку подсказки "`mysql>`" на строку, заданную значением *str*. Эта строка может содержать специальные последовательности (см. раздел "Определение последовательностей подсказки mysql"). Этот параметр был добавлен в MySQL 4.0.2.

- `--quick, -q`. Обычно `mysql` перед отображением результирующего набора запроса на экране полностью извлекает этот результат с сервера. Этот параметр заставляет программу отображать каждую строку сразу после получения с сервера, что способствует уменьшению загрузки памяти и позволяет успешно выполнять запросы, которые в других условиях могут работать с ошибкой. Однако этот параметр не следует задавать для интерактивного использования. Если один пользователь временно приостановит работу `mysql`, сервер будет ждать продолжения, что не замедлит отразиться на работе других клиентов.

- `--raw, -r(boolean)`. Записывает значения столбцов, не удаляя специальные символы. Этот параметр используется вместе с параметром `--batch`.

- `--safe-updates, -U (boolean)`. Этот параметр накладывает определенные ограничения на ваши возможности и может пригодиться для начинающих пользователей MySQL. Операторы модификации (операторы, которые модифицируют данные) разрешены только тогда, когда записи модифицируются по значениям ключа или при использовании продолжения `LIMIT`. Это позволит предотвратить запросы, которые ошибочно изменяют или вытирают большие части таблиц. Кроме того, результирующие наборы, полученные с помощью простых запросов и запросов объединения, ограничены одной тысячей или одним миллионом строк соответственно. Эти пределы могут изменяться в зависимости от установок переменных `select_limit` и `max_join_size`. Этот параметр был добавлен в MySQL 3.23.11.

- `--skip-column-names, -N`. Подавляет вывод имен столбцов в результатах запросов. Этого же результата можно добиться, дважды определив параметр `--silent`. Этот параметр был добавлен в MySQL 3.22.20. Форма `-N` не используется начиная с MySQL 4.0.2. См. описание параметра `--column-names`.

- `--skip-line-numbers, -L`. Не записывает номера строк при возникновении ошибок. Этот параметр был добавлен в MySQL 3.22.5. Форма `-L` не применяется начиная с MySQL 4.0.2. См. описание `--line-numbers`.

- `--table, -t(boolean)`. Выводит результат выполнения запроса в табличном формате: строки разделены вертикальными линиями и выровнены по

вертикали. Это стандартный формат вывода при запуске mysql в пакетном режиме.

- `--tee=file_name`. Добавить копию всего вывода в указанный файл. Копирование вывода можно отключить с помощью параметра `--disable-tee`. Этот параметр не работает в пакетном режиме. Он был добавлен в MySQL 3.23.28. До версии 4.0.2 файл tee отключался параметром `--no-tee`, а не `--disable-tee`.
- `--unbuffered`, `-n`. После каждого запроса очищает буфер, используемый для обмена с сервером.
- `--vertical`, `-E`. Выводит результирующий набор запроса вертикально. Каждая запись такого набора отображается в виде набора строк. Одна строка соответствует одному столбцу. Строки состоят из названий столбцов и их значений. Перед каждой записью выводится строка с номером записи внутри результирующего набора. Подобный вертикальный формат отображения может быть полезен для случая, когда длина строк результирующего набора очень велика.

  Если этот параметр не определен, задавать вертикальный формат для отдельных запросов можно посредством завершения их последовательностью "\G" вместо ";" или "\g".

  Этот параметр впервые появился в MySQL 3.22.5.
- `--wait`, `-W`. Если подключиться к серверу невозможно, запущенная с этим параметром программа немного ждет и снова пытается установить соединение.
- `--xml`, `-X (boolean)`. Вывод в формате XML. Этот параметр был добавлен в MySQL 4.0.0.

Переменные mysql

Следующие переменные программы mysql могут быть установлены с помощью команд, описанных выше, в разделе "Установка программных переменных".

- `connect_timeout`. Время в секундах до наступления тайм-аута при подключении к серверу. Этот параметр был добавлен в MySQL 3.23.28.
- `max_allowed_packet`. Максимальный размер буфера, используемого для обмена данными между сервером и клиентом.
- `max_join_size`. Ограничение количества строк, возвращаемых оператором SELECT, в котором применяется объединение, если задан параметр `--safe-updates`. Этот параметр был добавлен в MySQL 3.23.11.
- `net_buffer_length`. Исходный размер буфера, используемого для обмена данными между сервером и клиентом. Этот буфер может быть расширен до `max_allowed_packet` байтов.
- `select_limit`. Ограничение количества строк, возвращаемых оператором SELECT, если задан параметр `--safe-updates`. Этот параметр был добавлен в MySQL 3.23.11.

Команды mysql

Помимо отправляемых на сервер MySQL операторов SQL, программа mysql распознает еще несколько своих собственных команд. Каждая такая команда должна задаваться отдельной строкой, в конце которой ставить точку с запятой необязательно. Большинство команд имеет длинную форму, состоящую из полного слова, и короткую форму, состоящую из обратной косой черты и следующей за ней буквы. Регистр символов в командах длинной формы не учитывается. Представленные в короткой форме команды необходимо задавать с учетом определенного ниже регистра.

Заметим, что при отключенных именованных командах (например, параметре `--disable-named-commands`) длинные имена команд распознаются только при длинных подсказках "`mysql>`".

- `clear`, `\c`. Отменяет текущий запрос. Текущим называется запрос, который вводится в командной строке. Следовательно, данная команда не в состоянии отменить запрос, который уже отправлен на сервер и результаты которого выводятся на экран.
- `connect [`*db_name* `[`*host_name*`]]`, `\r [`*db_name* `[`*host_name*`]]`. Подключается к указанной базе данных на определенном компьютере. Если имя базы данных или компьютера в командной строке не указаны, вместо них используются последние значения текущего ссанса `mysql`.
- `edit`, `\e`. Редактирует текущий запрос. Используемый для этих целей редактор определяется значениями переменных `EDITOR` и `VISUAL`. Если ни одно из них не установлено, запускается редактор `vi`. Но эта команда под ОС Windows не работает.
- `ego`, `\G`. Отправляет текущий запрос на сервер и вертикально выводит его результаты. Эта команда впервые появилась в MySQL 3.22.11.
- `exit`. Эта команда аналогична команде `quit`.
- `go`, `\g`, `;`. Отправляет текущий запрос на сервер и выводит результаты его выполнения.
- `help`, `\h`, `?`. Отображает справочную информацию о доступных командах `mysql`.
- `nopager`, `\n`. Отключает постраничный вывод и отправляет вывод на стандартное устройство вывода. Эта команда была добавлена в MySQL 3.23.28. Под управлением ОС Windows данная команда не работает.
- `notee`, `\t`. Прекращает запись в файл `tee`. Эта команда была добавлена в MySQL 3.23.28.
- `pager [`*program*`]`, `\P [`*program*`]`. Передает вывод на программу постраничной печати, заданную параметром *program* или переменной среды PAGER. Эта команда была добавлена в MySQL 3.23.28, но под управлением ОС Windows она не работает.
- `print`, `\p`. Распечатывает текущий запрос (его текст, а не результаты выполнения).

- `prompt` *`arguments`*, `\R` *`arguments`*. Переопределяет первичную подсказку `mysql`. Все символы, следующие за первым пробелом после ключевого слова подсказки, становятся частью строки подсказки, включая и другие пробелы. Эта строка может содержать специальные последовательности, которые будут описаны в разделе "Определение последовательностей подсказки mysql". Для того чтобы вернуться к первоначальной подсказке, задайте подсказку или `\R` без аргумента. Эта команда была добавлена в MySQL 4.0.2.
- `quit`, `\q`. Выход из `mysql`.
- `rehash`, `\#`. Хранит данные, необходимые для автоматического завершения имен баз данных, таблиц и столбцов. Более детально об этом средстве рассказывается в описании параметра `--no-auto-rehash`.
- `source` *`file_name`*, `\.` *`file_name`*. Читает и выполняет запросы, содержащиеся в указанном файле. Если имена включают разделители в виде обратной косой черты ('\'), то в ОС Windows необходимо использовать вместо них косую черту ('/'). Эта команда была добавлена в MySQL 3.23.9.
- `status`, `\s`. Извлекает и выводит информацию о состоянии сервера. Эта команда весьма полезна для определения версии сервера и текущей базы данных, безопасности соединения и т.д.
- `system` *`command`*, `\!` *`command`*. Выполняет команду с помощью стандартной оболочки. Эта команда была добавлена в 4.0.1. Для ОС Windows эта команда отсутствует.
- `tee` *`file_name`*, `\T` *`file_name`*. Копирует вывод в конец указанного файла. Эта команда была добавлена в MySQL 3.23.28.
- `use` *`db_name`*, `\u` *`db_name`*. Устанавливает указанную базу данных в качестве текущей.

Определение последовательностей подсказки `mysql`

Для изменения первоначальной подсказки "`mysql>`" могут использоваться переменная среды MYSQL_PS1, параметр `--prompt` или команда `prompt`. Для определения могут использоваться следующие управляющие последовательности.

Последовательность	Описание
\c	Вводимая строка
\d	Текущая база данных или "(none)", если никакая база данных не выбрана
\D	Полная дата и время
\h	Текущий узел
\m	Минуты
\o	Номер месяца
\O	Аббревиатура месяца (три буквы)
\p	Текущий номер порта, имя разъема или именованный канал
\P	Индикатор a.m./p.m.
\r	Час (12-часовой формат)

Последовательность	Описание
\R	Час (24-часовой формат)
\s	Секунды
\S	Точка с запятой
\t	Табуляция
\u	Имя текущего пользователя, без имени узла
\U	Имя текущего пользователя с именем узла
\v	Версия сервера
\y	Год (две цифры)
\Y	Год (четыре цифры)
\w	День недели (три буквы)
\'	Одинарная кавычка
\"	Двойная кавычка
_	Символ пробела
\	Символ пробела (последовательность содержит обратную косую черту и пробел)
\\	Литерал '\'
\n	Новая строка (перевод строки)
\x	Литерал 'x' для любого символа 'x', не перечисленного выше

Сценарий mysqlaccess

Этот сценарий позволяет подключиться к серверу, извлечь информацию о привилегиях доступа и протестировать результаты присвоения привилегий. В процессе работы сценарий `mysqlaccess` использует копии таблиц `user`, `db` и `host` базы данных `mysql`. (Но этот сценарий нельзя использовать для проверки привилегий доступа к таблицам и столбцам.) Внесенные во временные таблицы изменения можно перенести в таблицы реальных баз данных `mysql`.

Чтобы максимально воспользоваться преимуществами программы `mysqlaccess`, пользователь должен обладать достаточными привилегиями для доступа к таблицам разрешений.

Использование

```
mysqlaccess [host_name [user_name [db_name]]] options
```

Стандартные параметры, поддерживаемые сценарием mysqlaccess

```
--host      --password      --user      --version
```

Специальные параметры mysqlaccess

- `--brief`, `-b`. Отображает результаты в однострочном формате.
- `--commit`. Копирует временные таблицы разрешений в базу данных `mysql`. Не забудьте после задания этого параметра запустить команду `mysqladmin flush-privileges`, чтобы сервер "заметил" изменения.

- `--copy`. Загружает содержимое таблиц разрешений во временные таблицы.
- `--db=db_name`, `-d db_name`. Задает имя базы данных.
- `--debug=n`. Устанавливает уровень отладки. Значение *n* может быть целым числом от 0 до 3.
- `--howto`. Выводит несколько примеров использования программы `mysqlaccess`.
- `--old_server`. Определяет, что версия сервера MySQL превышает 3.21. Это заставляет `mysqlaccess` вносить некоторые изменения в отправляемые на сервер запросы.
- `--plan`. Отображает список усовершенствований, планируемых в будущих версиях `mysqlaccess`.
- `--preview`. Показывает различия между реальными и временными таблицами разрешений.
- `--relnotes`. Распечатывает замечания версии `mysqlaccess`.
- `--rhost=host_name`, `-H host_name`. Определяет удаленный компьютер с сервером, к которому осуществляется подключение.
- `--rollback`. Отменяет изменения, внесенные во временные таблицы разрешений.
- `--spassword=pass_val`, `-P pass_val`. Определяет пароль для суперпользователя MySQL (пользователя, обладающего достаточными привилегиями для изменения таблиц разрешений).
- `--superuser=user_name`, `-U user_name`. Определяет имя суперпользователя MySQL.
- `--table`, `-t`. Выводит результаты в табличном формате.

Программа `mysqladmin`

Программа `mysqladmin` предназначена для взаимодействия с сервером MySQL и выполнения административных операций. Ее же можно использовать для получения информации о сервере, управления его функционированием, установки паролей, а также для создания или удаления баз данных.

Использование

```
mysqladmin [options] command ...
```

Стандартные параметры, поддерживаемые программой `mysqladmin`

```
--character-sets-dir --password          --socket
--compress           --pipe              --user
--debug              --port              --verbose
--help               --set-variable      --version
--host               --silent
```

Параметр `--silent` приводит к "тихому" завершению работы `mysqladmin`, если невозможно подключиться к серверу. Параметр `--verbose` был добавлен в

3.22.30. С ним mysqladmin печатает более подробную информацию. Параметр --character-sets-dir был добавлен в 3.23.21. Начиная с MySQL 4 программа mysqladmin также поддерживает стандартные параметры SSL.

Специальные параметры mysqadmin

- --count=*n*, -c *n*. Количество итераций, необходимых при заданном параметре --sleep. Этот параметр был добавлен в MySQL 4.0.3.
- --force, -f. Этот параметр имеет две особенности. Во-первых, он указывает программе mysqladmin не запрашивать подтверждение для команды drop *db_name*. Во-вторых, если в командной строке задано несколько команд, mysqladmin попытается обязательно все их выполнить, даже если в процессе этого выполнения возникает ошибка. В обычных условиях программа mysqladmin завершает работу сразу при появлении первой ошибки.
- --relative, -r (*boolean*). Отражает разницу между текущим и предыдущим значениями при задании вместе с параметром --sleep. В настоящее время этот параметр работает только с командой extended-status.
- --sleep=*n*, -i *n*. Выполняет заданные в командной строке команды повторно с интервалом *n* секунд между повторениями.
- --timeout=*n*, -t *n*. Ожидает *n* секунд перед попыткой подключения к серверу. Этот параметр впервые появился в MySQL 3.22.1 и изъят в 3.23.29, когда была добавлена переменная connect_timeout.
- --vertical, -E (*boolean*). Этот параметр аналогичен -relative, но отображает результаты вертикально; был добавлен в MySQL 3.23.14.
- --wait[=*n*], -w[*n*]. Определяет число попыток повторного подключения к серверу, если невозможно установить соединение. Если значение *n* не определено, оно по умолчанию устанавливается равным 1. Если *n* задано после -w, не должно быть пробела, так как в противном случае значение не будет правильно интерпретировано.

Переменные программы mysqladmin

Следующие переменные программы mysqadmin могут быть установлены с помощью команд, описанных выше, в разделе "Установка программных переменных".

- connect_timeout. Время (в секундах) до наступления тайм-аута при подключении к серверу. Этот параметр был добавлен в MySQL 3.23.29, когда был упразднен параметр --timeout.
- shutdown_timeout. Время ожидания (в секундах) до выключения сервера. Этот параметр был добавлен в MySQL 3.23.34.

Команды mysqladmin

Пользователи этой программы имеют возможность задать в одной строке одну или несколько описанных далее команд с последующим указанием параметров. Имя команды может быть сокращено до префикса, если таковой является уни-

кальным. Например, имя processlist можно укоротить до префикса process или proc, но не до префикса p.

Некоторые из приведенных ниже команд имеют аналоги среди операторов SQL (о чем упоминается в их описаниях). Более детально о значениях операторов SQL см. в приложении Г, "Синтаксис SQL".

- create *db_name*. Создает новую базу данных с заданным именем. Эта команда аналогична SQL-оператору CREATE DATABASE *db_name*.
- drop *db_name*. Удаляет базу данных с заданным именем и все содержащиеся в ней таблицы. Эту команду следует использовать с большой осторожностью, поскольку ее результаты нельзя отменить. Как правило, если не задан параметр --force, программа mysqlaccess перед выполнением команды drop запрашивает подтверждение. Эта команда аналогична SQL-оператору DROP DATABASE *db_name*.
- debug. Выводит отладочную информацию.
- extended-status. Выводит имена и значения переменных состояния сервера. Эта команда аналогична SQL-оператору SHOW STATUS. Команда extended-status впервые появилась в MySQL 3.22.10.
- flush-hosts. Обновляет кэш главного компьютера. Эта команда аналогична SQL-оператору FLUSH HOSTS.
- flush-logs. Обновляет (закрывает и заново открывает) файлы журналов. Эта команда аналогична SQL-оператору FLUSH LOGS.
- flush-privileges. Повторно загружает таблицы разрешений. Эта команда аналогична SQL-оператору FLUSH PRIVILEGES.
- flush-status. Очищает переменные состояния. (Сбрасывает некоторые счетчики до нуля.) Эта команда аналогична SQL-оператору FLUSH STATUS.
- flush-tables. Обновляет кэш таблиц. Эта команда аналогична SQL-оператору FLUSH TABLES.
- flush-threads. Обновляет кэш потока.
- kill *id,id,...* . Завершает работу потоков, определенных с помощью идентификаторов. При определении нескольких номеров между ними нельзя оставлять пробелы, чтобы не "запутать" следующую за kill команду. Для определения работающих в настоящее время потоков можно воспользоваться командой mysqladmin processlist. Эта команда подобна запуску SQL-оператора KILL для каждого идентификатора.
- password *new_password*. Изменяет пароль для пользователя MySQL, который подключается к серверу после запуска команды mysqladmin, на новый *new_password*. Пользователь, изменяющий пароль, должен иметь возможность подключиться к серверу MySQL от имени пользователя, чей пароль изменяется. Это будет служить лучшим подтверждением того, что старый пароль также знаком администратору. Эта команда аналогична SQL-оператору SET PASSWORD.
- ping. Проверяет работу сервера MySQL.

- `processlist`. Отображает список выполняемых в настоящее время потоков сервера. Эта команда аналогична SQL-оператору `SHOW PROCESSLIST`. С параметром `--verbose` действие этой команды аналогично `SHOW FULL PROCESSLIST`.
- `refresh`. Обновляет кэш таблиц и таблицы разрешений, а также закрывает и снова открывает файлы журналов. Если сервер является главным сервером репликации, эта команда удаляет журналы двоичных обновлений в индексном файле двоичных обновлений и усекает этот индекс. Если это подчиненный сервер, то действие этой команды заключается в "забывании" его положения в главном журнале.
- `reload`. Повторно загружает таблицы разрешений. Эта команда аналогична SQL-оператору `FLUSH PRIVILEGES`.
- `shutdown`. Указывает серверу завершить работу.
- `start-slave`. Запуск подчиненного сервера репликации. Эта команда аналогична оператору `SLAVE START`. Она была добавлена в MySQL 3.23.16.
- `status`. Выводит краткую информацию о состоянии сервера.
- `stop-slave`. Остановка подчиненного сервера репликации. Эта команда аналогична оператору `SLAVE STOP`; была добавлена в MySQL 3.23.16.
- `variables`. Выводит имена и значения переменных сервера. Эта команда аналогична оператору `SHOW VARIABLES`. Начиная с версии MySQL 4.0.3, в которой поддерживаются как глобальные, так и сеансовые переменные, эта команда аналогична оператору `SHOW GLOBAL VARIABLES`. (Аналога оператору `SHOW SESSION VARIABLES` не существует, так как в этом нет никакого смысла.)
- `version`. Извлекает и отображает строку с информацией о версии сервера. Эта информация аналогична информации, возвращаемой функцией `VERSION()` (см. приложение B, "Операторы и функции").

Программа `mysqlbinlog`

Программа `mysqlbinlog` отображает содержимое файла регистрации двоичных обновлений. Эта программа может считывать файлы регистрации, расположенные на локальном компьютере, или подключаться к удаленному серверу и считывать журналы оттуда.

Формат журналов двоичного обновления время от времени изменяется. Во избежание проблем с совместимостью рекомендуется использовать версию `mysqlbinlog`, соответствующую версии сервера.

Использование

```
mysqlbinlog [options] file_name ...
```

Стандартные параметры, поддерживаемые программой `mysqlbinlog`

```
--debug  --host      --port   --version
--help   --password  --user
```

Параметры `--host`, `--password`, `--port` и `--user` используются, если необходимо подключиться к удаленному серверу для получения доступа к журналам регистрации. Иначе программа `mysqlbinlog` считывает локальные файлы регистрации, не подключаясь к серверу.

Специальные параметры `mysqlbinlog`

- `--database=db_name`, `-d db_name`. Выбрать операторы из журнала регистрации только для указанной базы данных. Этот параметр предназначен только для чтения локальных журналов.
- `--offset=n`, `-o n`. Пропустить первые *n* записей в регистрационном журнале.
- `--position=n`, `-j n`. Начать чтение регистрационного файла с позиции *n*.
- `--result-file=file_name`, `-r file_name`. Записать вывод в указанный файл.
- `--short-form`, `-s`. Отобразить из журнала только операторы; пропускать любую дополнительную информацию.
- `--table=tbl_name`, `-t tbl_name`. Получить общий дамп указанной таблицы.

Сценарий `mysqlbug`

В одной из глав *Справочного руководства по MySQL* детально описывается процедура заполнения отчета об ошибках. Воспользовавшись приведенными рекомендациями, вы будете уверены, что создание отчета окажется эффективным, а его содержимого будет достаточно для устранения проблемы. Основная часть описанной процедуры связана с использованием сценария `mysqlbug`. Именно этот сценарий применяется для создания и отправки отчета об ошибках в почтовый список MySQL. Сценарий `mysqlbug` собирает информацию о рабочей системе и конфигурации MySQL, после чего открывает перед пользователем окно редактора с содержимым отсылаемого сообщения. (Программа `mysqlbug` делает попытку определить используемый редактор, для чего просматриваются переменные среды `VISUAL EDITOR`. Если ни одна из переменных не установлена, `mysqlbug` использует редактор `emacs`.) Добавьте в это сообщение всю информацию, которую считаете нужным сообщить, сохраните изменения и закройте редактор. Сценарий `mysqlbug` предложит отправить созданное сообщение и в случае подтверждения отправит его.

Настоятельно рекомендуется использовать `mysqlbug` только в случае действительно серьезных проблем. Довольно часто предполагаемая "ошибка" вовсе таковой не является и уже давно описана в *Справочном руководстве по MySQL*. Так, например, довольно часто к ошибкам причисляют возвращение оператором `DELETE FROM tbl_name` нулевого значения. Действительно, так зачастую бывает, но это — нормальное "поведение" оператора. В справочном руководстве указывается, как подсчитать число удаленных строк, а также предоставляется много другой полезной информации. Другим не менее ценным источником информации является список рассылки MySQL. Подключиться к нему можно по следующему адресу: `http://www.mysql.com/documentation/`. Сценарий `mysqlbug` представляет со-

бой сценарий оболочки и не работает под управлением ОС Windows. Здесь можно использовать возможности приложения MySQLAdmin или MySQLCCWindows. В качестве шаблона сообщения о замеченных ошибках может использоваться файл mysqlbug.txt, который располагается в корне дистрибутива MySQL. Заполните его и перешлите по адресу bugs@lists.mysql.com.

Использование

```
mysqlbug [address]
```

Отчет об ошибках по умолчанию отправляется в почтовый список MySQL. Если определить в командной строке другой адрес, то отчет будет отправлен именно по нему. Можно определить свой собственный адрес, чтобы проверить работу mysqlbug. Это будет нелишним для тех, кто использует программу mysqlbug впервые и плохо знаком с принципами ее работы.

Программа mysqlcheck

mysqlcheck представляет собой программу, предназначенную для проверки и восстановления таблиц. Она предоставляет в распоряжение администратора интерфейс командной строки с операторами CHECK TABLE, ANALYZE TABLE, OPTIMIZE TABLE и REPAIR TABLE. Она подобна программе myisamchk, но используется при работающем сервере. Программа mysqlcheck отсылает на сервер административные запросы на выполнение. Данная программа не похожа на программу myisamchk, которая работает непосредственно на табличных файлах и требует координации доступа к таблицам с сервером или отключения сервера.

Программа mysqlcheck была добавлена в MySQL 3.23.38. Все параметры mysqlcheck поддерживаются таблицами типа MyISAM. Кроме того, программа mysqlcheck может анализировать таблицы BDB и, начиная с MySQL 3.23.40, анализировать таблицы типа InnoDB.

Использование

Программа mysqlcheck может работать в любом из трех режимов:

```
mysqlcheck [options] db_name [tbl_name] ...
mysqlcheck [options] --databases db_name ...
mysqlcheck [options] --all-databases
```

В первом случае проверяются заданные таблицы в указанной базе данных. Если конкретно не указана ни одна из таблиц, программа mysqlcheck проверяет в базе данных все таблицы. Во втором случае все аргументы считаются именами баз данных, и программа mysqlcheck проверяет таблицы в каждой из них. В третьем случае проверяются все таблицы во всех базах данных.

Стандартные параметры, поддерживаемые mysqlcheck

```
--character-sets-dir      --host       --socket
--compress                --password   --user
```

```
--debug                    --pipe        --verbose
--default-character-set    --port        --version
--help                     --silent
```

С MySQL 4 программа `mysqlcheck` стала поддерживать стандартные параметры работы с протоколом SSL.

Специальные параметры `mysqlcheck`

Программа `mysqlcheck` поддерживает параметры, позволяющие ей управлять обработкой таблиц. Ниже приводится список, который в основном фокусируется на совпадениях между этими параметрами и операторами SQL, к которым они относятся.

- `--all-databases, -A` (*boolean*). Проверить все таблицы во всех базах данных.
- `--analyze, -a`. Выполнить анализ таблицы с помощью оператора ANALYZE TABLE. (Например, так можно проанализировать распределение ключевых значений.) Результаты проведенного анализа могут помочь выполнить быстрее поиск с применением индекса и объединений.
- `--all-in-1, -1` (*boolean*). Без этого параметра программа `mysqlcheck` запускает отдельные запросы на все таблицы. Этот параметр позволяет группировать таблицы по базе данных и вызвать все таблицы в каждой базе данных с помощью одного запроса.
- `--auto-repair` (*boolean*). Если проверяемые таблицы имеют проблемы, необходимо запустить вторую фазу для автоматического восстановления после того, как проверочная фаза была завершена.
- `--check, -c`. Запустить оператор CHECK TABLE для проверки на ошибки. Это действие по умолчанию, если не задано явным образом никакого другого действия.
- `--check-only-changed, -C`. Проверить только те таблицы, которые были изменены со времени последней проверки, или те, которые не были закрыты правильно.
- `--databases, -B` (*boolean*). Интерпретировать аргументы как имена баз данных и проверить все таблицы во всех базах данных.
- `--extended, -e` (*boolean*). Выполнить расширенную проверку таблицы. При использовании с параметром `--repair` используется более расширенный, но вместе с этим и более медленный метод, чем тот, который используется для самого параметра `--repair`.
- `--fast, -F` (*boolean*). Проверить только те таблицы, которые не были закрыты правильно.
- `--force, -f` (*boolean*). Продолжать выполнение даже при возникновении ошибок.
- `--medium-check, -m`. Выполнить проверку таблицы, используя более быстрый метод, чем тот, который используется параметром `-extended`, но

при этом менее тщательный. Этого метода проверки в большинстве случаев будет достаточно.

- `--optimize, -o`. Выполнить оптимизацию таблицы с помощью оператора `OPTIMIZE TABLE`.
- `--quick, -q` (*boolean*). При проверке таблиц этот параметр позволяет пропускать связи в строках данных. При одновременном использовании с параметром `--repair` этот параметр восстанавливает только индексный файл, оставляя файл данных нетронутым. В отличие от программы `myisamchk` задание этого параметра дважды не имеет никакого воздействия.
- `--repair, -r`. Выполняет восстановление таблицы с помощью оператора `REPAIR TABLE`. Этот режим восстановления позволяет решить большинство проблем за исключением появления двойных значений в уникальном индексе.
- `--tables`. Перекрывает действие параметра `--databases`.
- `--use-frm` (*boolean*). При использовании с параметром `--repair` для выполнения операции восстановления таблицы для интерпретации файлов данных и перестройки индексного файла анализируется файл `.frm`. Этот параметр можно применять в случае разрушения индексного файла или нарушения его целостности. Этот параметр был добавлен в MySQL 4.0.5.

Соотношение между параметрами программы `mysqlcheck` и операторами SQL, которые в ней используются, описывается с помощью следующих таблиц, в которых отражено соответствие операторов языка SQL и параметров программы `mysqlcheck`.

Параметры проверки таблиц (только таблицы типа MyISAM и InnoDB).

Параметр	Соответствующий оператор
`--check`	`CHECK TABLE` *`tbl_list`*
`--check-only-changed`	`CHECK TABLE` *`tbl_list`* `CHANGED`
`--extended`	`CHECK TABLE` *`tbl_list`* `EXTENDED`
`--fast`	`CHECK TABLE` *`tbl_list`* `FAST`
`--medium-check`	`CHECK TABLE` *`tbl_list`* `MEDIUM`
`--quick`	`CHECK TABLE` *`tbl_list`* `QUICK`

Параметры анализа таблиц (только таблицы типа MyISAM и BDB).

Параметр	Соответствующий оператор
`--analyze`	`ANALYZE TABLE` *`tbl_list`*

Параметры восстановления таблиц (только таблицы типа MyISAM).

Параметр	Соответствующий оператор
`--repair`	`REPAIR TABLE` *`tbl_list`*
`--repair -quick`	`REPAIR TABLE` *`tbl_list`* `QUICK`
`--repair -extended`	`REPAIR TABLE` *`tbl_list`* `EXTENDED`
`--repair -use-frm`	`REPAIR TABLE` *`tbl_list`* `USE_FRM`

Параметры оптимизации таблиц (только таблицы типа MyISAM).

Параметр	Соответствующий оператор
`--optimize`	`OPTIMIZE TABLE` *`tbl_list`*

Программа `mysql_config`

Программа `mysql_config` представляет собой вспомогательное средство написания C-программ, работающих с базами данных СУБД MySQL. Она может использоваться для получения соответствующих флагов, необходимых для компиляции исходных файлов на языке программирования или подключения к библиотекам MySQL. Эта программа была добавлена в MySQL 3.23.21.

Использование

```
mysql_config [options]
```

Специальные параметры `mysql_config`

- `--cflags`. Отобразить флаги включаемых каталогов, необходимых для файлов заголовков MySQL.
- `--embedded`, `--embedded-libs`. Эти параметры являются синонимами параметра `--libmysqld-libs`.
- `--libs`. Отобразить флаги библиотек, необходимые для подключения клиентской библиотеки.
- `--socket`. Отобразить путь к стандартному файлу разъема.
- `--port`. Отобразить стандартный TCP/IP-порт MySQL.
- `--version`. Отобразить строку с версией СУБД MySQL.
- `--libmysqld-libs`. Отобразить строку с флагами библиотек, необходимыми для подключения библиотеки встроенного сервера `libmysqld`.

Программа `mysqld`

Программа `mysqld` является сервером MySQL, который обеспечивает клиентским программам доступ к базам данных. Следовательно, эта программа должна постоянно работать, иначе клиенты не смогут получить доступ к управляемым сервером базам данных. Сразу после запуска программа `mysqld` открывает сетевые порты и ожидает запросы на установление соединений от клиентов. Программа `mysqld` является многопотоковой и обрабатывает каждое клиентское соединение в отдельном потоке, что обеспечивает параллельность работы различных клиентов. Запросы на запись в базу данных выполняются одновременно. Когда сервер начинает исполнять такой запрос, он не приступит ни к одному другому запросу, связанному с этой же базой данных, пока не завершит текущий. Именно это не позволяет двум клиентам одновременно изменить одну и ту же строку таблицы.

Использование

Обычная последовательность запуска сервера состоит из имени сервера с указанием любой последовательности допустимых параметров:

```
mysqld [options]
```

В системах Windows NT сервер может устанавливаться для работы в режиме службы.

Например, сервер `mysqld-nt` может запускаться автоматически при запуске системы или удаляться из активных служб следующим образом:

```
C:\> mysqld-nt --install
C:\> mysqld-nt --remove
```

MySql — имя службы, принимаемое по умолчанию. Начиная с MySQL 4.0.2 появилась возможность задавать имя службы:

```
C:\> mysqld-nt --install service_name
C:\> mysqld-nt --remove service_name
```

Это позволяет нескольким серверам одновременно работать под различными именами служб. Если аргумент *service_name* не задан, в качестве имени службы берется `MySql`, при этом сервер во время запуска считывает из конфигурационного файла группу `[mysqld]`. Аргумент *service_name* присваивает это имя службе, и сервер берет во время старта параметры из группы `[service_name]` конфигурационного файла.

С версии MySQL 4.0.3 появилась возможность задания после имени службы параметра `--defaults-file`. Он позволяет указывать имя дополнительного параметрического файла для сервера, который будет считываться во время запуска:

```
C:\> mysqld-nt --install service_name --defaults-file=file_name
```

В этом случае аргумент *service_name* обязателен.

Замечания, сделанные в адрес параметра `-install`, справедливы и для параметра `--install-manual`.

Стандартные параметры, поддерживаемые программой `mysqld`

```
--character-sets-dir     --help              --socket
--debug                        --port                --user
--default-character-set  --set-variable  --version
```

Параметры `--character-sets-dir` и `--default-character-set` были добавлены в MySQL 3.23.14. Начиная с MySQL 4 `mysqld` также поддерживает стандартные параметры SSL.

Обратите внимание, что несмотря на поддержку параметра `-socket`, не поддерживается его короткая форма `-S`.

Параметр `--user` (если таковой задан) определяет имя пользователя, учетная запись которого будет применяться для запуска сервера. При подобном запуске сервер просматривает идентификатор пользователя и группы учетной записи в файле паролей, а затем соответствующим образом их изменяет. В итоге сервер работает с полномочиями заданного пользователя, а не пользователя `root`.

(Для достижения наибольшей эффективности работы необходимо запустить сервер с помощью учетной записи пользователя `root`.) Начиная с MySQL 4.0.2 параметр `--user` может быть представлен числовым идентификатором пользователя.

Специальные параметры `mysqld`

В первом разделе перечислены общие параметры. Потом следуют разделы с параметрами, специфическими только для ОС Windows, определенных дескрипторов таблиц и процедуры репликации.

- `--ansi`, `-a`. Указание серверу руководствоваться стандартом ANSI для синтаксиса SQL, а не синтаксисом MySQL. Этот параметр предназначен для совместимости сервера со стандартами ANSI.

  Этот параметр был добавлен в MySQL 3.23.6. Он эквивалентен использованию параметра `--sql-mode` с флагами `REAL_AS_FLOAT`, `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE`, `SERIALIZE` и `ONLY_FULL_GROUP_BY`.

- `--basedir=`*`path_name`*, `-b` *`path_name`*. Определяет путь к каталогу инсталляции MySQL. Если впоследствии в процессе работы сервера задаются относительные пути, то они начинаются с указанного здесь каталога.

- `--big-tables`. Позволяет обрабатывать большие результирующие наборы за счет сохранения всех временных результатов на диске, а не в памяти. Это позволяет избежать появления сообщений о "заполнении таблиц", которые возникают из-за недостатка памяти. Этот параметр необходим только для версий, предшествующих MySQL 3.23.

- `--bind-address=`*`ip_addr`*. Привязать сервер к заданному IP-адресу. Обычно программа `mysqld` привязывается к IP-адресу, по умолчанию установленному для компьютера, на котором запущен сервер. С помощью этого параметра можно выбрать альтернативный адрес, если используемый компьютер имеет сразу несколько адресов.

- `--bootstrap`. Этот параметр используется инсталляционными сценариями при первой инсталляции MySQL. Этот параметр был добавлен в MySQL 3.22.10.

- `--chroot=`*`dir_name`*, `-r` *`dir_name`*. Запускает сервер MySQL, связанный со специальным корневым каталогом. Этим параметром задается путь к этому каталогу. Более детально о работе в среде `chroot()` рассказывается в разделе справочного руководства по UNIX, посвященному `chroot()`. Этот параметр впервые появился в MySQL 3.22.2.

- `--concurrent-insert` (*`boolean`*). Разрешает параллельные вставки в таблицы MyISAM или запрещает их параметром `--skip-concurrent-insert`. Если в таблице MyISAM нет пропусков, механизм параллельных вставок позволяет добавлять новые записи одновременно с выполнением выборки уже существующих строк. Этот параметр был добавлен с логическим параметром в MySQL 4.0.2. В версиях 3.23.25–4.0.1 может использоваться только `--skip-concurrent-insert`.

- `--core-file`. Позволяет до выхода сгенерировать файл дампа памяти при возникновении фатального сбоя. Этот параметр был добавлен в MySQL 3.23.23.

- `--datadir=path_name`, `-h path_name`. Определяет путь к каталогу данных MySQL.
- `--default-table-type=type`. Устанавливает формат хранения таблиц по умолчанию. В качестве аргумента *type* могут выступать значения `ISAM`, `MyISAM`, `HEAP`, `BDB` или `InnoDB`. (Регистр символов при их написании не учитывается.) Если данный параметр не задан, по умолчанию используется формат MyISAM. Этот параметр впервые появился в MySQL 3.23.
- `--delay-key-write=val`. Устанавливает режим задержки записи ключей для файлов типа MyISAM. Значение *val* может принимать значение `ON` (задержка записи ключей осуществляется на потабличной основе в соответствии со значениями `DELAY_KEY_WRITE`, созданными в момент создания таблиц; это значение принимается по умолчанию), `OFF` (никогда не задерживать запись ключей ни для одной таблицы MyISAM) или `ALL` (задерживать запись ключей для всех таблиц MyISAM). Значения `OFF` и `ALL` реализуют политику, не зависящую от объявления таблиц при их создании.

  Общепринятой практикой является запуск подчиненных серверов репликации со значением параметра `delay-key-write=ALL`. Это позволяет повысить производительность для таблиц MyISAM, задерживая запись ключей независимо от того, как таблицы были созданы изначально.

  Этот параметр был добавлен в MySQL 3.23.3 и назван `--delay-key-write-for-all-tables` в 3.23.11. Начиная с MySQL 4.0.3 его опять можно использовать. Для его отмены используется параметр `--skip-delay-key-write`. Это позволяет добиться эффекта, равносильного эффекту от значения ключа `OFF`.
- `--des-key-file=file_name`. Имя файла, в котором хранятся DES-ключи для функций `DES_ENCRYPT()` и `DES_DECRYPT()`. Описание формата этого файла см. в приложении B, "Операторы и функции". Этот параметр был добавлен в MySQL 4.0.1.
- `--enable-locking`. См. описание параметра `--external-locking`. Этот параметр был добавлен в MySQL 3.22.4.
- `--enable-pstack` (*boolean*). Активизирует режим распечатки стека символов при возникновении ошибки. Этот параметр был добавлен в MySQL 4.0.0.
- `--exit-info[=n]`, `-T[n]`. Выводит информацию отладки по завершении работы программы. Если после `-T` задано значение *n*, между ними не должно быть пробела или значение будет интерпретировано неверно. Этот параметр впервые появился в MySQL 3.22.
- `--external-locking` (*boolean*). Активизирует внешнюю блокировку (блокировку файловой системы) для таких ОС, как Linux, где по умолчанию такая блокировка отключена. Этот параметр был добавлен в MySQL 4.0.3. До этого для активизации или отключения внешней блокировки использовались параметры `--enable-locking` или `--skip-locking`.
- `--flush`. Сохраняет все таблицы на диске после каждого восстановления. Это уменьшает риск разрушения таблицы при сбое, но серьезно снижает производительность. Это очень полезно при нестабильной работе системы.

Этот параметр был добавлен в MySQL 3.22.9. Он работает только для таблиц MyISAM и ISAM.

- `--init-file=file_name`. Определяет файл с операторами SQL, выполняемый в процессе запуска. Относительное имя файла интерпретируется с каталогом данных в качестве корневого каталога. Этот файл должен содержать по одному оператору в строке.
- `--language=lang_name`, `-L`. Определяет язык вывода сообщений об ошибках. Обычно в качестве аргумента `lang_name` выступает значение `english` или `german`, но он может быть и полным путем к каталогу, в котором содержатся языковые файлы.
- `--local-infile (boolean)`. Активизирует или отключает LOAD DATA LOCAL. Начиная с MySQL 3.23.49 возможности LOCAL могут присутствовать, будучи отключенными по умолчанию. Для того чтобы активизировать режим LOCAL, на стороне сервера необходимо активизировать параметр `--local-infile`. Этот параметр также может использоваться для отключения режима LOCAL на стороне сервера, если он активизирован по умолчанию. Используйте `--disable-local-infile` начиная с MySQL 4.0.2 и `--local-infile=0` — до этой версии.
- `--log[=file_name]`, `-l [file_name]`. Активизирует регистрацию в файл общего журнала. Этот журнал содержит информацию о клиентских соединениях и запросах. Если путь `file_name` не задан, для записи журнала по умолчанию используется файл *HOSTNAME*`.log`, где *HOSTNAME* — имя компьютера с сервером. Если путь в `file_name` задан как относительный, он определяется начиная с каталога данных. Если после `-l` задано значение `file_name`, между ними не должно быть пробела, в противном случае значение будет интерпретировано неверно.
- `--log-bin[=file_name]`. Активизирует журнал двоичных обновлений. Если значение `file_name` не задано, то имя файла регистрации будет *HOSTNAME*`-bin.nnn` и размещаться он будет в каталоге данных, где *HOSTNAME* — это имя узла сервера, а *nnn* — последовательный номер, на единицу больший используемого для предыдущего файла журнала обновлений. Относительное имя файла интерпретируется с каталогом данных в качестве корневого каталога. Этот параметр был добавлен в MySQL 3.23.14.
- `--log-bin-index=file_name`. Активизирует индексный файл журнала двоичных обновлений. Относительное имя файла интерпретируется с каталогом данных в качестве корневого каталога. Этот параметр был добавлен в MySQL 3.23.15.
- `--log-slow-queries[=file_name]`. Включает регистрацию медленных запросов в указанный файл. Если файл не указан, по умолчанию регистрация производится в файл *HOSTNAME*`-slow.log`, расположенный в каталоге данных, где *HOSTNAME* — имя узла сервера. Относительное имя файла интерпретируется с каталогом данных в качестве корневого каталога. Этот параметр был добавлен в MySQL 3.23.29.
- `--log-isam[=file_name]`. Активизирует регистрацию в файл индексов, которая необходима только для операций отладки таблиц ISAM/MyISAM.

По умолчанию регистрация производится в файл `myisam.log`, расположенный в каталоге данных.

- `--log-long-format`. Записывает дополнительную информацию в журнал обновлений. Этот параметр впервые появился в MySQL 3.22.7.
- `--log-update[=file_name]`. Активизирует регистрацию в файл журнала обновлений. Журнал обновлений содержит текст всех запросов на изменение таблиц баз данных. Если путь *file_name* не задан, для записи журнала по умолчанию используется файл *HOSTNAME*.*nnn*, где *HOSTNAME* — имя компьютера с сервером, а *nnn* — последовательный номер, на единицу больший используемого в предыдущем файле журнала обновлений.

  Если же путь *file_name* задан с именем файла без расширения, сервер добавляет числовое расширение в форме описанной ранее последовательности. Определенное вместе с расширением имя файла используется все время без изменений. Если путь *file_name* задан как относительный, он определяется начиная с каталога данных. Начиная с MySQL 3.23.14 журнал обновлений был заменен журналом двоичных обновлений.
- `--log-warnings` (*boolean*). Записывает некритические диагностические сообщения в журнал. Этот параметр был добавлен в виде параметра `--warnings` в MySQL 3.23.40 и переименован в `--log-warnings` в 4.0.3.
- `--low-priority-updates`. Устанавливает для операций записи меньший приоритет, чем для операций выборки. Этот параметр был добавлен в MySQL 3.23. (В версиях MySQL 3.22.5–3.23 этот параметр назывался `--low-priority-inserts`.)
- `--memlock` (*boolean*). Если необходимо заблокировать сервер в памяти. Этот параметр был добавлен в MySQL 3.23.25. Он действует только в ОС Solaris, только если сервер работает под управлением пользователя `root`.
- `--myisam-recover=level`. Управляет типом автоматической проверки таблиц MyISAM, которые сервер выполняет во время запуска. Значение *level* может быть пустым, что отключает проверку, или списком, разделенным запятыми, который содержит одно или несколько следующих значений: `DEFAULT` (равносильно отсутствию значения), `BACKUP` (создание резервной копии таблицы при ее обновлении), `FORCE` (сделать восстановление таблицы при потере более чем одной строки данных) или `QUICK` (быстрое восстановление). Этот параметр был добавлен в MySQL 3.23.25.
- `--new`, `-n`. Использует новый, возможно, небезопасный шаблон. Эта возможность MySQL является весьма новой и не объявлена официально, поэтому ее следует применять с большой осторожностью.
- `--old-protocol`, `-o`. Использует протокол, который применялся для взаимодействия сервера и клиентов в версиях MySQL, предшествующих 3.21. Этот параметр может потребоваться только для взаимодействия сервера с очень старыми клиентскими программами.
- `--old-passwords`. Начиная с MySQL 4.1 сервер поддерживает новый метод шифрования паролей. Существующие учетные записи, пароли которых зашифрованы с применением старого метода, по-прежнему поддерживаются,

но новые пароли шифруются с помощью нового метода. Используйте этот параметр для того, чтобы получить возможность использования старого метода шифрования паролей при создании новых паролей. (Это может пригодиться при переходе на более старую версию сервера или переносе учетной записи на старый сервер.) Этот параметр был добавлен в MySQL 4.1.

- `--one-thread`. Запускается с одним потоком. Подобный запуск используется для отладки под Linux, которая обычно управляет по крайней мере тремя потоками. Этот параметр впервые появился в MySQL 3.22.2.
- `--pid-file=`*`file_name`*. При запуске сервер `mysqld` записывает идентификатор своего процесса (PID — process ID) в файл, а этот параметр определяет путь к данному PID-файлу. Впоследствии этот файл может использоваться другими процессами для определения номера процесса сервера, как правило, с целью отправки сигнала. Так, например, сценарий `mysql.server` считывает такой PID-файл, если необходимо отправить серверу сигнал о завершении работы. Этот параметр не действует в ОС Windows или при работе встроенного сервера. Если путь *`file_name`* задан как относительный, он определяется начиная с каталога данных.
- `--safe=mode`. Этот параметр аналогичен параметру `--skip-new`, однако отключает еще большее число возможностей. Его можно использовать в тех случаях, когда MySQL работает нестабильно или запуск сложных запросов приводит к выводу некорректных результатов. Если применение этого параметра положительно отражается на работе сервера, упомяните этот факт при описании проблемы в отчете `mysqlbug`.
- `--safe-show-database` (*`boolean`*). Не отображает имена баз данных пользователей, которые не имеют права доступа к ним. Этот параметр был добавлен в MySQL 3.23.30. Однако от его применения отказались в MySQL 4.0.2, когда была добавлена привилегия `SHOW DATABASES` для управления возможностью просмотра имен баз данных.
- `--safe-user-create` (*`boolean`*). Запрещает создание учетных записей пользователями, которые не имеют права доступа к таблицам разрешения. Этот параметр был добавлен в MySQL 3.23.41.
- `--safemalloc-mem-limit=`*`n`*. Моделирование недостатка памяти — это значение устанавливает предел доступной памяти. Этот параметр может быть использован, только если сервер был построен с параметром `--with-debug=full` во время конфигурации. Этот параметр был добавлен в MySQL 3.23.28.

  `--skip-concurrent-insert`. См. описание параметра `--concurrent-insert`.
- `--skip-grant-tables`, `-Sg`(*`boolean`*). Отменяет использование таблиц разрешений для проверки клиентских соединений. В результате любой клиент может получить доступ ко всем данным. Чтобы снова указать серверу использовать таблицы разрешений, запустите команду `mysqladmin flush-privileges` или оператор `FLUSH PRIVILEGES`. Форма `-Sg` этого параметра не используется с версии MySQL 4.0.0.

- `--skip-host-cache`. Отключает кэширование таблиц.
- `--skip-locking`. См. описание параметра `--external-locking`. Этот параметр не используется с версии MySQL 4.0.3.
- `--skip-name-resolve`. Отменяет попытки разрешения имен компьютеров. Если этот параметр задан, в таблицах разрешений компьютеры должны задаваться либо посредством IP-адреса, либо словом `localhost`.
- `--skip-networking`. Запрещает установление TCP/IP-соединений. В результате клиенты смогут подключаться к серверу с локального компьютера, указывая имя `localhost`.
- `--skip-new`. Запрещает использовать новые, возможно, небезопасные шаблоны. См. описание параметра `--new`.
- `--skip-safemalloc`. Не выполняет проверку распределения памяти. Этот параметр может применяться только в том случае, если сервер был построен с использованием параметра `--with-debug=full`. Этот параметр был добавлен в MySQL 3.23.37.
- `--skip-show-database`. Не позволяет пользователям запускать запросы `SHOW DATABASES` или `SHOW TABLES` к базам данных, к которым эти пользователи не имеют прав доступа. Этот параметр впервые появился в MySQL 3.23.
- `--skip-stack-trace`. Не печатает дерево стека при сбоях. Этот параметр был добавлен в MySQL 3.23.38.
- `--skip-symlink`. Отменяет символические ссылки на таблицы. Этот параметр был добавлен в MySQL 3.23.39.
- `--skip-thread-priority`. Как правило, запросы на изменение таблиц обладают большим приоритетом, чем запросы выборки данных. Этот параметр позволяет уровнять приоритет запросов разного вида.
- `--sql-bin-update-same (boolean)`. Объединяет флаги `SQL_LOG_BIN` и `SQL_LOG_UPDATE` таким образом, что установка одного из них (оператором `SET`) устанавливает и другой. Этот параметр был добавлен в MySQL 3.23.16.
- `--sql-mode=flags`. Изменяет некоторые аспекты поведения сервера, что приводит его поведение в соответствие со стандартами ANSI SQL или с другими серверами. Флаги должны представлять собой списки, состоящие из одного или нескольких параметров, разделенных запятыми.
  - `ANSI_QUOTES`. Рассматривать двойные кавычки как символ кавычки для таких идентификаторов, как идентификаторы баз данных, таблиц и имен столбцов, а не как символ выделения строк. (Обратные кавычки по-прежнему разрешено использовать для выделения имен.)
  - `IGNORE_SPACE`. Разрешить использование пробелов между именами функций и открывающими скобками. Это приводит к распознаванию имен функций как зарезервированных слов.
  - `NO_UNSIGNED_SUBTRACTION`. Начиная с MySQL 4 вычитание целых операндов дает беззнаковый результат, если хотя бы один из операндов является беззнаковым. Этот параметр позволяет получать беззнаковые результаты, что равносильно поведению версий MySQL, предшествующих 4.

- ONLY_FULL_GROUP_BY. Обычный синтаксис MySQL разрешает применение операторов SELECT с указанием в списке выводимых столбцов, которые не упоминаются в предложении GROUP BY:

  ```
  SELECT a, b, COUNT(*) FROM t GROUP BY a;
  ```

  Флаг ONLY_FULL_GROUP_BY требует, чтобы все выводимые столбцы были перечислены в предложении GROUP BY:

  ```
  SELECT a, b, COUNT(*) FROM t GROUP BY a, b;
  ```

- PIPES_AS_CONCAT. Рассматривать символ "||" как оператор конкатенации строк, а не как логическое "ИЛИ".
- REAL_AS_FLOAT. Тип столбцов REAL становится синонимом типа FLOAT, а не DOUBLE.
- SERIALIZE. Использовать SERIALIZABLE в качестве уровня изоляции транзакций по умолчанию.

Параметр `--sql-mode` равнозначен параметру `--ansi` в том смысле, что он разрешает поведение в соответствии со стандартом ANSI, но более гибок, так как позволяет устанавливать определенные возможности в отдельности. Заметим также, что поведение NO_UNSIGNED_SUBTRACTION может быть задано только с помощью параметра `--sql-mode`.

Параметр `--sql-mode` был добавлен в MySQL 3.23.41. Флаг NO_UNSIGNED_SUBTRACTION был добавлен в MySQL 4.0.0.

- `--temp-pool` (*boolean*). Позволяет серверу использовать для создания временных файлов ограниченный набор имен, а не создавать уникальное имя для каждого файла. Это позволяет избежать определенных проблем кэширования на ОС Linux. Этот параметр был добавлен в MySQL 3.23.33, а начиная с MySQL 4.0.3 он активизирован по умолчанию.
- `--transaction-isolation=`*level*. Устанавливает уровень изоляции транзакций по умолчанию. Допустимые значения уровней составляют READ-UNCOMMITTED, READ-COMMITTED, REPEATABLE-READ и SERIALIZABLE. Этот параметр был добавлен в MySQL 3.23.26.
- `--tmpdir=`*dir_name*, `-t` *dir_name*. Определяет путь к каталогу, используемому для хранения временных файлов. Этот параметр впервые появился в MySQL 3.22.4. Начиная с MySQL 4.1 значение этого параметра может быть задано списком каталогов, которые могут использоваться по очереди. Под UNIX имена каталогов разделяются двоеточиями; а под Windows – точками с запятой.
- `--warnings`. См. описание параметра `--log-warnings`.

Параметры для Windows

Помимо определенных ранее, существуют также параметры, относящиеся только к серверам, работающим под управлением Windows. Некоторые из них имеют отношение к службам и применимы только в системах Windows NT.

- `--console` (*boolean*). Отображает окно консоли для сообщений об ошибках. Этот параметр впервые появился в MySQL 3.22.4.

- `--enable-named-pipe` (*boolean*). До MySQL 3.23.50 именованные каналы для NT-серверов активизировались по умолчанию (т.е. серверы с суффиксом `-nt` в именах). Начиная с 3.23.50 именованные каналы по умолчанию отключены, а этот параметр может использоваться для включения именованных каналов.

  Заметим, что именованные каналы могут вызывать проблемы во время отключения сервера, и это основная причина, почему настоятельно рекомендуется по возможности избегать их использования.
- `--install`. Инсталлирует сервер `mysqld-nt` в виде службы (относится только к Windows NT).
- `--install-manual`. Инсталлирует сервер как сервис, который не запускается автоматически при запуске ОС Windows. Эту службу должен запускать администратор. Этот параметр был добавлен в MySQL 3.23.44 (относится только к Windows NT).
- `--remove`. Удаляет сервер `mysqld-nt` как службу (относится только к Windows NT).
- `--standalone`. Запускает сервер `mysqld-nt` в виде отдельной программы, а не службы (относится только к Windows NT).
- `--use-symbolic-links` (*boolean*). Поддерживает создание символических ссылок на каталоги баз данных. Этот параметр был добавлен в MySQL 3.23.17. Начиная с MySQL 4.0 поддержка создания символических ссылок запускается по умолчанию, а для ее отключения используется параметр `--skip-symlink`.

Параметры BDB

В этом разделе описываются параметры, предназначенные для дескриптора таблиц BDB.

- `--bdb-home=`*dir_name*. Корневой каталог таблиц BDB. При определении явным образом это значение должно равняться `--datadir`. Этот параметр был добавлен в MySQL 3.23.14.
- `--bdb-lock-detect=`*val*. Установить режим обнаружения/разрешения клинча таблиц BDB. Значение *val* должно быть одним из следующих значений, отражающих, по каким принципам выбирается прерываемая транзакция для ее прерывания при необходимости выхода из тупика: `YOUNGEST` (последняя из начатых транзакций), `OLDEST` (самая старая траназакция), `RANDOM` (случайно выбранная транзакция) или `DEFAULT` (транзакция выбирается в соответствии с политикой, которая сейчас равносильна `RANDOM`). Этот параметр был добавлен в MySQL 3.23.15.
- `--bdb-logdir=`*dir_name*. Путь к каталогу, в который дескриптор таблиц типа BDB записывает регистрационные файлы. Этот параметр был добавлен в 3.23.14.
- `--bdb-no-recover`. Дескриптор таблиц BDB не должен автоматически восстанавливать BDB-файлы во время запуска. Он может пригодиться для

запуска сервера в некоторых случаях, когда он выходит из строя во время запуска из-за сбоя автоматического восстановления Этот параметр был добавлен в MySQL 3.23.30. (Параметр `--bdb-recover` использовался в версиях 3.23.16–3.23.29, но с противоположным действием.)

- `--bdb-no-sync`. Не сбрасывать журналы BDB синхронно. Этот параметр был добавлен в MySQL 3.23.14.
- `--bdb-shared-data`. Запустить дескриптор BDB в многопроцессном режиме. Этот параметр был добавлен в MySQL 3.23.29.
- `--bdb-tempdir=`*`dir_name`*. Путь к каталогу, в котором дескриптор BDB-таблиц записывает временные файлы. Этот параметр был добавлен в MySQL 3.23.14.
- `--skip-bdb`. Отключить обработчик BDB-таблиц. Если BDB-таблицы не используются, этот параметр экономит память. Этот параметр был добавлен в MySQL 3.23.15.

Параметры InnoDB

В этом разделе описываются параметры, предназначенные для дескриптора таблиц InnoDB. Многие параметры, работающие с таблицами InnoDB, были добавлены в MySQL 3.23.29 с именами, которые начинались на `--innobase` и были переименованы в 3.23.37. Теперь они начинаются с `--innodb`.

- `--innodb_data_file_path=`*`filespec_list`*. Спецификация файлов, содержащих компоненты табличного пространства InnoDB. О формате значения параметра см. в инструкциях по настройке таблиц InnoDB главы 11, "Общее администрирование MySQL". Этот параметр был добавлен в MySQL 3.23.37.
- `--innodb_data_home_dir=`*`dir_name`*. Путь к каталогу, в котором расположены компоненты табличного пространства InnoDB. Этот параметр был добавлен в MySQL 3.23.37.
- `--innodb_fast_shutdown` (*`boolean`*). Для ускорения процесса выключения сервера дескриптор InnoDB пропускает некоторые операции, которые он обычно выполняет. Этот параметр был добавлен в MySQL 3.23.44.
- `--innodb_flush_log_at_trx_commit=`*`n`*. Данный параметр имеет значение от 1 (умолчание), которое приводит к обновлению журнала InnoDB после выполнения транзакции. Установка параметра равным 0 приведет к уменьшению количества сбрасываемой на диск информации, которое выполняет InnoDB. Однако это увеличивает вероятность потери нескольких из наиболее недавних транзакций при сбое в работе сервера. Вот перечень возможных значений.

Значение	Описание
0	Производить запись в журнал со скоростью одна запись в секунду и сбрасывать на диск
1	Производить запись в журнал после каждой транзакции и сбрасывать на диск
2	Производить запись в журнал после каждой транзакции и сбрасывать на диск один раз в секунду

Этот параметр был добавлен в MySQL 3.23.37. Значение параметра, равное 2, допустимо задавать начиная с MySQL 3.23.52.

- `--innodb_flush_method=val`. Данный параметр имеет отношение только к ОС UNIX. Он определяет метод, используемый для обновления данных в регистрационные журналы InnoDB. Допустимыми значениями являются `fdatasync` (умолчание) и `O_DSYNC`. Этот параметр был добавлен в MySQL 3.23.39.
- `--innodb_log_group_home_dir=`*`dir_name`*. Путь к каталогу, в котором дескриптор InnoDB будет записывать свои регистрационные файлы. Этот параметр был добавлен в MySQL 3.23.37.
- `--innodb_log_arch_dir=`*`dir_name`*. Этот параметр в настоящее время не используется; был добавлен в MySQL 3.23.37.
- `--innodb_log_archive=`*`n`*. Этот параметр в настоящее время не используется; был добавлен в MySQL 3.23.37.
- `--skip-innodb`. Отключить дескриптор таблиц InnoDB. Если таблицы InnoDB не используются, этот параметр экономит память. Этот параметр был добавлен как `--skip-innobase` в MySQL 3.23.26 и переименован в `--skip-innodb` в 3.23.37.

Параметры репликации

Параметры, которым посвящен этот раздел, имеют непосредственное отношение к возможностям репликации MySQL. Для подчиненного сервера репликации некоторые параметры используются совместно с главным информационным файлом (по умолчанию это файл `master.info` в каталоге данных). Если при запуске сервера файла `master.info` не существует, для установления соединения с главным сервером используются значения параметров `--master-host`, `--master-user`, `--master-password`, `--master-port` и `--master-connect-retry`. Он также сохраняет значения в файле `master.info`. Если файл во время запуска сервера существует, он использует содержимое этого файла и игнорирует параметры. Чтобы значения параметров возымели действие, удалите файл и перезапустите сервер. Имя файла может быть изменено параметром `--master-info-file`.

- `--abort-slave-event-count=`*`n`*. Этот параметр используется сценарием `mysql-test-run` для тестирования репликации. Он был добавлен в MySQL 3.23.28.
- `--binlog-do-db=`*`db_name`*. Журнал будет обновляться на главном сервере репликации только для указанных баз данных. Другие базы данных не будут реплицироваться. Для регистрации обновлений нескольких баз данных повторите этот параметр по одному разу для каждой базы данных. Этот параметр был добавлен в MySQL 3.23.23.
- `--binlog-ignore-db=`*`db_name`*. Не регистрировать обновления для указанной базы данных на главном сервере репликации. Для отмены регистрации обновлений для нескольких баз данных повторите этот параметр по одному разу для каждой базы данных. Этот параметр был добавлен в MySQL 3.23.23.
- `--disconnect-slave-event-count=`*`n`*. Этот параметр используется сценарием `mysql-test-run` для тестирования репликации; был добавлен в MySQL 3.23.28.

- `--init-rpl-role=val`. Отобразить роль в процедуре репликации; значение `val` может быть `master` или `slave`. Этот параметр используется сценарием `mysql-test-run` для тестирования репликации; был добавлен в MySQL 4.0.0.
- `--log-slave-updates` (*boolean*). Этот параметр указывает подчиненному серверу репликации регистрировать обновления, которые он принимает с главного сервера, в свой журнал двоичных обновлений. Это необходимо делать в том случае, если подчиненный сервер является главным сервером для другого сервера (т.е. в случае объединения подчиненных серверов в цепочку). Этот параметр был добавлен в MySQL 3.23.17.
- `--master-connect-retry=n`. Для подчиненного сервера репликации время в секундах между попытками подключиться к главному серверу в том случае, если главный сервер отсутствует. Этот параметр был добавлен в MySQL 3.23.15.
- `--master-host=host_name`. Для подчиненного сервера репликации это узел, на котором работает главный сервер. Значение *host_name* может быть именем или IP-адресом. Этот параметр был добавлен в MySQL 3.23.15.
- `--master-info-file=file_name`. Для подчиненного сервера репликации это имя файла, который хранит информацию о состоянии текущей репликации. Содержимым этого файла являются координаты репликации (имя журнала регистрации двоичного обновления главного сервера и положение), главный узел, имя пользователя, номер порта и интервал попыток соединения. По умолчанию этот файл имеет имя `master.info`, который находится в каталоге данных. Этот параметр был добавлен в MySQL 3.23.15.
- `--master-password=pass_val`. Для подчиненного сервера репликации — это пароль учетной записи, которая используется при подключении к главному серверу. Если пароль отсутствует, по умолчанию принимается пустой пароль. Этот параметр был добавлен в MySQL 3.23.15.
- `--master-port=port_num`. Для подчиненного сервера репликации — это порт TCP/IP, используемый для подключения к главному серверу. Если номер порта не указан, по умолчанию принимается номер порта, с которым был скомпилирован сервер (обычно 3306). Этот параметр был добавлен в MySQL 3.23.15.
- `--master-retry-count=n`. Для подчиненного сервера репликации — это количество попыток подключения к главному серверу. Этот параметр был добавлен в MySQL 3.23.43.
- `--master-ssl` (*boolean*).
  `--master-ssl-cert=file_name`
  `--master-ssl-key=file_name`

  Эти параметры были добавлены в MySQL 4.0.0, но до сих пор не используются. Они зарезервированы для осуществления репликаций по безопасным соединениям. (Этот набор параметров все еще является неполным. Еще должны быть добавлены `--master-ssl-ca` и `--master-ssl-cipher`.)
- `--master-user=user_name`. Имя пользователя, которым пользуется подчиненный сервер репликации при подключении к главному серверу. Если имя пользователя не указано, по умолчанию принимается значение "test". Этот параметр был добавлен в MySQL 3.23.15.

- `--max-binlog-dump-events=n`. Данный параметр используется сценарием `mysql-test-run` для тестирования репликации. Он был добавлен в MySQL 3.23.40.
- `--old-rpl-compat (boolean)`. Задать серверу использование старого формата репликации журнала регистрации двоичного обновления для операторов `LOAD DATA`, которые не сохраняют данные в журналах регистрации. Этот параметр был добавлен в MySQL 4.0.0.
- `--reckless-slave`. Данный параметр используется для отладки репликации; был добавлен в MySQL 4.0.2.
- `--relay-log=file_name`. Задать для подчиненного сервера репликации имя журнала переключений. (В MySQL 4 подчиненный поток В/В сохраняет обновления, считываемые в регистрационном журнале переключений, а поток SQL считывает журнал переключений для запросов и обновляет их.) Имя стандартного журнала переключений — *HOSTNAME*`-relay-bin.nnn`, который размещается в каталоге данных, где *HOSTNAME* — имя компьютера с сервером, а *nnn* — последовательный номер, на единицу больший используемого в предыдущем файле журнала обновлений. Этот параметр был добавлен в MySQL 4.0.2.
- `--relay-log-index=file_name`. Задать подчиненному серверу репликации имя индексного файла журнала переключений. Имя стандартного журнала переключений — *HOSTNAME*`-relay-bin.index`, и он размещается в каталоге данных, где *HOSTNAME* — имя компьютера с сервером. Этот параметр был добавлен в MySQL 4.0.2.
- `--relay-log-info-file=file_name`. Задать подчиненному серверу репликации имя индексного файла журнала переключений. По умолчанию имя стандартного журнала переключений `relay-log.info`, он размещается в каталоге данных. Этот параметр был добавлен в MySQL 4.0.2.
- `--replicate-do-db=db_name`. Указание подчиненному серверу репликации реплицировать указанную базу данных. Для того чтобы задать несколько баз данных, необходимо указать имена всех баз данных. Этот параметр был добавлен в MySQL 3.23.16.
- `--replicate-do-table=db_name.tbl_name`. Указание подчиненному серверу репликации реплицировать указанную таблицу. Для того чтобы задать несколько таблиц, необходимо задать параметр. Этот параметр был добавлен в MySQL 3.23.28.
- `--replicate-ignore-db=db_name`. Указание подчиненному серверу репликации не реплицировать указанную базу данных. Для того чтобы задать несколько баз данных, необходимо задать параметр несколько раз. Этот параметр был добавлен в MySQL 3.23.16.
- `--replicate-ignore-table=tbl_name`. Указание подчиненному серверу репликации не реплицировать указанную таблицу. Для того чтобы задать несколько таблиц, необходимо указать имя таблицы несколько раз. Этот параметр был добавлен в MySQL 3.23.28.
- `--replicate-rewrite-db=master_db->slave_db`. Этот параметр указывает подчиненному серверу репликации рассматривать одну базу данных в качестве другой. Обновления, сделанные в исходной базе данных

master_db, на главном сервере репликации реплицируются как обновления базы данных *slave_db* на подчиненном сервере. Этот параметр был добавлен в MySQL 3.23.26. При задании в командной строке значение этого параметра должно быть заключено в кавычки, чтобы интерпретатор команд не воспринял символ '>' как оператор перенаправления вывода.

- `--replicate-wild-do-table=`*pattern*. Указание серверу репликации реплицировать таблицы, удовлетворяющие заданному шаблону. Для того чтобы задать несколько шаблонов, этот параметр необходимо задать несколько раз. Этот параметр был добавлен в MySQL 3.23.28.
- `--replicate-wild-ignore-table=`*pattern*. Указание серверу репликации не реплицировать таблицы, удовлетворяющие заданному шаблону. Для того чтобы задать несколько шаблонов, этот параметр необходимо задать несколько раз. Этот параметр был добавлен в MySQL 3.23.28.
- `--server-id=`*n*. Значение идентификатора сервера репликации. Это значение должно быть уникальным. Данный параметр был добавлен в MySQL 3.23.26.
- `--show-slave-auth-info` (*boolean*). Отобразить имя пользователя подчиненного сервера и его пароль в выводе оператора `SHOW SLAVE STATUS`. Этот параметр был добавлен в MySQL 4.0.0.
- `--skip-slave-start`. Не запускать подчиненный сервер автоматически. В этом случае он запускается вручную оператором `SLAVE START`. Этот параметр был добавлен в MySQL 3.23.26.
- `--slave-load-tmpdir=`*dir_name*. Путь к каталогу, который используется подчиненным сервером при обработке операторов `LOAD DATA`. Этот параметр был добавлен в MySQL 4.0.0.
- `--slave-skip-errors=`*error_list*. Перечень ошибок, которые подчиненный сервер репликации должен проигнорировать, а не задерживать репликацию при ее отсутствии. Значение `all` означает, что будут проигнорированы все ошибки, что существенно помогает при поведении, которое обеспечивается параметром `--reckless-slave`. В противном случае значение должно представлять собой список, содержащий один или несколько номеров ошибок, разделенных запятыми. Этот параметр был добавлен в MySQL 3.23.47.
- `--sporadic-binlog-dump-fail` (*boolean*). Этот параметр используется сценарием `mysql-test-run` для тестирования репликации. Он был добавлен в MySQL 3.23.40.

Переменные mysqld

Для просмотра используемых по умолчанию значений переменных `mysqld` можно воспользоваться командой

```
% mysqld --help
```

Чтобы узнать, какие значения переменных использует работающий в настоящее время сервер `mysqld`, запустите команду

```
% mysqladmin variables
```

Текущие значения переменных можно также проверить и с помощью запроса `SHOW VARIABLES`. Переменные, которые можно просмотреть с помощью этого

оператора, см. в описании оператора SHOW в приложении Г, "Синтаксис SQL". Любая переменная mysqld может быть установлена во время запуска с использованием указаний, данных в разделе "Установка программных переменных". Кроме того, начиная с MySQL 4.0.3 несколько переменных может быть обновлено динамически (более подробную информацию см. в описании оператора SET в приложении Г, "Синтаксис SQL").

Сценарий mysqld_multi

Сценарий mysqld_multi упрощает запуск нескольких серверов mysqld на одном компьютере. Он позволяет запускать, останавливать серверы или определять их работоспособность.

Использование

```
mysqld_multi [options] command server_list
```

Параметр *command* может принимать значения start, stop или report. Аргумент *server_list* отражает, с какими серверами предстоит работать. Подробно о работе с mysqld_multi см. в главе 11, "Общее администрирование MySQL".

Стандартные параметры, которые поддерживаются программой mysqld_multi

```
--help  --password  --user  --version
```

Программа mysqld_multi передает значения параметров --user и --password программе mysqladmin, если требуется остановить сервер или определить его работоспособность.

Специальные параметры mysqld_multi

- --config-file. Конфигурационный файл, в котором перечисляются параметры серверов, с которыми будет работать сценарий mysqld_multi. Без --config-file на предмет наличия параметров для серверов просматриваются файлы /etc/my.cnf и .my.cnf в корневом каталоге. (mysqld_multi ищет в стандартных конфигурационных файлах свои собственные параметры. Данный парамстр в это поведение корректив не вносит.)
- --example. Отобразить пример конфигурационного файла, в котором продемонстрированы группы конфигурационного файла, используемые со сценарием mysqld_multi.
- --log=*file_name* Имя регистрационного файла, в который mysqld_multi должен протоколировать свои действия. Вывод добавляется в журнал, если таковой уже существует. По умолчанию регистрационным файлом является файл /tmp/mysqld_multi.log. Для того чтобы отключить регистрацию, воспользуйтесь параметром --no-log.
- --mysqladmin=*file_name*. Путь к программе mysqladmin, которая будет использоваться. Это может пригодиться, если mysqld_multi не сможет

самостоятельно найти mysqladmin или если вы захотите использовать другую версию.

- --mysqld. Путь к используемой программе mysqld. Это может пригодиться, если mysqld_multi не сможет самостоятельно найти mysqld или потребуется использовать другую версию. (Здесь можно задать путь к mysqld, mysqld_safe или safe_mysqld.)
- --no-log. Отображать протокол непосредственно на экране, а не записывать его в регистрационный файл. (Этот параметр необходимо задавать для того, чтобы видеть протокол, а не отправлять его в регистрационный файл, так как по умолчанию протоколирование производится в файл.)
- --tcp-ip. По умолчанию mysqld_multi делает попытки подключиться к серверу через файл разъема UNIX. Этот параметр позволяет при попытке соединения воспользоваться протоколом TCP/IP. Он удобен, когда сервер работает, но файл разъема удален, в таком случае сервер будет доступен только через TCP/IP-соединение.

Сценарий mysqld_safe

Сценарий mysqld_safe является сценарием оболочки, запускающим сервер mysqld и управляющий им. Если сервер отключается, сценарий mysqld_safe перезапускает его. До MySQL 4 сценарий mysqld_safe был известен под именем safe_mysqld.

mysqld_safe – это сценарий оболочки, поэтому он недоступен в ОС Windows.

Использование

```
mysqld_safe [options]
```

Специальные параметры mysld_safe

Параметры, которые используются с mysqld, могут также использоваться с mysqld_safe, который просто передает эти параметры mysqld. Кроме того, сценарий mysqld_safe принимает и свои собственные параметры.

- --core-file-size=*n*. Ограничение размера файлов дампа *n* байтами. Этот параметр был добавлен в MySQL 3.23.28.
- --err-log=*file_name*. Использовать этот файл для протоколирования ошибок. Относительные имена интерпретируются относительно каталога, из которого был запущен сценарий mysqld_safe. Имя стандартного журнала ошибок – *HOSTNAME*.err, он размещается в каталоге данных, где *HOSTNAME* – имя компьютера с сервером. Данный параметр был добавлен в MySQL 3.23.22.
- --ledir=*dir_name*. Искать сервер в этом каталоге (место, где расположен каталог libexec). Этот параметр был добавлен в MySQL 3.23.22.
- --open-files-limit=*n*. Установить количество имеющихся открытых дескрипторов файлов равным *n*. Этот параметр был добавлен в MySQL 3.23.41.

- `--mysqld=file_name`. Воспользоваться значением `file_name` как путем к программе `mysqld`. Этот параметр был добавлен в MySQL 3.23.30.
- `--timezone=tzspec`. Установить временной пояс `tzspec`. Может понадобиться в том случае, если сервер по какой-либо причине не может определить временной пояс автоматически. Этот параметр был добавлен в MySQL 3.23.28.

Подсказки по модернизации

При обновлении версии MySQL 3.x до MySQL 4.x, где сценарий `mysqld_safe` фигурировал под именем `safe_mysqld`, необходимо обратить внимание на следующие моменты.

- Любой сценарий запуска, запускающий сценарий `safe_mysqld`, должен быть изменен с тем, чтобы запускать `mysqld_safe`. Одним из таких примеров является сценарий `mysql.server`. Однако есть новые версии 4.x, которые автоматически обновляют все ссылки на этот сценарий. При использовании своих собственных сценариев запуска выполните все необходимые настройки.
- Необходимо переименовать группы `[safe_mysqld]`, имеющиеся в конфигурационном файле, в `[mysqld_safe]`.

Программа mysqldump

Программа `mysqldump` записывает содержимое таблиц баз данных в текстовые файлы. Впоследствии такие файлы можно использовать в самых разных целях, например, для резервирования баз данных, переноса их на другой сервер или создания тестовой базы данных на основании содержимого уже существующей базы.

По умолчанию созданный текстовый файл состоит из оператора `CREATE TABLE`, который воссоздает структуру таблицы, и набора операторов `INSERT`, восстанавливающих ее содержимое. При задании параметра `--tab` содержимое таблицы записывается в файл данных в "сыром" формате, а SQL-оператор создания таблицы записывается в отдельный файл.

Использование

`mysqldump` может работать в одном из трех режимов:

```
mysqldump [options] db_name [tbl_name]
mysqldump [options] --databases db_name ...
mysqldump [options] --all-databases
```

В первом случае записываются заданные таблицы указанной базы данных. Если после определения базы данных `db_name` не приводится ни одного имени таблицы, в файл записывается содержимое всех таблиц указанной базы данных. Во втором случае все аргументы берутся как имена баз данных, и `mysqldump` сохраняет все таблицы одной базы данных в один файл. В третьем случае `mysqldump` сохраняет все таблицы из всех баз данных.

Наиболее часто программа mysqldump используется следующим образом:

```
% mysqldump --opt db_name > backip_file
```

Заметим, что этот архивный файл должен импортироваться в MySQL программой mysql, а не mysqlimport:

```
% mysql db_name < backup_file
```

Стандартные параметры, поддерживаемые программой mysqldump

```
--character-sets-dir      --host        --socket
--compress                --password    --user
--debug                   --pipe        --verbose
--default-character-set   --port        --version
--help                    --set-variable
```

Параметры --character-sets-dir и --default-character-set были добавлены в MySQL 3.23.15. Начиная с MySQL 4 mysqldump также поддерживает стандартные параметры SSL.

Специальные параметры mysqldump

Описанные ниже параметры управляют функционированием программы mysqldump. В следующем разделе "Параметры форматирования данных" будут представлены параметры, которые вместе с параметром --tab могут использоваться для представления формата файлов данных.

- add-drop-table (*boolean*). Добавляет оператор DROP TABLE IF EXISTS перед каждым оператором CREATE TABLE. Этот параметр был добавлен в MySQL 3.22.4.
- --add-locks. Добавляет операторы LOCK TABLE и UNLOCK TABLE до и после набора операторов INSERT, загружающих данные в каждую таблицу. Этот параметр был добавлен в MySQL 3.22.3.
- --all, -a (*boolean*). Добавляет дополнительную информацию в операторы CREATE TABLE, которые генерируются программой. Это может быть тип таблицы, начальное значение для AUTO_INCREMENT и т.д. Это информация, которую можно задать в части table_options оператора CREATE TABLE. (См. приложение Г, "Синтаксис SQL".) Этот параметр был добавлен в MySQL 3.22.23.
- --all-databases, -A (boolean). Сохраняет все таблицы во всех базах данных. Этот параметр был добавлен в MySQL 3.23.12.
- --allow-keywords (*boolean*). Позволяет создавать имена столбцов, которые являются ключевыми словами.
- --complete-insert, -c(*boolean*). Использует операторы INSERT для указания имен всех вставляемых столбцов.

- `--databases, -B` (*boolean*). Интерпретирует все аргументы как имена баз данных и сохраняет все таблицы в каждой базе данных. Этот параметр был добавлен в MySQL 3.23.12.
- `--delayed-insert`. Записывает операторы `INSERT DELAYED`. Этот параметр был добавлен в MySQL 3.23.15.
- `--disable-keys, -K` (*boolean*). Добавляет операторы `ALTER TABLE ... DISABLE KEYS` и `ALTER TABLE ... ENABLE KEYS` в вывод для отключения обновления ключей во время обработки операторов `INSERT`. Это ускоряет создание индексов для таблиц MyISAM. Этот параметр был добавлен в MySQL 3.23.48.
- `--extended-insert, -e` (boolean). Записывает операторы `INSERT`, вставляющие несколько строк. Этот параметр был добавлен в MySQL 3.23.48.
- `--first-slave, -x` (*boolean*). Блокирует все таблицы во всех базах данных с помощью оператора `FLUSH TABLES WITH READ LOCK`. (Однако этот тип блокировки неэффективен для получения надежной блокировки таблиц InnoDB; и вместо этого следует использовать параметр `--single-transaction`.) Этот параметр был добавлен в MySQL 3.23.22.
- `--flush-logs, -F`. Обновляет регистрационные файлы сервера перед запуском процесса резервирования.
- `--force, -f` (*boolean*). Продолжает работу даже при возникновении ошибок.
- `--lock-tables, -l`. Устанавливает блокировку на все таблицы перед их резервированием. Нельзя использовать с параметром `--single-transaction`.
- `--master-data`. Помогает получить резервную копию, которая может быть использована подчиненным сервером. При этом в конце данных записываются расположение имени файла главного сервера и позиция. Применение параметра `--master-data` автоматически активизирует `--first-slave`. Этот параметр был добавлен в MySQL 3.23.48.
- `--no-autocommit` (*boolean*). Включает операторы `INSERT` для каждой таблицы в пределы одной транзакции. Полученный результат может быть загружен эффективнее, чем при выполнении каждого оператора в отдельности в режиме `auto-commit`. Этот параметр был добавлен в MySQL 3.23.48.
- `--no-create-db, -n` (*boolean*). Не записывает операторы `CREATE DATABASE`. (Обычно они добавляются в вывод автоматически при использовании параметров `--databases` или `--all-databases`.) Этот параметр был добавлен в MySQL 3.23.12.
- `--no-create-info, -t`. Не записывает операторы `CREATE TABLE`. Этот параметр может пригодиться при резервном копировании только табличных данных.
- `--no-data, -d` (*boolean*). Не записывает данные таблиц. Это может пригодиться при резервном копировании только операторов `CREATE TABLE`.
- `--opt`. Оптимизирует скорость резервирования таблицы и создает новый файл таким образом, чтобы ускорить процесс восстановления. Этот пара-

метр включает параметры `--add-drop-table`, `--add-locks`, `--all`, `--disable-keys`, `--extended-insert`, `--quick` и `--lock-tables`.

- `--quick`, `-q`. По умолчанию программа `mysqldump` считывает все содержимое таблицы в память, а затем уже из памяти записывает его в файл. Этот параметр заставляет программу записывать каждую строку на диск сразу после получения с сервера.

  При использовании этого параметра ни в коем случае нельзя приостанавливать работу программы `mysqldump`. Ведь сервер будет ожидать возобновления ее работы, что не замедлит сказаться на работе других клиентов.

- `--quote-names`, `-Q` (*boolean*). Выделяет имена таблиц и столбцов символами обратных кавычек (``). Такая возможность может потребоваться, когда в именах присутствуют зарезервированные слова или специальные символы. Этот параметр был добавлен в MySQL 3.23.6.

- `--result-file=`*file_name*, `-r` *file_name*. Выводит результаты в указанный файл. Этот параметр предназначался для ОС Windows, где он предотвращает преобразование символов перевода строки в пары "возврат каретки/перевод строки". Этот параметр был добавлен в MySQL 3.23.28.

- `--single-transaction` (*boolean*). Позволяет осуществить целостное резервное копирование для таблиц типа InnoDB. Идея заключается в том, чтобы все таблицы резервировались при одной транзакции. Это дает эффект блокирования всех записей одновременно. Данный параметр был добавлен в MySQL 4.0.2. Он не может использоваться одновременно с параметром `--lock-tables`.

- `--tab=`*dump_dir*, `-T` *dump_dir*. Позволяет программе `mysqldump` для каждой таблицы создать две таблицы, размещая их в каталоге *dump_dir*. Указанный каталог должен уже существовать. Для каждой таблицы *tbl_name* создаются файл *dump_dir/table_name.txt*, в который заносятся данные таблиц, и *dump_dir/table_name.sql*, в который записываются операторы `CREATE TABLE`. Для задания этого параметра пользователь должен обладать привилегией `FILE`.

  Значения столбцов в файле данных по умолчанию разделяются символами табуляции, а строки завершаются символом перехода на новую строку. Используя описанные в разделе "Параметры форматирования данных", этот формат можно изменить. Эффект от параметра `--tab` может обескуражить, если вы не понимаете принципы его работы.

  - Одни файлы могут записаться на диск сервера, а другие — на диск клиента. Это объясняется тем, что файлы с расширением `SQL` сохраняются в каталоге *dump_dir* на компьютере сервера, а файлы с расширением `TXT` — на компьютере клиента. Если это различные машины, результирующие файлы будут созданы на разных машинах. Для того чтобы избежать любой неопределенности относительно того места, куда записываются эти файлы, необходимо при использовании данного параметра запускать программу `mysqldump` на узле сервера.
  - Файлы `*.txt` будут принадлежать учетной записи, которая используется для запуска сервера, а `*.sql` будут принадлежать вам. Это следствие того

факта, что сервер сам записывает файлы *.txt, в то время как операторы CREATE TABLE отсылаются сервером на mysqldump, который записывает файлы *.sql.

- --tables. Перекрывает действие параметра --databases. Этот параметр был добавлен в MySQL 3.23.12.
- --where=*where_clause*, -W *where_clause*. При задании этого параметра извлекаются и записываются в файлы данных только записи, определенные предложением *where_clause*. Это предложение следует заключить в кавычки, чтобы оболочка не интерпретировала его как аргументы строки с несколькими командами. Данный параметр впервые появился в MySQL 3.22.7.
- --xml, -X. Генерирует вывод в формате XML. Этот параметр можно использовать начиная с MySQL 3.23.51 (он был добавлен в 3.23.48, но формат выводимых им данных не был хорошо проработан).

Параметры форматирования данных

Определив параметр --tab или -T с целью создания отдельного файла данных для каждой таблицы, можно задать дополнительные параметры формата. Значения этих параметров, возможно, придется заключить в кавычки. Описанные далее параметры аналогичны параметрам формата данных, используемым в операторе LOAD DATA. Подробнее о них см. в приложении Г, "Синтаксис SQL".

- --fields-enclosed-by=*char*. Определяет, что значения столбцов необходимо заключить в указанный символ — это, как правило, кавычки. По умолчанию значения столбцов не заключаются в символы. Этот параметр несовместим с использованием параметра --fields-optionally-enclosed-by.
- --fields-escaped-by=*char*. Определяет управляющий символ для специальных символов. По умолчанию такой символ отсутствует.
- --fields-optionally-enclosed-by=*char*. Определяет, что значения столбцов могут заключаться в заданный символ, обычно это кавычки. Такой символ применяется для нечисловых столбцов. По умолчанию значения столбцов не заключаются в кавычки. Этот параметр несовместим с использованием параметра --fields-enclosed-by.
- --fields-terminated-by=*char*. Определяет символ разделения значений столбцов в файле данных. По умолчанию для разделения значений применяется символ табуляции.
- --lines-terminated-by=*str*. Определяет строку (которая может состоять из одного или нескольких символов), записываемую в конце выводимых строк. По умолчанию в качестве такой строки выступает символ перехода на новую строку. Этот параметр был добавлен в MySQL 3.22.4.

Переменные mysqldump

С помощью параметра --set-variables можно установить значения следующих переменных mysqldump.

- max_allowed_packet. Максимальный размер буфера, используемого для обмена данными между сервером и клиентом.
- net_buffer_length. Исходный размер буфера, используемого для обмена данными между сервером и клиентом. Впоследствии этот буфер можно увеличить до размера max_allowed_packet байт.

Сценарий mysqlhotcopy

Программа mysqlhotcopy выполняет эффективное резервное копирование баз данных и таблиц. Она работает только с таблицами типа MyISAM и ISAM. Программа mysqlhotcopy является сценарием, написанным на языке Perl, требующим установленной поддержки DBI-интерфейса. Сейчас он на ОС Windows не работает.

Программа mysqlhotcopy соединяет с сервером на локальном узле и отсылает операторы обновления и блокировки на сервер для каждой копируемой таблицы и копирует табличные файлы в другое место. Это гарантирует, что во время копирования таблицы обновляться не будут. (Существенным является еще тот момент, что, для того, чтобы передать серверу, что указанные таблицы не надо трогать, пока вы непосредственно работаете с табличными файлами, программа mysqlhotcopy реализует протокол, описанный в главе 13, "Резервное копирование и восстановление баз данных".)

Программа mysqlhotcopy была добавлена в MySQL 3.23.11.

Использование

Эта программа запускается различными способами. Общий синтаксис запуска следующий:

```
mysqlhotcopy [options] db_name[./regex/] [new_db_name | dir_name]
```

Например, для того, чтобы скопировать базу данных *db_name* db_name_copy в каталоге данных, необходимо задать команду

```
% mysqlhotcopy [options] db_name_copy
```

А для того, чтобы скопировать базу данных *db_name* db_name в каталоге /tmp, необходимо сделать следующее:

```
% mysqlhotcopy [options] db_name /tmp
```

Другие примеры приведены в документации, которую можно получить с помощью команды

```
% perldoc mysqlhotcopy
```

Стандартные параметры, поддерживаемые программой mysqlhotcopy

```
--debug  --host       --port   --user
--help   --password   --socket
```

Параметр --host предназначен только для обозначения имени локального узла. Обычно сценарий mysqlhotcopy делает попытку подключиться к локаль-

ному серверу через файл разъема ОС UNIX. Если вместо этого задать реальное имя сервера через параметр `--host`, он подключится через TCP/IP-соединение. В этом случае может использоваться параметр `--port` для определения номера порта, отличного от значения номера порта, заданного по умолчанию. Параметр `--host` был добавлен в MySQL 3.23.52.

Специальные параметры `mysqlhotcopy`

- `--allowold`. Если каталог, являющийся приемником для операции копирования, уже существует, переименуйте его с использованием суффикса `_old`. Если результатом копирования будет сбой, переименованное название каталога будет восстановлено с оригинальным именем. Если процедура копирования будет успешной, переименованный каталог будет удален при условии отсутствия параметра `--keepold`.
- `--checkpoint=`*`db_name.tbl_name`*. Записать запись с контрольной точкой в таблицу, которая должна быть предварительно создана со следующей структурой:

```
CREATE TABLE tbl_name
(
    time_stamp TIMESTAMP NOT NULL,
    src VARCHAR(32),
    dest VARCHAR(60),
    msg VARCHAR(255)
);
```

  где `src` и `dest` являются именами исходной и полученной баз данных, а поле `msg` отображает успех или неудачу операции копирования.
- `--dryrun`, `-n`. Режим "No execution". `mysqlhotcopy` сообщает о действиях, которые ему необходимо предпринять для выполнения команды, в действительности не выполняя их. Это очень полезно при проверке функциональных возможностей программы `mysqlhotcopy`.
- `--flushlog`. Обновить журналы после того, как таблицы были заблокированы, и перед тем, как их скопировать. Это будет иметь эффект закрытия их во время копирования.
- `--keepold`. Если предыдущий целевой каталог существует, переименовать его с суффиксом `_old` до начала процедуры копирования. Этот параметр предполагает использование параметра `--allowold`.
- `--method=`*`copy_method`*. Используется для копирования файлов. При значении `cp` используется программа `cp`. В качестве эксперимента также имеется поддержка метода `scp`. В этом случае значение параметра `copy_method` должно представлять собой полный синтаксис используемой команды `scp`, и каталог-приемник должен уже существовать. Использование метода `scp` может сопровождаться тем, что таблицы будут заблокированы на значительно более длительный период времени, чем при локальном копировании. Это происходит вследствие того, что для копирования по сети требуется значительно больше времени.
- `--noindices`. Не копировать индексный файл. (Если впоследствии резервные копии потребуются для восстановления таблиц, вы сможете вос-

становить индексы командой myisamchk --recover для таблиц типа MyISAM или isamchk -recover – для таблиц типа ISAM.)

- --quiet, -q. Отключить весь вывод за исключением сообщений об ошибках.
- --record_log_pos=*db_name.tbl_name*. Перед копированием таблиц запустить операторы SHOW MASTER STATUS и SHOW SLAVE STATUS и записать полученные результаты в заданную таблицу, которая должна быть предварительно создана со следующей структурой:

```
CREATE TABLE tbl_name
(
    host VARCHAR(60) NOT NULL,
    time_stamp TIMESTAMP NOT NULL,
    log_file VARCHAR(32) NULL,
    log_pos INT NULL,
    master_host VARCHAR(60) NULL,
    master_log_file VARCHAR(32) NULL,
    master_log_pos INT NULL,
    PRIMARY KEY (host)
);
```

  Результаты работы оператора SHOW MASTER STATUS записываются в столбцах log_file и log_pos. Эта информация представляет собой координаты репликации для двоичного журнала. Подчиненный сервер должен начинать работу с этих координат в случае его инициализации из резервных файлов как подчиненного сервера, если узлом резервного копирования является главный сервер репликации. Результаты, полученные от SHOW SLAVE STATUS, записываются в столбцах master_host, master_log_file и master_log_pos; они могут использоваться, если узел резервного копирования является подчиненным сервером репликации и вам необходимо инициализировать другой подчиненный сервер для данного главного сервера на основании данных из резервных файлов.
- --regexp=*pattern*. Скопировать базы данных, имена которых совпадают с заданным регулярным выражением. Последним аргументом команды должен быть каталог копирования базы данных.
- --resetmaster. Обновить журнал двоичного обновления, задав оператор RESET MASTER после того, как все таблицы будут заблокированы, но перед тем, как они будут скопированы.
- --resetslave. Обновить информацию в файле master.info, задав оператор RESET SLAVE после того, как все таблицы будут заблокированы, но перед тем, как они будут скопированы.
- --suffix=*str*. Используется при копировании баз данных в каталог баз данных. Каждое новое имя каталога базы данных будет получено путем добавления к старому имени указанного суффикса.
- --tmpdir=*dir_name*. Путь к каталогу, в котором создаются временные файлы. Умолчание заключается в использовании каталога, указанного в переменной среды TMPDIR или /tmp, в том случае, если эта переменная не установлена.

Программа mysqlimport

Программа mysqlimport применяется для считывания содержимого текстовых файлов и загрузки его в существующие таблицы. Она функционирует как интерфейс командной строки для SQL-оператора LOAD DATA и является весьма эффективным способом добавления строк в таблицы. Она *не* позволяет считывать файлы дампа, созданные в формате SQL. Для чтения таких файлов необходимо пользоваться программой mysql.

Использование

```
mysqlimport [options] db_name file_name ...
```

Аргумент *db_name* определяет базу данных, в таблицы которой будут добавлены значения. Собственно таблицы определяются именами текстовых файлов. В имени каждого файла удаляется расширение, а основа используется в качестве названия таблицы, в которую загружаются данные файла. Например, программа mysqlimport загрузит содержимое файла president.txt в таблицу president.

Стандартные параметры, поддерживаемые программой mysqlimport

```
--character-sets-dir      --host       --socket
--compress                --password   --user
--debug                   --pipe       --verbose
--default-character-set   --port       --version
--help                    --silent
```

Параметры --character-sets-dir и --default-character-set были добавлены в MySQL 3.23.21 и 3.23.41 соответственно. С MySQL 4 mysqldump также поддерживает стандартные параметры SSL.

Специальные параметры mysqlimport

Описанные ниже параметры определяют, как программа mysqlimport обрабатывает файлы с данными. Ниже, в разделе "Параметры форматирования данных", описываются параметры, определяющие формат представления данных в текстовых файлах.

- --columns=*col_list*. Перечень столбцов таблицы, к которым относятся столбцы в файле данных. Значения вводимых строк будут загружены в указанные столбцы, а столбцам, имя которых не указано, будет присвоено их значение по умолчанию. Значение *col_list* представляет собой список одного или нескольких столбцов, разделенных запятыми. Этот параметр был добавлен в MySQL 3.23.17.
- --delete, -d (*boolean*). Очищает каждую таблицу перед загрузкой в нее данных.
- --force, -f (*boolean*). Продолжает загрузку строк даже при появлении ошибки.

- `--ignore, -i`. Если вставляемая строка содержит уже существующее значение уникального ключа, этот параметр оставляет существующую строку и отбрасывает добавляемую. Параметры `--ignore` и `--replace` являются взаимоисключающими.
- `--ignore-lines=n`. Проигнорировать первые *n* строк файла данных. Этот параметр может использоваться, например, для пропуска строки с наименованиями столбцов. Этот параметр был добавлен в MySQL 4.0.2.
- `--local, -L (boolean)`. Программа `mysqlimport` по умолчанию разрешает серверу считать файл данных. Это означает, что файл данных должен размещаться на компьютере сервера, а работающий с программой пользователь должен обладать привилегией `FILE`. Параметр `--local` указывает программе `mysqlimport` считать файл на локальном компьютере и отправить его на сервер. Это замедляет работу, но позволяет запускать программу `mysqlimport` на компьютере клиента или на компьютере сервера, не обладая при этом привилегией `FILE`.

  Параметр `--local` впервые появился в MySQL 3.22.15. С MySQL 3.23.49 он будет неэффективным, если сервер был настроен с запрещением использования `LOAD DATA LOCAL`.
- `--lock-tables, -l (boolean)`. Устанавливает блокировку на все таблицы перед загрузкой в них данных.
- `--low-priority (boolean)`. Использует модификатор планирования `LOW_PRIORITY` при загрузке данных в таблицу. Этот параметр был добавлен в MySQL 3.22.27.
- `--replace, -r (boolean)`. Если добавляемая в таблицу строка содержит дублирующее значение уникального ключа, этот параметр заменяет строку таблицы добавляемой строкой. Параметры `--ignore` и `--replace` являются взаимоисключающими.

Параметры форматирования данных

Программа `mysqlimport` по умолчанию предполагает, что значения файла данных разделены символами табуляции, а строки завершаются символами перехода на новую строку. Ожидаемый формат можно изменять с помощью описанных ниже параметров. Значения этих параметров, возможно, придется заключить в кавычки. Описанные далее параметры аналогичны параметрам формата данных, используемым в операторе `LOAD DATA`. Более детальную информацию см. в приложении Г, "Синтаксис SQL".

- `--fields-enclosed-by=char`. Определяет, что значения столбцов необходимо заключить в заданный символ, как правило, в кавычки. По умолчанию значения столбцов не заключаются в символы. Этот параметр несовместим с использованием параметра `--fields-optionally-enclosed-by`.
- `--fields-escaped-by=char`. Определяет управляющий символ для специальных символов. По умолчанию такой символ отсутствует.
- `--fields-optionally-enclosed-by=char`. Определяет, что значения столбцов могут заключаться в заданный символ, обычно это кавычки. Такой

символ применяется для нечисловых столбцов. По умолчанию значения столбцов не заключаются в кавычки. Этот параметр несовместим с использованием параметра `--fields-enclosed-by`.

- `--fields-terminated-by=`*`char`*. Определяет символ разделения значений столбцов в файле данных. По умолчанию для разделения значений применяется символ табуляции.
- `--lines-terminated-by=`*`str`*. Определяет строку (которая может состоять из одного или нескольких символов), записываемую в конце выводимых строк. По умолчанию в качестве такой строки выступает символ перехода на новую строку. Этот параметр был добавлен в MySQL 3.22.4.

Сценарий `mysql_install_db`

Сценарий `mysql_install_db` предназначен для создания на сервере каталога данных, инициализации базы данных `mysql`, содержащей таблицы разрешения, и создания пустой базы данных `test`. Сценарий `mysql_install_db` заполняет таблицы разрешения первоначальными учетными записями для пользователя `root` и анонимного пользователя. Подробнее об этом см. в главе 11, "Общее администрирование MySQL". Сценарий `mysql_install_db` представляет собой сценарий оболочки, который отсутствует в ОС Windows. Однако здесь он и не нужен, так как дистрибутив для ОС Windows уже содержит предварительно инициализированные базы данных `mysql` и `test`.

Использование

```
mysql_install_db [options]
```

Специальные параметры `mysql_install_db`

Номера версий, указанные в этом разделе, показывают, с какого момента можно начать использовать параметр в командной строке, но многие из них можно задавать гораздо раньше, указывая их в группе `[mysqld]` глобального конфигурационного файла. С MySQL 3.23.39 этот сценарий получил возможность считывать свою собственную параметрическую группу `[mysql_install_db]`, которая является более полезной для таких групп, как `--ldata` и `--force`, являющихся понятными только для `mysql_install_db`.

- `--basedir=`*`dir_name`*. Использовать этот каталог как основной каталог MySQL.
- `--datadir=`*`dir_name`*, `--ldata=`*`dir_name`*. Использовать этот каталог как основной каталог MySQL. Параметр `--datadir` был добавлен в MySQL 3.23.17, а `-ldata` — в MySQL 3.23.24.
- `--force`. Запустить, даже если текущий компьютер не может быть обнаружен. Вместо имени узла при создании таблиц разрешения будет использоваться IP-адрес узла, что означает, что при всех соединениях, кроме подключения к узлу `localhost`, необходимо будет пользоваться исключительно IP-адресами. Этот параметр был добавлен в MySQL 3.22.17.

- `--user=user_name`. Работа сервера под управлением указанного пользователя. Этот параметр гарантирует то, что владельцем любого каталога и файла, созданных сервером под управлением пользователя `root` в ОС UNIX, будет именно этот пользователь. Этот параметр был добавлен в MySQL 3.23.17.

Сценарий `mysql.server`

Сценарий `mysql.server` предназначается для запуска и остановки сервера `mysqld` вызовом сценария `mysqld_safe` (или, в зависимости от используемой версии MySQL, `safe_mysqld`). Сценарий `mysql.server` — это сценарий оболочки, поэтому он отсутствует в ОС Windows. Этот параметр был добавлен в MySQL 3.22.7.

Использование

```
mysql.server start
mysql.server stop
```

Обычно сценарий `mysql.server` используется в ОС типа System V и устанавливается в одном из рабочих каталогов в дереве каталога `/etc`. Система запускает сервер вызовом сценария с аргументом `start` во время запуска системы. Система отключает сервер вызовом сценария с аргументом `stop` во время остановки сервера. Этот сценарий также может запускаться вручную с добавлением аргумента `start` или `stop`.

Программа `mysqlshow`

Программа `mysqlshow` создает список баз данных, таблицы внутри баз данных, а также предоставляет информацию о столбцах и индексах таблиц. Эта программа является своего рода интерфейсом командной строки SQL-оператора `SHOW`.

Использование

```
mysqlshow [options] [db_name [tbl_name [col_name]]]
```

Если имя базы данных *db_name* не определено, программа `mysqlshow` выводит список всех баз данных, размещенных на компьютере сервера. Если задано имя базы данных, но отсутствует название таблицы, выводится список всех таблиц заданной базы данных. Если же указываются имена базы данных и таблицы, но отсутствуют названия столбцов, выводится список всех столбцов заданной таблицы. Если же определены все три названия, программа `mysqlshow` отображает информацию только о заданном столбце.

Если последний аргумент включает символ шаблона оболочки ("*" или "?"), выводимые данные ограничиваются значениями, соответствующими этим шаблонам. Символы "*" и "?" используются как SQL-символы "%" и "_" для оператора `LIKE`. Начиная с MySQL 3.22.26 `mysqlshow` также интерпретирует как групповые символы '%' и '_'.

Стандартные параметры, поддерживаемые программой mysqlshow

```
--character-sets-dir --host       --socket
--compress           --password   --user
--debug              --pipe       --verbose
--help-port          --version
```

Параметр `--character-sets-dir` был добавлен в MySQL 3.23.21. Начиная с MySQL 4 `mysqlshow` также поддерживает стандартные параметры SSL.

Специальные параметры mysqlshow

- `--status`, `-i` (*boolean*). Выводит информацию такого же вида, что и оператор `SHOW TABLE STATUS`.

  Параметр `--status` впервые появился в MySQL 3.23.

- `--keys`, `-k` (*boolean*). Помимо информации о столбцах таблицы, отображает также информацию о ее индексах. Этот параметр вступает в силу только при определении имени таблицы.

Сценарий safe_mysqld

`safe_mysqld` — это имя, под которым до MySQL 4 работал сценарий `mysqld_safe`. Данные о его использовании можно найти в описании сценария `mysqld_safe`, там же описана последовательность действий при переходе с MySQL 3.x на MySQL 4.x.

E

Программный интерфейс приложений C

В этом приложении...

Компилирование и редактирование связей

Типы данных программного интерфейса приложений C

Функции программного интерфейса приложений C

В этом приложении описывается программный интерфейс приложений языка C, используемый клиентской библиотекой MySQL. Программный интерфейс приложений (application programming interface — API) состоит из набора функций, предназначенных для взаимодействия с серверами MySQL и базами данных, а также из набора используемых этими функциями типов данных. Функции клиентской библиотеки можно классифицировать по следующим категориям.

- Функции управления соединением для установки и завершения соединений с сервером.
- Функции диагностирования ошибок.
- Функции конструирования и выполнения запросов, предназначенные для создания запросов и отправки их на сервер.
- Функции обработки результатов, полученных от запросов, возвращающих данные.
- Информационные функции, предоставляющие информацию о клиенте, сервере, версии протокола и текущем соединении.
- Функции администрирования, контролирующие работу сервера.

- Функции обработки процессов, предназначенные для создания процессовых клиентов.
- Функции для обмена данными с вложенным сервером MySQL, `libmysqld`.
- Отладочные функции, предназначенные для генерирования отладочной информации.
- Устаревшие функции. Если не сказано ничего другого, можно предположить, что все описанные здесь функции представлены как минимум в MySQL 3.22.0.

Данное приложение следует рассматривать как справочное руководство с небольшими фрагментами кода, иллюстрирующими примеры использования клиентской библиотеки. Полное описание клиентских программ и замечания по их написанию см. в главе 6, "Программный интерфейс MySQL C API".

Компилирование и редактирование связей

На исходном уровне клиентская библиотека определяется файлом заголовка `mysql.h`, который может включаться в состав исходных файлов:

```
#include <mysql.h>
```

Чтобы указать компилятору месторасположение этого файла, воспользуйтесь параметром `-I`*path*, где *path* — это путь к каталогу, в котором установлены файлы заголовков MySQL. Если файлы заголовка MySQL инсталлированы, например в каталоге `/usr/include/mysql` или `/usr/local/mysql/include`, исходный файл `my_func.c` можно откомпилировать с помощью следующих команд:

```
% gcc -I/usr/include/mysql -c my_func.c
% gcc -I/usr/local/mysql/include -c my_func.c
```

Другие файлы заголовков MySQL можно найти в том же каталоге, что и файл `mysql.h`. Файл `mysql_com.h`, например, содержит константы и макросы, используемые для интерпретации результирующих метаданных запроса. Файлы `errmsg.h` и `mysqld_error.h` содержат константы кодов ошибок. (Этот файл входит в состав файла `mysql.h`, поэтому, чтобы просмотреть его содержимое, достаточно включить файл `mysql.h`.)

На уровне объектов клиентская библиотека предоставляется как библиотека `mysqlclient`. При редактировании связей определите в команде связывания параметр `-lmysqlclient`. Возможно, придется также с помощью параметра `-L`*path*, где *path* — это путь к каталогу, в котором установлены файлы заголовков MySQL, указать редактору связей месторасположение библиотеки:

```
% gcc -o myprog myprog.o -L/usr/lib/mysql -lmysqlclient
% gcc -o myprog myprog.o -L/usr/local/mysql/lib -lmysqlclient
```

Завершение работы команды редактирования связей с ошибкой "unresolved symbol" свидетельствует о том, что редактору связей необходимо задать поиск дополнительных библиотек. Такими библиотеками могут быть математическая библиотека (`-lm`) и библиотека `zlib` (`-lz` или `-lgz`).

Простейшим способом определения соответствующего каталога для компиляции файла заголовков или флагов библиотек для связывания является использование про-

граммы mysql_config, которая имеется в MySQL 3.23.21. Для определения флагов, которые соответствуют вашей системе, запустите ее следующим образом:

```
% mysql_config --cflags
-I'/usr/local/mysql/include/mysql'
% mysql_config --libs
-L'/usr/local/mysql/lib/mysql' -lmysqlclient -lz -lcrypt -lnsl -lm
```

Этот листинг является только примером и, скорее всего, в вашей системе приобретет другой вид.

Типы данных программного интерфейса приложений C

Типы данных клиентской библиотеки MySQL предназначены для представления объектов, которые обрабатываются на протяжении сеанса работы с сервером. Существуют отдельные типы данных для соединения, результатов запроса, строк результирующего набора и метаданных (описательная информация о столбцах результирующего набора). Заметьте, что в последующем материале понятия "столбец" и "поле" являются синонимами.

Скалярные типы данных

Скалярные типы данных MySQL представляют большие целочисленные и логические значения, а также смещения полей.

- my_bool. Логический тип, используемый для значений, возвращаемых функциями mysql_change_user(), mysql_eof() и mysql_thread_init().

- my_ulonglong. Тип длинного целого числа, который используется для значений, возвращаемых такими функциями, как mysql_affected_rows(), mysql_num_rows() и mysql_insert_id(). Эти функции возвращают либо значения счетчиков строк, либо другие потенциально большие числа.

  Для вывода значения my_longlong необходимо преобразовать его в unsigned long и воспользоваться форматом "%lu", например:

  ```
  printf ("Row count = %lu\n", (unsigned long) mysql_affected_rows
  (conn));
  ```

  Если этого не сделать, в некоторых системах это значение не будет распечатано правильно, так как не существует стандартов распечатки длинных значений типа long функцией printf(). Однако, если распечатываемое значение может превышать максимальное значение, допустимое для unsigned long (2^{32}-1), формат %lu может не сработать.

  Для того чтобы проверить наличие каких-либо специфических средств распечатки значений, можно обратиться к документации по функции printf(). Например, может присутствовать спецификатор формата %llu.

- MYSQL_FIELD_OFFSET. Этот тип данных используется функциями mysql_field_tell() и mysql_field_seek() для представления смещения внутри набора структур MYSQL_FIELD текущего результирующего набора.

- MYSQL_ROW_OFFSET. Этот тип данных используется функциями mysql_field_tell() и mysql_field_seek() для представления смещения внутри набора строк текущего результирующего набора.

Нескалярные типы данных

Нескалярными типами данных MySQL являются структуры и массивы. Каждую структуру MYSQL или MYSQL_RES следует рассматривать как своего рода "черный ящик". Это означает, что ссылаться можно только на всю структуру сразу, но не на отдельные ее элементы, хотя типы MYSQL_ROW и MYSQL_FIELD подобных ограничений не имеют. Поэтому доступ к ним можно получать открыто для считывания данных и метаданных, возвращаемых в виде результирующего набора запроса.

- MYSQL. Основной тип клиентской библиотеки – структура MySQL, используемая механизмом обработки соединений. Этот дескриптор содержит информацию о состоянии соединения с сервером. Пользователь создает структуру MySQL, после чего передает ее программе соединения для открытия сеанса работы с сервером. После установления соединения дескриптор можно использовать для посылки запросов, генерирования результирующих наборов, получения сообщений об ошибках и других операций. После завершения соединения передайте дескриптор на mysql_close(), и данный дескриптор больше никогда не будет использоваться.
- MYSQL_FIELD. Клиентская библиотека использует структуры MYSQL_FIELD для предоставления метаданных о столбцах результирующего набора (по одной структуре на каждый столбец). Количество структур MYSQL_FIELD определяет функция mysql_num_fields(). Пользователь может получать доступ к структурам полей посредством вызова функции mysql_fetch_field() или перемещаться в любом направлении по структурам с помощью функций mysql_field_tell() и mysql_field_seek().

  Структура MYSQL_FIELD исключительно полезна для представления или интерпретации содержимого строк данных. Она имеет следующий вид:

```
typedef struct st_mysql_field {
        char *name;
        char *org_name;
        char *table;
        char *org_table;
        char *db;
        char *def;
        unsigned long length;
        unsigned long max_length;
        unsigned int flags;
        unsigned int decimals;
        enum enum_field_types type;
} MYSQL_FIELD;
```

  Элементы структуры MYSQL_FIELD имеют следующие значения.

  - name. Имя столбца, представленное в виде строки, завершающейся символом NULL. Если значения столбцов рассчитываются как результаты выражений, имя name представляет собой это выражение в строковой форме. Если столбец или выражение заданы в качестве псевдонима, имя name яв-

ляется именем псевдонима. Результатом выполнения следующего запроса, например, станут значения name столбцов "my_col", "4*(my_col+1)", "mc" и "myexpr":

```
SELECT my_col, 4*(my_col+1), my_col AS mc, 4*(mycol+1) AS
myexpr ...
```

- org_name. Этот элемент подобен name, за исключением того, что игнорируются псевдонимы столбцов, т.е. org_name представляет оригинальное имя столбца. Для столбца, который вычисляется как результат выражения, элемент org_name является пустой строкой. Начиная с MySQL 4.1 этот элемент не используется.
- table. Имя таблицы, столбец которой считывается, представленное в виде строки, завершающейся символом NULL. Если значения столбца рассчитываются как результаты выражений, table представляет собой пустую строку. Например, если запустить запрос, аналогичный следующему, имя таблицы для первого столбца (mycol) будет равно mytbl, а для второго (mycol+0) окажется пустой строкой:

  ```
  SELECT mycol, mycol+0 FROM mytbl ...
  ```

- org_table. Этот элемент аналогичен элементу table, за исключением того, что псевдонимы будут проигнорированы, т.е. org_table представляет оригинальное имя таблицы. Для вычисляемых столбцов элемент org_table является пустой строкой. Этот элемент до MySQL 4.1 не использовался.
- db. База данных, где расположена таблица. Завершается символом NULL. Для вычисляемых столбцов элемент db является пустой строкой. Этот элемент до MySQL 4.1 не использовался.
- def. Значение по умолчанию для столбца, представленное в форме строки, завершающейся символом NULL. Этот элемент структуры MYSQL_FIELD устанавливается только посредством вызова функции mysql_list_fields(). Во всех остальных случаях он имеет значение NULL. (Чтобы получить значения столбцов по умолчанию, можно запустить запрос DESCRIBE *tbl_name* или SHOW FIELDS COLUMNS *tbl_name* и исследовать результирующий набор.)
- length. Длина столбца в соответствии с параметрами оператора CREATE TABLE, с помощью которого была создана таблица. Длина столбца рассчитывается на основании выражения и определяется длинами элементов выражения.
- max_length. Длина самого длинного значения столбца, реально присутствующего в результирующем наборе. Если столбец, например, содержит значения "Bill", "Jack" и "Belvidere", значение max_length для него будет равно 9.

  Поскольку значение max_length можно определить только после вывода всех строк, оно отвечает действительности только для результирующих наборов, созданных функцией mysql_store_result(). Для наборов, созданных функцией mysql_use_result(), значение max_length равно 0.

- `flags`. Элемент `flags` определяет атрибуты столбца. Каждый отдельный атрибут представлен собственным разрядом внутри значения `flags`. Значения разрядов могут проверяться с помощью констант разрядной маски, представленной в табл. Е.1. Чтобы определить, являются ли значения столбца беззнаковыми (UNSIGNED), достаточно проверить значение `flags` следующим образом:

```
if (field->flags & UNSIGNED_FLAG)
  printf ("%s значения имеют тип UNSIGNED\n", field->name;
```

Таблица Е.1. Значения элемента `flags` структуры `MYSQL_FIELD`

Значение `flags`	Назначение
AUTO_INCREMENT_FLAG	Столбец имеет атрибут AUTO_INCREMENT
BINARY_FLAG	Столбец имеет атрибут BINARY
MULTIPLE_KEY_FLAG	Столбец является частью индекса NON_UNIQUE
NOT_NULL_FLAG	Столбец не может содержать значения NULL
PRI_KEY_FLAG	Столбец является частью PRIMARY KEY
UNIQUE_KEY_FLAG	Столбец является частью индекса UNIQUE
UNSIGNED_FLAG	Столбец имеет атрибут UNSIGNED
ZEROFILL_FLAG	Столбец имеет атрибут ZEROFILL

BINARY_FLAG представляет собой набор строковых столбцов, чувствительных к регистру. Он включает столбцы, для которых ключевое слово BINARY определено явным образом (как CHAR BINARY), а также столбцы BLOB. Несколько констант `flags` отражают типы столбцов, а не их атрибуты; но сейчас от их использования отказались, так как для определения типа столбца необходимо использовать `field->type`. В табл. Е.2 содержится перечень всех исключенных констант.

Некоторые константы `flags` отражают тип столбца, а не его атрибуты. В настоящее время они исключены, поэтому для определения типа столбца необходимо использовать запись `field->type`. Исключенные из MySQL константы описаны в табл. Е.2.

Таблица Е.2. Исключенные значения элемента `flags` структуры `MYSQL_FIELD`

Значение `flags`	Назначение
BLOB_FLAG	Столбец содержит значения типа BLOB или TEXT
ENUM_FLAG	Столбец содержит значения типа ENUM
SET_FLAG	Столбец содержит значения типа SET
TIMESTAMP_FLAG	Столбец содержит значения типа TIMESTAMP

- `decimals`. Число десятичных знаков для столбцов числового типа. Для нечисловых столбцов это значение равно нулю. Так, например, столбец DECIMAL(8,3) имеет значение `decimals`, равное 3, а столбец типа BLOB — 0.
- `type`. Тип столбца. Определяется как результат выражения и устанавливается в зависимости от типов элементов этого выражения. Так, например, если столбец `mycol` имеет тип VARCHAR(20), элемент `type` бу-

дет иметь значение FIELD_TYPE_VAR_STRING, а тип type для столбца типа LENGTH(mycol) будет FIELD_TYPE_LONGLONG. Возможные значения элемента type представлены в файле mysql_com.h и описываются в табл. Е.3.

Таблица Е.3. Значения элемента type структуры MYSQL_FIELD

Значение type	Тип столбца
FIELD_TYPE_BLOB	BLOB или TEXT
FIELD_TYPE_DATE	DATE
FIELD_TYPE_DATETIME	DATETIME
FIELD_TYPE_DECIMAL	DECIMAL или NUMERIC
FIELD_TYPE_DOUBLE	DOUBLE или REAL
FIELD_TYPE_ENUM	ENUM
FIELD_TYPE_FLOAT	FLOAT
FIELD_TYPE_INT24	MEDIUMINT
FIELD_TYPE_LONG	INT
FIELD_TYPE_LONGLONG	BIGINT
FIELD_TYPE_NULL	NULL
FIELD_TYPE_SET	SET
FIELD_TYPE_SHORT	SMALLINT
FIELD_TYPE_STRING	CHAR
FIELD_TYPE_TIME	TIME
FIELD_TYPE_TIMESTAMP	TIMESTAMP
FIELD_TYPE_TINY	TINYINT
FIELD_TYPE_VAR_STRING	VARCHAR
FIELD_TYPE_YEAR	YEAR

В исходных файлах более старых версий можно найти ссылки на тип FIELD_TYPE_CHAR. Это однобайтовый тип, который в современных версиях имеет название FIELD_TYPE_TINY.

- MYSQL_RES. Запросы, которые возвращают клиенту запрашиваемые данные, например SELECT и SHOW, манипулируют в процессе своей работы результирующим набором, представленным в виде структуры MYSQL_RES. Эта структура содержит информацию о возвращаемых запросом строках.

  После считывания результирующего набора пользователь получает возможность вызывать функции интерфейса API для получения данных результирующего набора (значения строк набора) или метаданных (информация о результирующем наборе, например количество столбцов, их типы, длины и т.п.).

- MYSQL_ROW. Тип MYSQL_ROW содержит значения одной строки данных, представленной в виде массива строк определенного размера. В строковой форме функциями возвращаются все значения, даже числовые. Единствен-

ное исключение составляют строковые значения NULL, которые представляются в структуре MYSQL_ROW С-указателем NULL.

Число значений в строке определяется функцией mysql_num_fields(). Значение i-го столбца в строке задается элементом row[i], где i может изменяться в пределах от 0 до mysql_num_fields(res_set)-1. (res_set – это указатель на результирующий набор MYSQL_RES.)

Заметьте, что тип MYSQL_ROW уже является указателем, поэтому переменную строки можно описывать следующим образом:

```
MYSQL_ROW row;        /* правильно */
```

Нельзя описывать ее так:

```
MYSQL_ROW *row;       /* неправильно */
```

Значения типа MYSQL_ROW всегда заканчиваются нулями, поэтому недвоичные значения могут рассматриваться как строки, заканчивающиеся нулем. Однако информационные значения с двоичными данными также могут включать нули, поэтому должны обрабатываться как строки определенной длины. Для определения длин значений строки воспользуйтесь функцией mysql_fetch_lengths():

```
unsigned long *length;
length = mysql_fetch_length (res_set);
```

Длина значения *i*-го столбца строки представляется значением length[*i*]. Для значений NULL эта функция возвращает длину 0.

Макросы

Файл mysql.h содержит несколько макросов, которые позволяют легко исследовать значения элементов структуры MYSQL_FIELD. Макрос IS_NUM() проверяет элемент type, а остальные описанные здесь макросы исследуют элемент flags.

- Макрос IS_NUM() возвращает значение "True" (не равен нулю), если значения столбцов имеют числовой тип:

```
if (IS_NUM (field->type))
    printf ("Поле %s является цифровым \n", field->name);
```

- Макрос IS_PRI_KEY() возвращает значение "True", если столбец является частью первичного ключа:

```
if (IS_PRI_KEY (field->flags))
    printf ("Поле %s является частью первичного ключа \n",
    field->name);
```

- Макрос IS_NOT_NULL возвращает значение "True", если столбец не может содержать значения NULL:

```
if (IS_NOT_NULL (field->flags))
   printf ("Поле %s не может содержать значения NULL\n",
field->name);
```

- Макрос IS_BLOB возвращает значение "True", если столбец имеет тип BLOB или TEXT. Однако этот макрос также тестирует в элементе flags неиспользуемый бит BLOB_FLAG, поэтому макрос IS_BLOB() также не используется.

Функции программного интерфейса приложений C

Функции клиентской библиотеки можно классифицировать по следующим категориям. Некоторые параметры функции появляются в описаниях всех функций и имеют следующее значение.

- conn. Указатель на дескриптор MYSQL-соединения при подключении к серверу.
- res_set. Указатель на структуру результирующего набора MYSQL_RES.
- field. Указатель на структуру информации о столбцах MYSQL_FIELD.
- row. Строка данных MYSQL_ROW из результирующего набора.
- row_num. Номер строки в пределах результирующего набора, начиная с нуля и до числа выбранных строк минус один.
- col_num. Количество столбцов в строке результирующего набора, начиная с нуля до числа выбранных строк минус один.

Для краткости изложения представленные значения этих параметров можно использовать в последующих описаниях функций. Все приведенные функции присутствуют в клиентской библиотеке в версии MySQL 3.21.0 и выше, если не указано другое.

Функции управления соединением

Эти функции применяются для установления и разрыва соединений с сервером, установки параметров, влияющих на способ установления соединения, восстановления временно приостановленных соединений и смены текущего пользователя. Типичная последовательность обработки включает вызов функции mysql_init() для инициализации дескриптора соединения, mysql_real_connect() для установления соединения и mysql_close() для завершения соединения, если в нем нет больше необходимости. Если необходимо использовать специальные параметры или установить закодированное SSL-соединение, вызовите mysql_options() или mysql_ssl_set() после вызова функции mysql_init() и перед вызовом функции mysql_real_connect().

- my_bool
  mysql_change_user (MYSQL *conn,
  const char *user_name,
  const char *password,
  const char *db_name);

  Изменяет имя пользователя и базу данных по умолчанию для определенного указателем conn соединения. Эта база данных становится базой по умолчанию для ссылок на таблицы, в которых спецификатор базы данных не определен. Если аргумент db_name равен NULL, база данных по умолчанию не устанавливается.

  Функция mysql_change_user() возвращает значение "True", если пользователю разрешено подключение к серверу и он обладает достаточными полномочиями для доступа к указанной базе. В любом другом случае функ-

ция завершается ошибкой, а текущий пользователь и база данных остаются неизменными.

Эту функцию удобно использовать для изменения текущего пользователя, не закрывая соединение и открывая его снова с другими параметрами. Она может быть использована для создания устойчивых соединений в программах, которые во время их выполнения используются различными пользователями.

Функция `mysql_change_user()` впервые появилась в MySQL 3.23.3.

- `void`
  `mysql_close` `(MYSQL *conn);`

  Закрывает соединение, определенное указателем `conn`. Эту программу рекомендуется вызывать по завершении сеанса работы с сервером. Если дескриптор соединения автоматически инициируется функцией `mysql_init()`, функция `mysql_close()` обнуляет его.

  Не следует вызывать функцию `mysql_close()`, если попытка установки соединения оказалась неудачной.

- `MYSQL *`
  `mysql_init` `(MYSQL *conn);`

  Инициализирует дескриптор соединения и возвращает указатель на него. Если аргумент `conn` указывает на уже существующую структуру `MYSQL`, функция `mysql_init` инициализирует этот механизм и возвращает его адрес:

```
MYSQL conn_struct, *conn;
conn = mysql_init (&conn_struct);
```

  Если значение `conn` равно `NULL`, функция `mysql_init()` инициализирует новый дескриптор и возвращает адрес:

```
MYSQL *conn;
conn = mysql_init(NULL);
```

  Второй подход предпочтительнее первого, так как клиентская библиотека размещает и инициализирует дескриптор, тем самым избегая проблем, которые могут возникнуть при использовании общих библиотек при обновлении MySQL, при которой используется другая внутренняя организация структуры `MYSQL`.

  В случае безуспешного завершения функция `mysql_init()` возвращает значение `NULL`. Это может произойти в том случае, когда функция `mysql_init()` не в состоянии назначить новый дескриптор.

  Если функция `mysql_init()` определяет новый дескриптор, то при закрытии соединения функция `mysql_close()` обнуляет его автоматически.

  Функция `mysql_init()` впервые появилась в MySQL 3.22.1.

- `int`
  `mysql_options` `(MYSQL *conn,`
  `enum mysql_option option,`
  `const char *arg);`

  Эта функция позволяет управлять состоянием соединения более точно, чем с помощью одной функции `mysql_real_connect()`. Вызовите ее после функ-

ции mysql_init() и перед запуском mysql_real_connect(). Для установки нескольких параметров функцию mysql_options() можно вызывать несколько раз. (После многократных вызовов функции mysql_options() при установке определенного параметра mysql_real_connect() использует самое последнее значение этого параметра.)

Аргумент option определяет устанавливаемый параметр соединения. Дополнительная информация о параметре, если таковая имеется, задается с помощью аргумента arg. (Обратите внимание, arg – это указатель.) Если дополнительная информация не нужна, аргумент arg устанавливается равным NULL.

Функция mysql_options() возвращает нуль при успешном завершении и ненулевое значение, если значение option неизвестно. Эта функция была добавлена в MySQL 3.22.1, но некоторые из имеющихся в данный момент значений были добавлены позднее, что нашло отражение в описании параметров. Для задания этого параметра можно использовать следующие параметры.

- MYSQL_INIT_COMMAND. Определяет запрос, выполняемый сразу после подключения к серверу. Собственно запрос задается в аргументе arg в виде строки, завершающейся значением NULL. Этот же запрос выполняется и после восстановления соединения (например, при вызове функции mysql_ping()). Все возвращаемые этим запросом данные результирующего набора пропускаются.

  Параметр MYSQL_INIT_COMMAND был добавлен в MySQL 3.22.10.

- MYSQL_OPT_COMPRESS. Определяет, что в соединении при обмене данными между сервером и клиентом должен использоваться протокол сжатия, если сервер, конечно, его поддерживает. Аргумент arg имеет значение NULL. Определить сжатие можно также и при вызове функции mysql_real_connect().

- MYSQL_OPT_CONNECT_TIMEOUT. Задает небольшой временной перерыв в соединении (в секундах). Аргумент arg указывает на число unsigned int, определяющее длительность перерыва.

- MYSQL_OPT_LOCAL_INFILE. Включает или отключает использование режима LOAD DATA LOCAL. Значение arg содержит указатель на значение типа unsigned int, которое для включения режима должно быть ненулевым, или равным нулю для его отключения. Этот параметр не будет работать, если сервер будет настроен так, чтобы навсегда запретить LOAD DATA LOCAL.

  Параметр MYSQL_OPT_LOCAL_INFILE был добавлен в MySQL 3.23.49. (В MySQL 3.22.15–3.23.48 LOAD DATA LOCAL всегда включен, а до 3.22.15 он отсутствовал.)

- MYSQL_OPT_NAMED_PIPE. Определяет, что соединение с сервером должно устанавливаться через именованный канал. Аргумент arg имеет значение NULL. Этот параметр предназначен только для клиентов Windows и используется только при установлении соединения с серверами Windows NT.

  Параметр MYSQL_INIT_COMMAND был добавлен в MySQL 3.22.5.

- MYSQL_READ_DEFAULT_FILE. Определяет конфигурационный файл, содержащий параметры соединения. Считываемые функцией параметры размещаются в группе [client] файла. Значение arg указывает на строку, содержащую имя файла, завершающуюся символом NULL. Параметры будут считываться из группы [client] непосредственно в этот файл при использовании MYSQL_READ_DEFAULT_GROUP для задания имени группы, параметры из этой группы также будут считываться из конфигурационного файла.

  Параметр MYSQL_READ_DEFAULT_FILE был добавлен в MySQL 3.22.10.

- MYSQL_READ_DEFAULT_GROUP. Определяет считываемую группу из конфигурационного файла, заданного параметром MYSQL_READ_DEFAULT_FILE. Значение arg указывает на строку, содержащую имя группы, завершающуюся символом NULL. (Имя группы задается без квадратных скобок.) Параметры этой группы считываются в дополнение к параметрам группы [client]. Если при этом конфигурационный файл не задан, клиентская библиотека ищет стандартные конфигурационные файлы и считывает их содержимое.

  Заметим, что если не заданы ни MYSQL_READ_DEFAULT_FILE, ни MYSQL_READ_DEFAULT_GROUP, конфигурационные файлы не считываются.

  Параметр MYSQL_READ_DEFAULT_GROUP был добавлен в MySQL 3.22.10.

- MYSQL_SET_CHARSET_DIR. Задает путь к каталогу, где располагаются файлы кодировок. Значение arg указывает на строку, содержащую путь к каталогу, завершающуюся символом NULL. Каталог располагается на компьютере клиента; этот параметр используется тогда, когда клиенту требуется получить доступ к кодировкам, которые не были скомпилированы в библиотеку клиента, но для которых имеются файлы определения.

  Параметр MYSQL_SET_CHARSET_DIR был добавлен в MySQL 3.23.14.

- MYSQL_SET_CHARSET_NAME. Определяет имя стандартной кодировки. Значение arg указывает на строку, содержащую имя кодировки, завершающуюся символом NULL.

  MYSQL_SET_CHARSET_NAME был добавлен в MySQL 3.23.14.

  Параметры можно задавать в конфигурационном файле как в группе [client], так и в группе, которую можно задать с помощью параметра MYSQL_READ_DEFAULT_GROUP. Список параметров, имеющих отношение к клиентским программам, см. в разделе "Конфигурационные файлы" приложения Д, "Программы MySQL".

  В путях для ОС Windows, определенных с помощью параметров MYSQL_READ_DEFAULT_FILE или MYSQL_SET_CHARSET_DIR, символы '\' задаются как '/' или как '\\', хотя первый вариант предпочтительнее.

  В приведенном ниже примере функция mysql_options() вызывается для установки таких значений параметров соединения, чтобы функция mysql_read_connect() пыталась установить соединение через именованный канал с помощью протокола сжатия и временного перерыва в 10 секунд. Эти параметры считываются из групп [client] и [mygroup]

файла C:\my.cnf.extra. Сразу после установления соединения выполняется оператор SET SQL_BIG_TABLES=1.

```
MYSQL *conn
unsigned int timeout;

if ((conn = mysql_init (NULL) == NULL)
  ... обработка ошибок ...
mysql_options (conn, MYSQL_READ_DEFAULT_FILE,
"C:\my.cnf.special");
mysql_options (conn, MYSQL_READ_DEFAULT_GROUP, "mygroup");
mysql_options (conn, MYSQL_OPT_NAMED_PIPE, NULL);
timeout = 10;
mysql_options (conn, MYSQL_OPT_CONNECT_TIMEOUT, (char *)
&timeout);
 mysql_options (conn, MYSQL_INIT_COMMAND, "SET
SQL_BIG_TABLES=1");
mysql_options (conn, MYSQL_OPT_COMPRESS, NULL);
if (mysql_real_connect (conn, ...) == NULL)
  ... обработка ошибок ...
```

- `int` **`mysql_ping`** `(MYSQL *conn);`

  Проверяет, находится ли еще определенное указателем conn соединение в активном состоянии. Если соединение оборвано, функция mysql_ping() восстанавливает его с теми же параметрами, которые используются при установлении исходного соединения. Соответственно, не следует вызывать функцию mysql_ping(), если предыдущее соединение не было успешно установлено функцией mysql_real_connect(). Возвращает нуль, если соединение активно или было успешно восстановлено, при возникновении ошибки возвращает ненулевое значение.

  Функция mysql_ping() впервые появилась в MySQL 3.22.1.

- `MYSQL *` **`mysql_real_connect`** `(MYSQL *conn,`

  ```
  const char *host_name,
  const char *user_name,
  const char *password,
  const char *db_name,
  unsigned int port_num,
  const char *socket_name,
  unsigned int flags);
  ```

  Подключается к серверу и возвращает указатель на дескриптор соединения. Аргумент conn должен быть указателем на существующий дескриптор соединений, инициализированный функцией mysql_init(). Адрес дескриптора возвращается при успешном установлении соединения. Если при подключении возникла ошибка, функция возвращает значение NULL.

  Чтобы в случае неудачной попытки установления соединения получить информацию об ошибке, можно задать значение дескриптора conn функциям mysql_errno() и mysql_error(). Не следует однако задавать значение conn другим программам клиентской библиотеки, которые предполагают успешное установление соединения.

  Аргумент host_name определяет имя сервера, к которому выполняется подключение. В табл. Е.4 показано, какое соединение пытается установить

клиент при различных значениях аргумента host_name в Windows и UNIX. (Если соединение устанавливается через разъем UNIX или именованный канал, имя разъема или канала определяются параметром socket_name.) Следует напомнить, что использование имени "localhost" является особенностью систем UNIX. Это свидетельствует о том, что вы хотите использовать разъем UNIX, а не соединение TCP/IP. Для подключения к локальному узлу с использованием TCP/IP-соединения в качестве значения host_name необходимо задать "127.0.0.1" (IP-адрес интерфейса "обратной петли" локального компьютера), а не строку "localhost".

Таблица Е.4. Типы клиентских соединений, определяемые типом имен компьютера сервера

Значение имени компьютера	Тип соединения UNIX	Тип соединения Windows
Имя компьютера	TCP/IP-соединение с указанным компьютером	TCP/IP-соединение с указанным компьютером
IP-адрес	TCP/IP-соединение с указанным компьютером	TCP/IP-соединение с указанным компьютером
localhost	Соединение с локальным компьютером через разъем UNIX	TCP/IP-соединение с локальным компьютером
127.0.0.1	Соединение с указанным компьютером через разъем UNIX	TCP/IP-соединение с указанным компьютером
.	Не применяется	Соединение с локальным компьютером через именованный канал
NULL	Соединение с локальным компьютером через разъем UNIX	TCP/IP-соединение с локальным компьютером. В Windows NT перед установлением этого соединения выполняется попытка установить соединения через именованный канал

Имя пользователя MySQL задается аргументом user_name. Если он имеет значение NULL, клиентская библиотека отправляет имя, установленное по умолчанию. В работающих под управлением UNIX системах именем по умолчанию является регистрационное имя, а в работающих под Windows — имя, определенное переменной среды USER (если она установлена), либо ODBC.

Аргумент password устанавливает пароль. Если он имеет значение NULL, пользователь сможет подключиться только в том случае, если таблица разрешений будет содержать запись с его именем, компьютером, с которого осуществляется подключение, и пустым полем пароля.

Используемая рабочая база задается аргументом db_name. Если он имеет значение NULL, исходная база данных не выбирается.

Аргумент port_num определяет номер порта, используемого для установления TCP/IP-соединения. Если этот аргумент имеет значение 0, выбирается установленный по умолчанию номер порта.

Аргумент socket_name является именем разъема UNIX, которое будет использоваться для подключения к "localhost" под UNIX, или именем име-

нованного канала для соединений в Windows. Если оно равно NULL, используется стандартный разъем или канал.

Номер порта и имя разъема выбираются в соответствии со значением `host_name`. Если подключение к серверу выполняется с локального компьютера, функция `mysql_real_connect()` пытается установить соединение с помощью разъема домена UNIX (при работе под управлением UNIX) или именованного канала (при работе под управлением Windows). Во всех остальных случаях соединение устанавливается с помощью протокола TCP/IP.

Аргумент `flags` может иметь значение 0, отражающее полное отсутствие параметров, или одно (или несколько) из перечисленных ниже значений Устанавливаемые в нем параметры влияют на функционирование сервера.

- `CLIENT_COMPRESS`. Задает использование соединением "клиент/сервер" протокола сжатия при условии, что протокол это поддерживает.
- `CLIENT_FOUND_ROWS`. Запросы `UPDATE` возвращают число соответствующих условиям строк, а не число измененных строк. Применение этого параметра может заставить оптимизатор MySQL производить изменения медленнее.
- `CLIENT_IGNORE_SPACE`. Обычно за именами функций должны сразу же следовать скобки, заключающие перечень аргументов, без какого-либо пробела. Этот параметр сообщает серверу, что он должен проигнорировать все пробелы между именем функции и списком аргументов. Это имеет определенное побочное последствие, в результате чего все имена функций становятся зарезервированными словами.

  Этот параметр был добавлен в MySQL 3.22.7.
- `CLIENT_INTERACTIVE`. Идентифицирует данного клиента как интерактивного. Этот факт дает понять серверу, что он может закрыть соединение после определенного периода времени неактивности, равного значению переменной сервера `interactive_timeout`. Обычно значение этой переменной используется.

  Данный параметр был добавлен в MySQL 3.23.28.
- `CLIENT_NO_SCHEMA`. Запрещает использовать схему *`db_name.tbl_name.col_name`*. При задании этого параметра сервер разрешает в запросах ссылки только в формате *`tbl_name.col_name`*, *`tbl_name`* или *`col_name`*.
- `CLIENT_ODBC`. Работает с клиентом как с ODBC-клиентом.

  Этот параметр был добавлен в MySQL 3.22.4.
- `CLIENT_SSL`. Данный параметр является внутренним для SSL-соединений, и клиентские программы не могут его использовать.
- `CLIENT_TRANSACTIONS`. Этот параметр свидетельствует о том, что сервер "знает" о транзакциях. Другими словами, клиент "понимает", что сервер сделает откат активной транзакции при разрыве соединения во время транзакции, в том случае, если он не работает в режиме автовыполнения.

  Этот параметр был добавлен в MySQL 3.23.17.

Значения флагов являются разрядными, что позволяет комбинировать их с помощью символов "|" или "+", например: CLIENT_COMPRESS | CLIENT_ODBC или CLIENT_COMPRESS + CLIENT_ODBC.

Функция mysql_real_connect() впервые появилась в MySQL 3.21.10. Параметр db_name был добавлен в версии 3.22.0. Применение функции mysql_init() для инициализации аргументов MySQL берет свое начало в версии 3.22.1.

```
int mysql_ssl_set (MYSQL *conn,
const char *key,
const char *cert,
const char *ca,
const char *capath,
const char *cipher);
```

Эта функция используется для установки защищенного соединения с сервером MySQL с использованием протокола SSL. Если OpenSSL в клиентской библиотеке не скомпилирован, функция mysql_ssl_set() не сделает ничего. В противном случае она устанавливает всю необходимую информацию для установки зашифрованного соединения при вызове функции mysql_real_connect(). (Другими словами, для установки безопасного соединения сначала необходимо вызвать функцию mysql_ssl_set(), а затем mysql_real_connect().)

Функция mysql_ssl_set() всегда возвращает 0; любая ошибка установки протокола SSL приведет к ошибке во время вызова функции mysql_real_connect().

key представляет путь к ключевому файлу; cert — это путь к файлу сертификатов.

ca — путь к файлу бюро сертификации; capath — это путь к доверительным сертификатам, которые будут использоваться при проверке сертификатов.

cipher — это строка со списком используемых шифров. Любой неиспользуемый параметр должен иметь значение NULL.

Функция mysql_ssl_set() была добавлена в MySQL 4.0.0. Для ее использования требуется незначительная дополнительная перенастройка. (Подробнее о безопасных соединениях см. в главе 12, "Безопасность".)

Функции состояния и диагностики ошибок

Описанные в этом разделе функции применяются для определения причин возникновения ошибок и создания отчетов о них.

- `unsigned int mysql_errno (MYSQL *conn);`

  Возвращает код ошибки для последней вызываемой программы клиентской библиотеки, возвращающей значение состояния. Так, код ошибки 0 указывает на отсутствие ошибки. Все остальные значения кода указывают на наличие ошибок. Возможные коды ошибок описаны в файлах заголовков MySQL, errmsg.h и mysql_error.h.

```
if (mysql_errno (conn) == 0)
    printf ("Все в порядке\n");
else
    printf ("Что-то случилось!\n");
```

Функция `mysql_errno()` впервые появилась в MySQL 3.21.7.

- `const char * mysql_error (MYSQL *conn)`

  Возвращает строку, завершающуюся значением NULL и содержащую сообщение об ошибке последней вызываемой программы клиентской библиотеки, возвращающей значение состояния. Если ошибка отсутствует, эта функция возвращает пустую строку (строку с нулевой длиной, но не указатель NULL). Хотя обычно функция `mysql_error()` вызывается уже после появления ошибки, возвращаемое ею значение можно использовать для проверки наличия или отсутствия ошибки в работе:

```
char *err = mysql_error (conn);
if (mysql_error (conn)[0] == '\0') /* пустая строка? */
    printf ("Все в порядке \n");
else
    printf ("Что-то случилось! \n");
```

  Функция `mysql_error()` впервые появилась в MySQL 3.21.7 (возможно, даже чуть позже).

Функции создания и выполнения запросов

Описанные в этом разделе функции используются для отсылки запросов на сервер. Функция `mysql_real_escape_string()` позволяет создать запрос с помощью специальных символов, имеющих особое значение.

- `int mysql_query (MYSQL *conn, const char *query_str);`

  Эта функция отсылает запрос, заданный в виде строки, ограниченной символом NULL, на выполнение на сервер. Эта строка не должна содержать двоичных данных; в частности она не должна содержать байтов со значением NULL, так как функция `mysql_query()` обработает его как окончание запроса. Если ваш запрос содержит двоичные данные, воспользуйтесь функцией `mysql_real_query()`. Эта функция работает немного быстрее, чем `mysql_query()`.

- Функция `mysql_query()` возвращает нулевое значение в случае успешного завершения работы или ненулевое при ошибочном завершении. Успешным считается запрос, который сервер принял как допустимый и выполнил без ошибок. Успех не зависит от числа обработанных или возвращенных строк.

- `unsigned long mysql_real_escape_string (char *to_str, const char *from_str, unsigned long from_len)`

  Преобразует заданную строку, которая может содержать специальные символы, что позволяет использовать эту строку в качестве оператора SQL (при этом обработка ведется с учетом текущей кодировки). Допустимые специальные символы, а также их кодировки представлены в табл. Е.5.

  Единственными символами, которые для MySQL обозначаются управляющими символами, являются обратная косая черта и символ кавычки, огра-

ничивающий строку (‘'’ или ‘"’). Функция mysql_real_escape_string() убирает другие управляющие символы, чтобы сделать строки более “читабельными” и упростить их обработку в файлах регистрации.

Таблица Е.5. Кодировка специальных символов функции mysql_real_escape_string()

Специальный символ	Кодировка
NULL (ASCII 0)	\0 (обратная косая черта и нуль)
Обратная косая черта	\\ (обратная косая черта и обратная косая черта)
Одинарная кавычка	\' (обратная косая черта и одинарная кавычка)
Двойные кавычки	\" (обратная косая черта и двойные кавычки)
Переход на новую строку	\n (обратная косая черта и “n”)
Возврат каретки	\r (обратная косая черта и “r”)
<Ctrl+Z>	\Z (обратная косая черта и “Z”)

Кодируемый буфер задается как строка определенной длины. На этот буфер указывает аргумент from_str, а число разрядов в нем задается аргументом from_len. Функция mysql_real_escape_string() записывает результат в буфер, определенный аргументом to_str, и добавляет байт со значением NULL. Аргумент to_str должен указывать на существующий буфер длиной не меньше (from_len*2)+1 байт. (В наихудшем случае для кодирования каждого символа буфера from_str может потребоваться двухсимвольная последовательность, а также место для завершающего символа NULL.)

Функция mysql_real_escape_string() возвращает длину закодированной строки без учета завершающего пустого байта.

Результирующая закодированная строка не содержит внутренних нулей, однако завершается NULL, что позволяет использовать ее с такими функциями, как strlen() или strcat().

Записывая литеральную строку в программу, следует проявлять осторожность, чтобы полученная в результате кодирования функцией mysql_real_escape_string() строка не была ошибочно воспринята из-за лексикографических соглашений языка программирования C. Рассмотрим, например, следующую строку и выводимый результат функции:

```
to_len = mysql_real_escape_string (conn, to_str,
"\0\\'\"\n\r\032", 7);
printf ("to_len = %d, to_str = %s\n", to_len, to_str);
```

Результатом этого кода станет следующая строка:

```
to_len = 14, to_str = \0\\'\"\n\r\Z
```

Выведенное в буфер to_str значение очень похоже на строку третьего аргумента функции mysql_real_escape_string(), однако в действительности отличается от него.

Функция mysql_real_escape_string() была добавлена в MySQL 3.23.14 как замена mysql_escape_string(), которая не может работать с текущей кодировкой, и сейчас считается устаревшей. Для того чтобы создать код, который

будет работать в любой версии MySQL, включите такой фрагмент в любой исходный текст, который работает с функцией mysql_real_escape_string():

```
#if !defined(MYSQL_VERSION_ID) || (MYSQL_VERSION_ID<32314)
#define mysql_real_escape_string(conn,to_str,from_str,len) \
        mysql_escape_string(to_str,from_str,len)
#endif
```

Это позволит переключаться с функции mysql_real_escape_string() на mysql_escape_string(), когда исходный код компилируется для его использования с более старыми версиями MySQL.

- int **mysql_real_query** (MYSQL *conn, const char *query_string, unsigned long length);

  Считывая запрос, представленный в виде строки фиксированной длины, функция mysql_real_query() отправляет его на сервер для выполнения. Строка запроса может содержать двоичные данные (включая пустые байты). Его текст определяется указателем query_string, а длина задается аргументом length.

  Функция mysql_real_query() возвращает нуль в случае успешного выполнения запроса и ненулевое значение при возникновении ошибки. Успешным считается запрос, который сервер принял как допустимый и выполнил без ошибок. Успех не зависит от числа обработанных или возвращенных строк.

  До MySQL 4 аргумент length имел тип unsigned int.

- int **mysql_select_db** (MYSQL *conn, const char *db_name);

  Устанавливает базу данных, заданную аргументом db_name, как текущую. Эта база впоследствии используется по умолчанию для всех ссылок на таблицы, определенных без спецификатора базы данных. Применяющий эту функцию пользователь должен обладать полномочиями на доступ к базе данных, иначе функция mysql_select_db() не завершит успешно свою работу.

  Функция mysql_select_db() очень полезна для смены баз данных при соединении. Обычно база данных задается при вызове функции mysql_real_connect(), которая работает быстрее, чем при вызове mysql_select_db() после подключения.

  Функция mysql_select_db() возвращает нуль при успешном выполнении и ненулевое значение при возникновении ошибки.

Функции обработки результирующих наборов

После создания запросом результирующего набора специальные функции, описанию которых и посвящен этот раздел, позволяют произвести выборку этого результата и получить к нему доступ. Функции mysql_store_result() и mysql_use_result() создают результирующий набор, поэтому одну из них необходимо обязательно вызвать перед использованием описанных далее функций. Сравнение функций mysql_store_result() и mysql_use_result() представлено в табл. Е.6.

Таблица Е.6. Сравнение функций `mysql_store_result()` и `mysql_use_result()`

Функция `mysql_store_result()`	Функция `mysl_use_result()`
Все строки из результирующего набора выбираются функцией `mysql_store_result()`	Инициализирует результирующий набор, а выборка строк выполняется функцией `mysql_fetch_row()`
Требует больше памяти, поскольку все строки хранятся в памяти компьютера-клиента	Использует меньше памяти, поскольку записывается по одной строке
Работает медленней из-за переполнения, связанного с выделением памяти для всего результирующего набора	Работает быстрее, поскольку память необходимо выделять для хранения только текущей строки
Возвращаемое функцией `mysql_fetch_row()` значение NULL указывает на завершение результирующего набора и не используется для отображения ошибки	Возвращаемое функцией `mysql_fetch_row()` значение NULL указывает на завершение результирующего набора или наличие ошибки, поскольку сбой при обмене данными может привести к остановке операции выборки записей
Функция `mysql_num_rows()` может вызываться в любой момент после вызова функции `mysql_store_result()`	Функция `mysql_num_rows()` возвратит правильное число строк только после выборки всех строк
Функция `mysql_affected_rows()` является аналогом функции `mysql_num_rows()`	Функцию `mysql_affected_rows()` нельзя использовать
Произвольный доступ к результирующему набору возможен с помощью функций `mysql_data_seek()`, `mysql_row_seek()` и `mysql_row_tell()`	Произвольный доступ к результирующему набору невозможен. Строки могут обрабатываться только в порядке, возвращаемом с сервера. Функции `mysql_data_seek()`, `mysql_row_seek()` и `mysql_row_tell()` не используются
Доступ к таблицам блокируется только на период времени, необходимый для выборки строк данных	Таблицы могут оставаться заблокированными, если клиент временно приостановит выборку. Это не позволит другим клиентам получить доступ к этим таблицам
Элемент `max_length()` структур `MYSQL_FIELD` результирующего набора устанавливается равным длине наибольшего значения для столбца набора	Значение элемента `max_length()` не имеет никакого смысла, поскольку оно неизвестно до момента завершения полной выборки строк

- `my_ulonglong` **`mysql_affected_rows`** `(MYSQL *conn)`

  Возвращает число строк, измененных последним запросом DELETE, INSERT, REPLACE или UPDATE. Эту функцию для подобных запросов можно вызывать сразу после успешного вызова функции `mysql_query()`. Можно также вызвать функцию `mysql_affected_rows()` после выполнения оператора, возвращающего строки. В этом случае эта функция работает так же, как и функция `mysql_num_rows()`, и подвержена таким же ограничениям. Кроме того, существует еще одно ограничение, которое заключается в том, что возвращенное функцией `mysql_affected_rows()` значение не будет иметь смысла, если для выборки результата использовалась функция `mysql_use_result()`.

  Функция `mysql_affected_rows()` возвращает нуль, если ни один запрос не был отправлен либо если выполнен запрос, возвращающий строки, но ни одна из них не выбрана. Если возвращенное значение больше нуля, оно отражает число измененных (для запросов DELETE, INSERT, REPLACE и UPDATE) или возвращенных (для запросов, возвращающих данные) строк. Возвращенное значение -1 указывает на наличие ошибки или запуск функ-

ции mysql_affected_rows() после выполнения запроса, возвращающего строки, но до реальной выборки результирующего набора. Поскольку возвращаемое этой функцией значение не имеет знака, его анализ можно выполнить после предварительного преобразования результата в значение со знаком:

```
if ((long) mysql_affected_rows (conn) == -1)
    fprint (strderr, "Ошибка!\n");
```

Если пользователь определяет, что программа должна возвратить число строк, отвечающих условиям запроса UPDATE, функция mysql_affected_rows() возвращает именно это значение, а не число реально измененных строк. (MySQL не обновляет строку, если изменяемые столбцы включают значения, аналогичные новым.) Для выбора этого варианта необходимо либо определить параметр return_found_rows в конфигурационном файле, либо задать в параметре flags функции mysql_real_connect() параметр CLIENT_FOUND_ROWS.

Более детальную информацию о выводе значений типа my_ulonglong см. в разделе "Скалярные типы данных".

- void **mysql_data_seek** (MYSQL_RES *res_set, my_ulonglong row_num);

  Находит нужную строку в результирующем наборе. Значение аргумента row_num лежит в диапазоне от 0 до mysql_num_rows(res_set)-1. Если это значение выходит за пределы данного диапазона, возвращаемый результат окажется непредсказуемым.

  Для работы функции mysql_data_seek() необходимо, чтобы результирующий набор был выбран полностью. Поэтому данную функцию можно использовать только в том случае, если результирующий набор создан функцией mysql_store_result(), а не функцией mysql_use_result().

  Функция mysql_data_seek() отличается от mysql_row_seek() тем, что она принимает от функции mysql_row_tell() значение смещения, а не номер строки.

  До MySQL 3.23.7 аргумент row_num имел тип unsigned int.

- MYSQL_FIELD * **mysql_fetch_field** (MYSQL_RES *res_set);

  Возвращает структуру с информацией (метаданными) о столбце результирующего набора. При первом вызове функции mysql_fetch_field() после успешного выполнения запроса, возвращающего строки, возвращается информация о первом столбце. При каждом последующем вызове возвращается информация о последующих столбцах или значение NULL, если таких столбцов больше не осталось.

  Для определения позиции текущего столбца можно вызвать функцию mysql_field_tell(). Чтобы выбрать отдельный столбец, информация о котором будет возвращена при следующем вызове функции mysql_fetch_field(), воспользуйтесь функцией mysql_field_seek().

  В следующем примере отыскивается первая структура MYSQL_FIELD, а затем извлекаются все последующие структуры с информацией о столбцах:

```
MYSQL_FIELD *field;
unsigned inti;

mysql_field_seek (res_set, 0);
for (i = 0; i < mysql_num_fields (res_set); i++)
{
    field = mysql_fetch_field (res_set)
    printf ("column %u: name = %s max_length = %lu\n",
        i, field->name, field->max_length);
}
```

- `MYSQL_FIELD * mysql_fetch_fields (MYSQL_RES res_set);`

  Возвращает массив всех структур с информацией о столбцах результирующего набора. Доступ к ним можно получить следующим образом:

```
MYSQL_FIELD *field;
unsigned inti;

field = mysql_fetch_fields (res_set);
for (i = 0; i < mysql_num_fields (res_set); i++);
{
    printf ("column %u: name = %s max_length = %lu\n",
        i, field[i].name, field[i].max_length);
}
```

  Сравните этот код с примером, приведенным в описании функции `mysql_fetch_field()`. Обратите внимание на то, что хотя обе функции возвращают значения одинакового типа, доступ к таким значениям осуществляется с использованием разного синтаксиса. Функция `mysql_fetch_field()` возвращает указатель на структуру с одним полем; функция `mysql_fetch_fields()` возвращает указатель на массив структур полей.

- `MYSQL_FIELD * mysql_fetch_field_direct (MYSQL_RES *res_set, unsigned int col_num);`

  Считывая индекс столбца, возвращает для него информационную структуру. Значение аргумента `col_num` может задаваться в диапазоне от 0 до `mysql_num_fields(res_set)-1`. Если это значение выходит за пределы данного диапазона, возвращаемый результат окажется непредсказуемым.

  Как получать прямой доступ к структурам `MYSQL_FIELD`, показано в следующем примере:

```
MYSQL_FIELD *field;
unsigned inti;

for (i = 0; i < mysql_num_fields (res_set); i++)
{
    field = mysql_fetch_field_direct (res_set, i);
    printf ("column %u: name = %s max_length = %lu\n",
            i, field->name, field->max_length);
}
```

  Функция `mysql_fetch_field_direct()` не работает в версиях MySQL, предшествующих 3.23.

- `unsigned long * mysql_fetch_lengths (MYSQL_RES *res_set);`

Возвращает указатель на массив значений unsigned long, представляющих длины столбцов в текущей строке результирующего набора. Функцию mysql_fetch_lengths() *необходимо вызывать при каждом вызове функции* mysql_fetch_row(), иначе длины столбцов не будут синхронизированы с их значениями.

Для значений NULL длина равна нулю, однако нулевая длина сама по себе не является обязательным фактом наличия значения NULL. Поскольку длина пустой строки также равна нулю, пользователь еще должен проверить, является ли заданное значение столбца указателем NULL.

В следующем примере выводятся длины столбцов и значения текущей строки, причем на месте значения NULL выводится строка "NULL":

- ```
  unsigned long      *length;

  length = mysql_fetch_length (res_set);
  for (i = 0; i < mysql_num_fields (res_set); i++)
  {
     printf ("length is %lu, value is %s\n",
                     length[i], (row[i] != NULL ? row[i] : "NULL"));
  }
  ```

В версиях MySQL, предшествующих 3.22.7, возвращаемое ею значение имело тип unsigned int.

- MYSQL_ROW **mysql_fetch_row** (MYSQL_RES *res_set);

Возвращает указатель на следующую строку результирующего набора, представленного в виде массива строк (лишь значения столбцов NULL представляются в виде указателей на NULL). i-е значение строки является i-м элементом массива значений. Значение аргумента row_num лежит в диапазоне от 0 до mysql_num_rows(res_set)-1.

Значения всех типов данных, в том числе и числовых, представлены в виде строк. Поэтому перед выполнением какой-либо числовой операции над значением необходимо сначала его преобразовать, например с помощью функции atoi(), atof() или sscanf().

Если в результирующем наборе не остается строк, функция mysql_fetch_row() возвращает значение NULL. (Если для инициализации построчной выборки результирующего набора используется функция mysql_use_result(), функция mysql_fetch_row() возвращает значение NULL еще и в случае возникновения ошибки соединения.)

Информационные значения заканчиваются значением NULL, однако значения с двоичными данными следует обрабатывать не как строки, завершающиеся значением NULL, а как строки фиксированной длины. (Для определения длины значения столбца воспользуйтесь функцией mysql_fetch_length().)

В следующем примере показано, как организовать просмотр строк со значениями данных и определить, какие из значений равны NULL:

```
MYSQL_ROW    row;
unsigned inti;

while ((row = mysql_fetch_row (res_set)) != NULL)
```
```

```
{
    for (i = 0; i < mysql_num_fields (res_set); i++)
    {
        printf ("column %u: value is %s\n",
                i, (row[i] == NULL ? "NULL" : "not NULL"));
    }
}
```

Для определения типов значений столбцов можно использовать метаданные столбцов, записанные в информационных структурах столбцов MYSQL_FIELD. Как отмечалось ранее, доступ к ним можно получить с помощью функций mysql_fetch_field(), mysql_fetch_fields() или mysql_fetch_field_direct().

- unsigned int **mysql_field_count** (MYSQL *conn)

Возвращает число столбцов для последнего запроса заданного соединения. Эта функция обычно используется в тех случаях, когда функции mysql_store_result() или mysql_use_result() возвращают значение NULL. Функция mysql_field_count() позволяет определить, должен ли возвращаться результирующий набор. Возвращаемое ею значение 0 отражает отсутствие результирующего набора и соответственно отсутствие ошибки. Ненулевое значение показывает, что столбцы должны присутствовать и, поскольку они отсутствуют, имеет место ошибка.

В следующем примере показано, как использовать функцию mysql_field_count() для определения наличия ошибок:

```
res_set = mysql_store_result (conn);
if (res_set == NULL)     /* результирующий набор не был
                            возвращен */
{
    /*
    * Объясняется ли отсутствие результирующего
    * набора ошибкой, либо набор вовсе не должен возвращаться?
    */
    if (mysql_field_count (conn) > 0)
    {
        /*
         * результирующий набор ожидался, но функция
           mysql_store_result()
         не возвратила его, что означает наличие ошибки
        */
        printf ("При обработке результирующего набора произошла
                ошибка \n");
    }
    else
    {
        /*
         * результирующий набор не ожидался;
         * запрос не возвратил данные (это не был
         * запрос SELECT, SHOW, DESCRIBE или
         * EXPLAIN), поэтому запрос определяет
         * только число измененных строк
         */
        printf ("Обработано %lu строк \n"
                (unsigned long)mysql_affected_rows (conn));
    }
```

```
}
else    /* результирующий набор возвращен */
{
    /* ... обработка строк ... */
    mysql_free_result (res_set);
}
```

Функция `mysql_field_count()` впервые появилась в MySQL 3.22.24. До этой версии для подобных целей использовалась функция `mysql_num_fields()`. Для написания функции, которая бы работала независимо от версии MySQL, добавьте в любой файл, использующий функцию `mysql_field_count()`, следующий фрагмент кода:

```
#if !defined(MYSQL_VERSION_ID) || MYSQL_VERSION_ID<32224
#define mysql_field_count mysql_num_fields
#endif
```

Этот фрагмент заменяет функцию `mysql_field_count()` функцией `mysql_num_fields()` при компиляции исходного кода в более старых версиях MySQL.

- `MYSQL_FIELD_OFFSET` **`mysql_field_seek`** `(MYSQL_RES *res_set, MYSQL_FIELD_OFFSET offset);`

  Находит информационную структуру столбца, определенную аргументом `offset`. В результате при следующем вызове функции `mysql_fetch_field()` будет возвращена информационная структура для заданного столбца. Аргумент `offset` *не является* индексом столбца. Он представляет собой значение `MYSQL_FIELD_OFFSET`, полученное в результате более раннего вызова функции `mysql_field_tell()` или `mysql_field_seek()`.

  Для возвращения к первому столбцу задайте значение аргумента `offset` равным 0.

- `MYSQL_FIELD_OFFSET` **`mysql_field_tell`** `(MYSQL_RES *res_set);`

  Возвращает смещение информационной структуры текущего столбца. Это значение должно передаваться функции `mysql_field_seek()`.

- `void` **`mysql_free_result`** `(MYSQL_RES *res_set);`

  Освобождает память, занятую результирующим набором. Для каждого используемого результирующего набора вызывается функция `mysql_free_result`. Обычно результирующие наборы получаются в результате вызова функций `mysql_store_result()` или `mysql_use_result()`. Однако некоторые функции клиентской библиотеки генерируют результирующие наборы неявным образом, и разработчик программ также отвечает за освобождение этих наборов. Среди таких функций есть `mysql_list_dbs()`, `mysql_list_fields()`, `mysql_list_processes()` и `mysql_list_tables()`.

  Начиная с MySQL 3.22.3 функция `mysql_free_result()` автоматически выбирает и отменяет любые строки результирующего набора, сгенерированные функцией `mysql_use_result()`.

- `my_ulonglong` **`mysql_insert_id`** `(MYSQL conn)`

Возвращает значение AUTO_INCREMENT, сгенерированное последним выполняемым запросом данного соединения. Если запрос не был выполнен или в процессе выполнения не создал значения AUTO_INCREMENT, эта функция возвращает нулевое значение. Это означает, что функцию mysql_insert_id() следует вызывать сразу же после запроса, который должен сгенерировать новое значение. Если между выполнением этого запроса и запуском функции mysql_insert_id() будет выполнен промежуточный запрос, возвращаемое функцией значение будет сброшено. Если будет запущен другой запрос вызовом mysql_insert_id(), его значение будет переустановлено.

Нужно заметить, что функция mysql_insert_id() отличается от SQL-функции LAST_INSERT_ID(). Первая используется программой-клиентом и устанавливается для каждого запроса, в то время как значение SQL-функции хранится на сервере и одинаково для всех запросов, пока не будет создано новое значение AUTO_INCREMENT.

Возвращаемое функцией mysql_insert_id() значение относится к отдельному соединению и не связано с изменением значения AUTO_INCREMENT для других соединений. Как выводить значения типа my_ulonglong, см. в разделе "Скалярные типы данных".

- `unsigned int mysql_num_fields (MYSQL_RES *res_set);`

Возвращает число столбцов результирующего набора. Эта функция зачастую используется для определения числа столбцов текущей строки набора, что и показано в следующем примере:

```
MYSQL_ROW    row;
unsigned inti;

while ((row = mysql_fetch_row (res_set)) != NULL)
{
    for (i=0; i < mysql_num_fields (res_set); i++);
    {
        /* операции со значением row[i] ... */
    }
}
```

В версиях MySQL, предшествующих 3.22.24, функция mysql_num_fields() применялась также для выполнения действий, реализуемых в текущих версиях функцией mysql_field_count(). Другими словами, она проверяла, является ли возвращаемое функциями mysql_store_result() или mysql_use_result() значение NULL результатом ошибки. Именно поэтому в коде старых версий можно увидеть функцию mysql_num_fields(), вызываемую с указателем на дескриптор соединения, а не на результирующий набор. Раньше эту функцию можно было вызывать двумя способами. В текущих же версиях MySQL вызов функции mysql_num_fields() с дескриптором соединения не допускается. Для подобных целей следует написать программу с функцией mysql_field_count(). В представленном описании этой функции показывается, как использовать ее в старых версиях MySQL.

- my_longlong **mysql_num_rows** (MYSQL_RES *res_set)

  Возвращает число строк результирующего набора. Если данный результирующий набор был создан с помощью функции mysql_store_result(), функцию mysql_num_rows() можно затем вызвать в любой момент времени:

```
if ((res_set = mysql_store_result (conn)) == NULL)
{
    /* Теперь можно вызывать функцию mysql_num_rows() */
}
```

  Если для создания результирующего набора используется функция mysql_use_result(), функция mysql_num_rows() возвратит правильное значение только после выборки всех строк:

```
if ((res_set = mysql_use_result (conn)) == NULL)
{
    /* Функцию mysql_num_rows() нельзя еще вызывать */
    while ((row = mysql_fetch_row (res_set)) != NULL)
    {
        /* Функцию mysql_num_rows() нельзя еще вызывать */
    }
    /* Функцию mysql_num_rows() можно вызывать */
```

  Как выводить значения типа my_ulonglong, см. в разделе "Скалярные типы данных".

- MYSQL_ROW_OFFSET **mysql_row_seek** (MYSQL_RES *res_set, MYSQL_ROW_OFFSET offset);

  Находит определенную строку в результирующем наборе. Функция mysql_row_seek() подобна функции mysql_data_seek(), однако ее аргумент offset не является номером строки. В его качестве может выступать только значение, полученное в результате вызова функции mysql_row_tell() или mysql_row_seek().

  Функция mysql_row_seek() возвращает смещение предыдущей строки.

  Для работы функции mysql_row_seek() необходима полная выборка набора, что позволяет использовать ее только в том случае, когда результирующий набор создается функцией mysql_store_result(), но не mysql_use_result().

- MYSQL_ROW_OFFSET **mysql_row_tell** (MYSQL_RES *res_set)

  Возвращает смещение, отражающее позицию текущей строки в результирующем наборе. Это значение не является номером строки и может задаваться только функцией mysql_row_seek(), но не mysql_data_seek().

  Для работы функции mysql_row_tell() необходима полная выборка набора, что позволяет использовать ее только в том случае, когда результирующий набор создается функцией mysql_store_result(), но не mysql_use_result().

- MYSQL_RES * **mysql_store_result** (MYSQL *conn);

  Запускаемая после успешно выполненного запроса, эта функция возвращает результирующий набор и сохраняет его на компьютере пользователя. Если запрос вовсе не возвращает данные или при его выполнении произошла ошибка, функция mysql_store_result() возвращает значение NULL. При

получении такого значения вызовите функцию mysql_field_count() или одну из функций диагностики ошибок. Это позволит определить, является ли возвращаемое значение NULL следствием отсутствия результирующего набора или возникновения ошибки.

По завершении работы с результирующим набором задайте его в качестве аргумента функции mysql_free_result() для освобождения памяти.

Сравнение функций mysql_store_result() и mysql_use_result() представлено в табл. Е.6.

- MYSQL_RES **mysql_use_result** (MYSQL *conn);

  Запускаемая после успешно выполненного запроса, эта функция инициализирует выборку результирующего набора, но собственно строки с данными при этом не извлекает. Для последовательной выборки строк необходимо вызвать функцию mysql_fetch_row(). Если запрос вовсе не возвращает данных или при его выполнении возникает ошибка, функция mysql_use_result() возвращает значение NULL. При получении такого значения вызовите функцию mysql_field_count() или одну из функций, создающих отчеты об ошибках. Это позволит определить, является ли возвращаемое значение NULL следствием отсутствия результирующего набора или возникновения ошибки.

  По завершении работы с результирующим набором задайте его в качестве аргумента функции mysql_free_result() для освобождения памяти. Начиная с MySQL 3.22.3 этого достаточно для завершения обработки запросов, так как функция mysql_free_result() перед удалением из памяти результирующего набора автоматически ищет и отменяет все невыбранные строки. (До MySQL 3.22.3 необходимо выбирать строки самим до тех пор, пока ничего не останется, после чего вызвать функцию mysql_free_result(). В противном случае появится сообщение об ошибке "out of sync".)

  Сравнение функций mysql_store_result() и mysql_use_result() представлено в табл. Е.6.

  Обе функции, mysql_store_result() и mysql_use_result(), используются для выборки результирующих наборов, однако каждая из них определяет свой способ дальнейшей обработки наборов, а соответственно и функции такой обработки.

Информационные функции

Функции этого класса обеспечивают информацию о клиенте, сервере, версии протокола и текущем соединении. Значения, возвращаемые большинством из них, выбираются с сервера во время соединения и сохраняются в пределах клиентской библиотеки.

- const char * **mysql_character_set_name** (MYSQL *conn);

  Возвращает имя стандартной кодировки для заданного соединения, например "latin1". Функция mysql_character_set_name() была добавлена в MySQL 3.23.21.

- const char * **mysql_get_client_info** (void);

Возвращает строку, завершающуюся символом NULL, содержащую описание версии клиентской библиотеки, например "3.22.25".

- `const char * mysql_get_host_info (MYSQL *conn);`

  Возвращает строку, завершающуюся символом NULL, содержащую описание текущего соединения, например "Localhost via UNIX socket", "cobra.snake.net via TCP/IP" или ". via named pipe".

- `unsigned int mysql_get_proto_info (MYSQL *conn);`

  Возвращает номер версии протокола "клиент/сервер", используемого для клиентского соединения.

- `const char * mysql_get_server_info (MYSQL *conn)`

  Возвращает строку, завершающуюся символом NULL, содержащую описание версии сервера, например "4.0.2-alpha-log". За номером версии могут следовать суффиксы. Значения суффиксов см. в описании функции VERSION() в приложении В, "Операторы и функции".

- `const char * mysql_info (MYSQL *conn);`

  Возвращает строку, содержащую информацию о результате самого последнего выполненного запроса следующих типов. Диагностическая строка будет получена сразу же после каждого запроса.

```
ALTER TABLE ...
    Records: 0 Duplicates: 0 Warnings: 0
INSERT INTO ... SELECT ...
    Records: 0 Duplicates: 0 Warnings: 0
INSERT INTO ... VALUES (...),(...),...
    Records: 0 Duplicates: 0 Warnings: 0
LOAD DATA ...
    Records: 0 Deleted: 0 Skipped: 0 Warnings: 0
UPDATE ...
    Rows matched: 0 Changed: 0 Warnings: 0
```

  Числа могут варьироваться в зависимости от результата конкретного исполняемого запроса.

  Функция mysql_info() возвращает значение, отличное от "NULL", для INSERT INTO ... VALUES только в том случае, если оператор содержит больше одного списка значений. Для операторов, не представленных здесь, функция mysql_info() всегда возвращает значение "NULL".

  Строки, возвращаемые функцией mysql_info(), соответствуют синтаксису языка, используемого сервером. Таким образом, не стоит рассчитывать на то, что их синтаксис можно проанализировать простым просмотром определенных слов.

- `char * mysql_stat (MYSQL *conn);`

  Возвращает строку, завершающуюся символом NULL, содержащую описание информации о состоянии сервера, или значение NULL, если имеет место ошибка. Формат возвращаемой строки может меняться. В текущих версиях она выглядит примерно следующим образом:

```
Uptime: 189474 Threads: 4 Questions: 331869 Slow queries: 50
Opens: 1424 Flush tables: 1 Open tables: 64 Queries per second
avg: 0.279
```

Выводимые значения можно интерпретировать следующим образом.

- `Uptime`. Продолжительность работы сервера в секундах.
- `Threads`. Число работающих процессов сервера.
- `Questions`. Число выполненных сервером запросов.
- `Slow queries`. Число запросов, выполняемых дольше, чем задано параметром `long_query_time`.
- `Opens`. Общее число открытых сервером таблиц.
- `Flush tables`. Число выполненных команд FLUSH, REFRESH и RELOAD.
- `Open tables`. Число открытых в настоящее время таблиц.
- `Queries per second`. Отношение `Questions` к `Uptime`.

Функция `mysql_stat()` возвращает ту же информацию, что и команда `mysqladmin status`. (А где же еще команда `mysqladmin` берет информацию?)

- `unsigned long` **`mysql_thread_id`** `(MYSQL *conn);`

  Возвращает номер процесса, который сервер связывает с текущим соединением. Этот номер можно использовать как идентификатор для функции `mysql_kill()`.

  Функцию `mysql_thread_id()` рекомендуется запускать лишь в том случае, когда необходимо получить возвращаемое ею текущее значение. Если сохранить возвращенное значение, то через некоторое время оно может уже не соответствовать действительности. В частности, такая ситуация будет иметь место после разрыва и восстановления соединения (например, с помощью функции `mysql_ping()`), поскольку сервер присвоит новый номер процесса.

Административные функции

Описанные в этом разделе административные функции позволяют управлять различными аспектами функционирования сервера.

- `int` **`mysql_shutdown`** `(MYSQL *conn);`

  Указывает серверу завершить работу. Для использования этой функции пользователь должен обладать привилегией SHUTDOWN.

  Функция `mysql_shutdown()` возвращает нулевое значение в случае успешного выполнения операции и ненулевое значение в случае сбоя.

Функции поддержки процессовых клиентов

Описанные в этом разделе функции используются при работе многопроцессовых клиентов.

- `void` **`mysql_thread_end`** `(void);`

  Освобождает все переменные, связанные с процессом, которые были инициализированы функцией `mysql_thread_init()`. Для экономии памяти

эту функцию рекомендуется вызывать для завершения всех созданных вами процессов.

Функция `mysql_thread_end()` появилась в MySQL 4.0.0.

- `my_bool` **`mysql_thread_init`** `(void);`

  Инициализация всех переменных, связанных с процессом. Эта функция должна вызываться для каждого создаваемого процесса, который будет вызывать функции MySQL. Кроме того, при остановке процесса необходимо вызывать функцию `call mysql_thread_end()`.

  Функция `mysql_thread_init()` появилась в MySQL 4.0.0.

- `unsigned int` **`mysql_thread_safe`** `(void);`

  Возвращает значение, равное 1, если клиентская библиотека защищена от процессов, 0 — в противном случае. Значение, возвращаемое этой функцией, отражает настройку MySQL с параметром `--enable-thread-safe-client`.

  Функция `mysql_thread_safe()` появилась в MySQL 3.23.14.

Функции обмена данными со встроенным сервером

Этот раздел полностью посвящен программам, предназначенным для обмена данными `libmysqld`, встроенным сервером MySQL. Для того чтобы написать программу, использующую встроенный сервер, необходимо задать параметр `-lmysqld`, а не `-lmysqlclient`, при использовании своей программы для создания исполняемого кода. Для определения флагов, которые подходят для подключения встроенного сервера к создаваемой системе, воспользуйтесь следующей командой:

```
% mysql_config --libmysqld-libs
```

Обычная клиентская библиотека `-lmysqlclient` содержит пустые версии этих программ, которые не несут никакой функциональной нагрузки. Это значит, что вызывая эти функции, вполне возможно создать автономную клиентскую программу или такую программу, которая использует встроенный сервер, полученный из того же исходного кода, в зависимости от того, какая библиотека была выбрана на этапе редактирования связей.

- `void` **`mysql_server_end`** `(void);`

  Отключение встроенного сервера. Эта функция вызывается после прекращения использования сервера.

- `int` **`mysql_server_init`** `(int argc, char **argv, char **groups);`

  Инициализация встроенного сервера. Эта функция может использоваться перед вызовом любой функции `mysql_xxx()`.

  Аргументы `argc` и `argv` аналогичны стандартным аргументам, передаваемым `main()` в программах C (`argc` — это счетчик аргументов; если аргументы отсутствуют, `argc` должен быть равен 0). Иначе `argc` должен содержать количество аргументов, переданных на сервер. Аргумент `argv` является вектором строк, завершаемых символом NULL, в которых и содержатся аргументы. Заметим, что `argv[0]` игнорируется.

Аргумент groups является вектором строк с завершающим символом NULL, в которых обозначено, какие группы конфигурационных файлов будет читать встроенный сервер. Последним элементом вектора должен быть NULL. Если значение group равно NULL, сервер по умолчанию считывает группы конфигурационного файла [server] и [embedded]. Имена групп в векторе groups должны указываться без ограничивающих символов '[' и ']'.

Функции отладки

Описанные в этом разделе функции позволяют генерировать отладочную информацию на любом конце соединения как со стороны клиента, так и со стороны сервера. Однако для их использования необходимо, чтобы используемая версия MySQL была откомпилирована с поддержкой средств отладки. (Для этих целей при настройке дистрибутива MySQL используется параметр --with-debug или --with-debug=full, чтобы получить более полную информацию. Последний параметр инициирует библиотеку safemalloc, выполняющую расширенную проверку распределения памяти.)

- `void mysql_debug (const char *debug_string);`

  Выполняет операцию DBUG_PUSH с помощью строки debug_str. Формат этой строки описан в справочном руководстве по MySQL.

  Для использования функции mysql_debug() клиентская библиотека должна быть откомпилирована с поддержкой средств отладки.

- `int mysql_dump_debug_info (MYSQL *conn);`

  Указывает серверу записывать информацию отладки в журнал. Для использования этой функции до MySQL 4.0 пользователь должен обладать привилегией PROCESS, а после — SUPER.

  Функция mysql_dump_debug_info() возвращает нулевое значение в случае успешного выполнения операции и ненулевое значение при сбое.

Исключенные функции

Клиентская библиотека MySQL включает также множество функций, которые в настоящее время не используются, поскольку существует много других способов выполнить те же операции. Большинство из них можно заменить посредством задания соответствующего запроса функции mysql_query(). Так, например, функцию mysql_create_db("*db_name*") можно заменить следующей функцией:

```
mysql_query (conn, "CREATE DATABASE db_name")
```

С течением времени и по мере того, как MySQL распознавала все больше операторов SQL, некоторые функции оказывались ненужными и поэтому постепенно исключались из использования. Так, например, в результате добавления SQL-оператора FLUSH PRIVILEGES функция mysql_reload() стала абсолютно бесполезной. В представленных описаниях функций указывается версия MySQL, начиная с которой функция перестала использоваться. Ниже также описываются наиболее предпочтительные способы замены исключенных функций в текущих версиях. В клиентских библиотеках более ранних версий описываемую функцию, конечно же, можно ис-

пользовать. (Обязательно необходимо принимать во внимание и обновление версий. В общем, значительно безопаснее обновлять клиентские библиотеки MySQL, так как они будут работать как с новыми серверами, так и со старыми.)

Некоторые функции, такие, например, как `mysql_connect()`, `mysql_eof()` и `mysql_escape_string()`, больше не используются, поскольку заменены функциями с более мощными возможностями или возвращающими больший объем информации.

Планируя долгую и плодотворную работу в будущем, пользователям следует избегать описанных функций. Некоторые или все из них постепенно будут исключаться из арсенала СУБД MySQL.

- `MYSQL * mysql_connect (MYSQL *conn, const char *host_name, const char *user_name, const char *password);`

  Эта функция является "предшественником" функции `mysql_real_connect()`. В настоящее время функция `mysql_connect()` реализована в виде варианта вызова функции `mysql_real_connect()`.

  Эта функция исключена начиная с MySQL 3.22.0.

- `int mysql_create_db (MYSQL *conn, const char *db_name);`

  Создает базу данных с именем, определяемым аргументом `db_name`. В настоящее время эту же операцию можно выполнить, задав оператор `CREATE DATABASE` функции `mysql_query()`.

  Функция `mysql_create_db()` возвращает нулевое значение в случае успешного выполнения операции и ненулевое значение в случае сбоя.

  Эта функция исключена начиная с MySQL 3.21.15.

- `int mysql_drop_db (MYSQL *conn, const char *db_name);`

  Удаляет базу данных `db_name`. В настоящее время эту же операцию можно выполнить, задав оператор `DROP DATABASE` в функции `mysql_query()`.

  Функция `mysql_drop_db()` возвращает нулевое значение в случае успешного выполнения операции и ненулевое значение в случае сбоя.

  Эта функция исключена начиная с MySQL 3.21.15.

- `my_bool mysql_eof (MYSQL_RES *res_set);`

  Возвращает ненулевое значение при достижении конца результирующего набора и нулевое значение при появлении ошибки. Эта функция особенно полезна, если для инициализации процесса выборки результирующего набора используется функция `mysql_use_result()`, а для последовательной выборки строк — функция `mysql_fetch_row()`. В этом случае возвращаемое функцией `mysql_fetch_row()` значение `NULL` может отражать либо достижение конца набора, либо возникновение ошибки. Определить реальную причину и позволяет функция `mysql_eof()`.

  Для достижения аналогичного эффекта в настоящее время можно использовать функции `mysql_errno()` и `mysql_error()`, хотя в действительности они возвращают гораздо больше информации. (Эти функции указывают также возможную причину возникновения ошибки, а не просто подтверждают факт ее наличия.)

Эта функция исключена начиная с MySQL 3.21.17.

- `unsigned long mysql_escape_string (char *to_str, const char *from_str, unsigned long from_len);`

  Кодирует строку, содержащую специальные символы, которые можно использовать в операторе SQL. Функция `mysql_escape_string()` равносильна функции `mysql_real_escape_string()` с той лишь разницей, что она не принимает аргумента `conn`, обозначающего текущее соединение, и не учитывает текущую кодировку при выполнении кодирования. Подробнее об этом можно узнать в описании функции `mysql_real_escape_string()`.

  Начиная с MySQL 3.23.14 функция `mysql_escape_string()` была исключена в пользу функции `mysql_real_escape_string()`.

  До MySQL 3.23.6 аргумент `from_len` имел тип `unsigned int`.

- `int mysql_kill (MYSQL *conn, unsigned long thread_id);`

  Завершает процесс сервера `thread_id` и возвращает нуль при успешном завершении или ненулевое значение при неуспешном завершении. Вы всегда можете завершить свои собственные процессы. Для завершения процессов, созданных другими пользователями, необходимо обладать привилегией `PROCESS` (до MySQL 4.0.2) и привилегией `SUPER` (начиная с 4.0.2).

  Аналогичного эффекта можно добиться с помощью оператора `KILL` и функции `mysql_query()`.

  Функция `mysql_kill()` была исключена в MySQL 3.22.9.

- `MYSQL_RES * mysql_list_dbs (MYSQL *conn, const char *wild);`

  Возвращает результирующий набор со списком имен баз данных сервера либо значение `NULL`, если имела место ошибка. Возвращаемый список включает имена баз данных, соответствующих заданному аргументом `wild` регулярному выражению SQL (которое может содержать специальные символы "`%`" и "`_`"), или имена всех баз данных, если аргумент `wild` имеет значение `NULL`. Для освобождения памяти, занимаемой результирующим набором, пользователь впоследствии должен вызвать функцию `mysql_free()`.

  Возвращаемый функцией `mysql_list_dbs()` список можно также получить, задав команду `SHOW DATABASES` в качестве аргумента функции `mysql_query()` и обработав результирующий набор.

  Эта функция исключена начиная с MySQL 3.22.0.

- `MYSQL_RES * mysql_list_fields (MYSQL *conn, const char *tbl_name, char *wild);`

  Возвращает результирующий набор со списком имен столбцов указанной таблицы либо значение `NULL`, если имела место ошибка. Возвращаемый список включает имена столбцов, соответствующих заданному аргументом `wild` регулярному выражению SQL (которое может содержать специальные символы "`%`" и "`_`"), или имена всех столбцов, если аргумент `wild` имеет значение `NULL`. Для освобождения памяти, занимаемой результирующим набором, пользователь впоследствии должен вызвать функцию `mysql_free()`.

Возвращаемый функцией mysql_list_fields() список можно также получить, задав команду SHOW COLUMNS в качестве аргумента функции mysql_query() и обработав результирующий набор.

Эта функция исключена начиная с MySQL 3.22.0.

- MYSQL_RES * **mysql_list_processes** (MYSQL *conn);

  Возвращает результирующий набор со списком всех работающих на сервере процессов либо значение NULL, если произошла ошибка. Если задающий эту функцию пользователь обладает привилегией PROCESS, список будет включать все процессы сервера. В любом другом случае в список попадут только собственные процессы пользователя. Для освобождения памяти, занимаемой результирующим набором, пользователь впоследствии должен вызвать функцию mysql_free().

  Возвращаемый функцией mysql_list_processes() список можно также получить, задав команду SHOW PROCESSLIST в качестве аргумента функции mysql_query() и обработав результирующий набор.

  Эта функция исключена начиная с MySQL 3.22.0.

- MYSQL_RES * **mysql_list_tables** (MYSQL *conn, char *wild);

  Возвращает результирующий набор со списком таблиц текущей базы данных или значение NULL, если произошла ошибка. Возвращаемый список включает имена таблиц, соответствующих заданному аргументом wild регулярному выражению SQL (которое может содержать специальные символы "%" и "_"), или имена всех таблиц, если аргумент wild имеет значение NULL. Для освобождения памяти, занимаемой результирующим набором, пользователь впоследствии должен вызвать функцию mysql_free().

  Возвращаемый функцией mysql_list_tables() список можно также получить, задав команду SHOW TABLES в качестве аргумента функции mysql_query() и обработав результирующий набор.

  Эта функция исключена начиная с MySQL 3.22.0.

- int **mysql_refresh** (MYSQL *conn, unsigned int options);

  По своему действию эта функция аналогична операторам SQL FLUSH и RESET, но она позволяет сбрасывать одновременно несколько параметров. Функция mysql_refresh() возвращает нуль при успешном завершении и ненулевое значение в противном случае.

  Значение параметров может состоять из одного или нескольких значений, приведенных с нижеследующем списке. Для того чтобы иметь возможность выполнить эти операции, необходимо обладать привилегией RELOAD.

  - REFRESH_GRANT. Перезагружает содержимое таблиц разрешения. Этот параметр эквивалентен оператору FLUSH PRIVILEGES.
  - REFRESH_HOSTS. Обновляет кэш узла. Этот параметр эквивалентен оператору FLUSH HOSTS.
  - REFRESH_LOG. Сбрасывает файлы регистрации и открывает их вновь. Этот параметр имеет отношение к любому журналу и эквивалентен оператору FLUSH LOGS.

- REFRESH_MASTER. Указывает главному серверу репликации удалить двоичные журналы регистрации, перечисленные в индексном файле двоичных журналов регистрации, кроме того, он используется для усечения индекса. Этот параметр эквивалентен оператору RESET MASTER.

  Он был добавлен в MySQL 3.23.19.

- REFRESH_SLAVE. Указывает подчиненному серверу репликации "забыть" свое положение в регистрационном журнале главного сервера. Этот параметр эквивалентен оператору RESET SLAVE.

  Он был добавлен в MySQL 3.23.19.

- REFRESH_STATUS. Инициализирует переменные состояния в 0. Этот параметр эквивалентен оператору FLUSH STATUS.

  Он был добавлен в MySQL 3.22.11.

- REFRESH_TABLES. Закрывает открытые таблицы. Этот параметр эквивалентен оператору FLUSH TABLES.

- REFRESH_THREADS. Обновляет кэш процесса.

  Этот параметр был добавлен в MySQL 3.23.16.

Эти параметрические флаги представляют собой битовые значения, поэтому их можно объединять с помощью операторов "|" или "+". Например, следующие выражения являются эквивалентными:

```
REFRESH_LOG | REFRESH_TABLES
REFRESH_LOG + REFRESH_TABLES
```

Функция mysql_refresh() была исключена, так как она дублирует действия операторов FLUSH или RESET. Подробнее об этих операторах см. в приложении Д, "Программы MySQL".

- `int mysql_reload (MYSQL *conn);`

  Указывает серверу перезагрузить таблицу разрешений. В текущих версиях эту же операцию можно выполнить, задав оператор FLUSH PRIVILEGES функции mysql_query(). Для использования функции mysql_reload() пользователь должен обладать привилегией RELOAD.

  Функция mysql_reload() возвращает нулевое значение в случае успешного выполнения операции и ненулевое значение в случае сбоя.

  Эта функция исключена начиная с MySQL 3.21.9.

Ж

Программный интерфейс DBI языка написания сценариев Perl

В этом приложении...

Написание сценариев
Методы интерфейса DBI
Функции программ интерфейса DBI
Атрибуты интерфейса DBI
Переменные среды интерфейса DBI

Это приложение посвящено описанию программного интерфейса DBI языка написания сценариев Perl. Данный интерфейс состоит из набора методов и атрибутов, используемых для взаимодействия с серверами баз данных и получения доступа к базам данных из сценариев Perl. Кроме того, в этом же приложении рассматриваются MySQL-расширения интерфейса DBI, обеспечиваемые драйвером баз данных DBD::mysql. Здесь не упоминается интерфейс `Mysqlperl`, который уже считается устаревшим.

Некоторые методы и атрибуты интерфейса DBI в последующем материале не рассматриваются, поскольку либо не применяются в MySQL, либо являются новыми, экспериментальными методами, которые находятся еще на стадии разработки и могут в будущем значительно видоизмениться или вовсе исчезнуть. Не описываются в этом приложении и некоторые устаревшие методы DBD. Чтобы получить более детальную информацию о новых или, наоборот, устаревших методах, обратитесь к справочному руководству по интерфейсу DBI либо к документации по MySQL в части, посвященной интерфейсу DBI, которую можно получить с помощью следующих команд:

```
% perldoc DBI
% perldoc DBI::FAQ
% preldoc DBD::mysql
```

Это приложение рекомендуется использовать как справочное руководство, поскольку оно включает краткие фрагменты кода, наглядно иллюстрирующие использование интерфейса DBI языка Perl. Полностью завершенные клиентские сценарии, а также замечания по их написанию см. в главе 7, "Программный интерфейс Perl API."

Написание сценариев

Каждый сценарий Perl, использующий модуль интерфейса DBI, должен включать следующую строку:

```
use DBI;
```

Добавлять строку `use` для отдельного модуля уровня DBD необязательно, поскольку интерфейс DBI автоматически активизирует соответствующий модуль при подключении к серверу.

Как правило, сценарий интерфейса DBI устанавливает соединение с помощью метода `connect()`, после чего отрабатывает запросы и закрывает соединение с помощью метода `disconnect()`, хотя методы, применяемые для исполнения запросов, могут изменяться. Так, например, запросы, не использующие `SELECT`, обычно выполняются с помощью метода `do()`. Запросы `SELECT` исполняются посредством передачи запроса методу `prepare()`, вызова метода `execute()` и построчной выборки результатов в цикле с помощью таких методов выборки строк, как `fetchrow_array()` или `fetchrow_hashref()`.

Строки запросов в DBI-сценарии должны состоять из одного оператора SQL, не завершающегося точкой с запятой "`;`" или последовательностью "`\g`". Это соглашение действует для клиентской программы `mysql` и не действительно для интерфейса DBI.

Методы интерфейса DBI

Предлагаемые в этом приложении описания методов представлены в несколько ином формате, чем при описании функций языка C в приложении E, "Программный интерфейс приложений C", и PHP-функций в приложении З, "Программный интерфейс языка PHP". В данном приложении функции записываются в форме прототипа с явным указанием типов возвращаемых значений и параметров. При этом тип значения задается первым символом имени переменной. Символ "$" отражает скалярный тип, "@" — массив, "%" — хэш (ассоциативный массив). Кроме того, любой параметр, предваряемый символом "\", необходимо определять как ссылку на переменную, а не собственно переменную. Имя переменной, заканчивающееся суффиксом "`ref`", показывает, что значением переменной является ссылка.

В последующем материале некоторые переменные имеют специальное значение. Все они описаны в табл. Ж.1.

Таблица Ж.1. Специальные имена переменных Perl DBI

Имя	Значение
`$drh`	Дескриптор объекта драйвера
`$dbh`	Дескриптор объекта базы данных
`$sth`	Дескриптор оператора (запроса)
`$fh`	Дескриптор открытого файла
`$h`	"Общий" дескриптор (значение зависит от контекста задания)
`$rc`	Код возврата операций, возвращающих значение "True" или "False"
`$rv`	Возвращаемое значение выполнения операций, являющееся целым числом
`$rows`	Возвращаемое значение операций, представляющее количество строк
`$str`	Возвращаемое значение для операций, возвращающих строки
`@ary`	Массив (список), представляющий перечень значений
`@row_ary`	Массив (список), представляющий возвращенную запросом строку значений

Многие описанные далее методы успешно считывают аргумент `%attr` с атрибутами, воздействующими на работу метода. Задавать этот аргумент необходимо посредством ссылки, что можно реализовать двумя способами.

- Сначала определить содержимое атрибута `%attr`, а затем передать его методу:

```
my (%attr) = (AttrName1 => value1, AttrName2 => value2);
$ret_val = $h->method (..., \%attr);
```

- Задать хэш непосредственно при вызове метода:

```
$ret_val = $h->method (..., {Attrname1 => value1, AttrName2 =>
value2});
```

Способ использования метода или функции определяется последовательностью вызова. Так, последовательность "`DBI->`" отражает метод класса интерфейса DBI, "`DBI::`" — функцию DBI, а "`$DBI::`" — переменную DBI. Если метод вызывается с помощью дескриптора, имя этого дескриптора отражает область применения метода. Например, дескриптор "`$dbh->`" указывает на метод дескриптора базы данных, "`$sth->`" — на метод дескриптора оператора, а "`$h->`" — на метод, который можно вызывать с разными дескрипторами. Необязательная информация задается в квадратных скобках (`[]`). Рассмотрим, например, последовательность вызова:

```
@row_ary = $dbh->selectrow_array ($statement, [\%attr [, @bind_values]]);
```

Как видно, метод `selectrow_array()` определяется как метод дескриптора базы данных, поскольку вызывается с помощью дескриптора "`$dbh->`". В качестве параметров его вызова выступают аргументы `$statement` (скалярное значение), `%attr` (хэш, который необходимо задавать как ссылку, что определяется первым символом "`\`") и `@bind_values`. Второй и третий параметры являются необязательными. В качестве возвращаемого значения выступает массив.

В описании метода определяется также, какое значение возвращается в случае возникновения ошибки, хотя сама ошибка возвращается только при отключении атрибута `RaiseError`. Если этот атрибут активизирован, сценарий автоматически завершит свое выполнение. В последующих описаниях под термином "запрос

SELECT" следует понимать как собственно запрос SELECT, так и любой другой, возвращающий строки запрос, например DESCRIBE, EXPLAIN или SHOW.

Методы класса DBI

Для описанных в этом разделе методов атрибут %attr может использоваться с целью определения атрибутов обработки методов. (Отсутствие атрибута или undef означает "нет атрибутов.") Для MySQL важнейшими являются атрибуты PrintError, RaiseError и AutoCommit. Атрибуты, переданные методам connect() или connect_cached(), входят в состав возвращаемого дескриптора базы данных, возвращаемого этими методами. Например, чтобы определить автоматическое завершение сценария в случае возникновения ошибки DBI, достаточно активизировать атрибут RaiseError:

```
$dbh = DBI->connect ($data_source, $user_name, $password, {RaiseError
=> 1});
```

Атрибуты PrintError, RaiseError и AutoCommit рассматриваются ниже, в разделе "Атрибуты интерфейса DBI".

- @ary = DBI->**available_drivers** ([$quiet]); Возвращает список доступных драйверов интерфейса DBI. Если этот метод находит несколько драйверов с одинаковыми именами и значение параметра $quiet равно 0, то на экран выводится предупреждение. Для отмены вывода предупреждения установите значение параметра $quiet равным 1.
- $dbh = DBI->**connect** ($data_source, $user_namem $password [, \%attr]); Метод connect() устанавливает соединение с сервером и возвращает дескриптор базы данных. Если попытка установления соединения не была успешной, возвращается значение undef. Для завершения установленного ранее соединения необходимо вызвать метод disconnect() с дескриптором, возвращенным ранее методом connect().

  ```
  $dbh = DBI->connect ("DBI:mysql:sampdb:cobra.snake.net",
              "sampadm", "secret", \%attr)
              or die "Соединение невозможно \n";
  $dbh->disconnect ();
  ```

  Источник данных можно задать в нескольких форматах. Первая часть его всегда задается как "DBI:mysql", где имя интерфейса DBI можно задавать символами любого регистра, а имя драйвера "mysql" обязательно должно вводиться строчными буквами. Остальная часть имени источника данных преобразуется драйвером, поэтому представленный в последующем описании синтаксис применять для всех драйверов необязательно.

  После имени драйвера в начальной части строки названия источника данных можно также задать имя базы данных и узла:

  ```
  $data_source = "DBI:mysql:db_name";
  $data_source = "DBI:mysql:db_name:host_name";
  ```

  Базу данных можно задать с помощью параметра *db_name* или предложения database=*db_name*. Узел может быть задан как *host_name* или host=*host_name*.

За первоначальной частью строки, задающей источник данных, можно задать несколько параметров в формате `attribute=value`. Перед каждым параметром необходимо поставить точку с запятой.

```
DBI:mysql:sampdb:localhost;mysql_socket=/tmp/mysql.sock;mysql_comp
ression=1
```

Драйвер MySQL распознает следующие параметры.

- `host=host_name`. Компьютер, с которым устанавливается соединение. С помощью формата `host_name:port_name` можно также определить номер порта для TCP/IP-соединений.

  В системах UNIX при подключении к компьютеру `localhost` по умолчанию используются доменные разъемы UNIX. (В этом случае для определения имени разъема используется `mysql_socket`.) При желании можно подключаться к локальному узлу с использованием протокола TCP/IP. Для этого можно воспользоваться `host=127.0.0.1`.

  В системах Windows NT подключение к узлу '.' означает подключение к локальному серверу через именованный канал или протокол TCP/IP. (В таком случае для определения имени канала можно использовать значение `mysql_socket`.)

- `port=port_num`. Номер порта, через который устанавливается соединение. Этот параметр игнорируется для соединений, отличных от TCP/IP (например, соединений по адресу `localhost` под ОС UNIX).

- `mysql_client_found_rows=val`. Для запросов `UPDATE` сервер MySQL по умолчанию возвращает количество задействованных строк (модифицированных), а не количество строк, соответствующих запросу. Присвоение параметру `mysql_client_found_rows` значения, равного 1, позволит возвращать количество соответствующих строк. При значении 0 сохраняется стандартное поведение.

  Параметр `mysql_client_found_rows` был добавлен в DBD::mysql 1.2208.

- `mysql_compression=1`. Этот параметр активизирует средства сжатия при обмене между клиентом и сервером MySQL.

  Параметр `mysql_compression` был добавлен в DBD::mysql 1.1920.

- `mysql_connect_timeout=seconds`. Время ожидания соединения (в секундах) до истечения тайм-аута и возвращения ошибки.

  Параметр `mysql_connect_timeout` был добавлен в DBD::mysql 1.2207.

- `mysql_local_infile=val`. Этот параметр используется начиная с MySQL 3.23.49, чтобы обеспечить возможность включения параметра `LOCAL` для оператора `LOAD DATA`. Установка этого параметра в 1 активизирует режим `LOCAL`, если он был отключен в клиентской библиотеке MySQL по умолчанию (если сервер не был настроен таким образом, чтобы его отключить). Установка этого параметра в 0 отключает режим `LOCAL`, если он был активизирован в клиентской библиотеке.

  Параметр `mysql_local_infile` был добавлен в DBD::mysql 2.1020.

- `mysql_read_default_file=file_name`. По умолчанию сценарий DBI не считывает конфигурационные файлы MySQL и не использует записан-

ные в них значения параметров MySQL. Используя этот параметр, можно задать подобное считывание параметров и определить считываемый конфигурационный файл. Имя файла должно содержать полный путь к файлу. (Иначе его путь будет рассматриваться относительно каталога, в котором запускается сценарий, что может привести к довольно неожиданным результатам.)

Чтобы задать считывание файла `.my.cnf` из корневого каталога при каждом запуске сценария, задайте имя файла `$ENV{HOME}/.my.cnf`. Эта возможность оказывается исключительно эффективной, если ожидается, что сценарий будет использоваться несколькими пользователями, каждый из которых будет устанавливать соединение от своего имени.

Задание имени конфигурационного файла, который включает букву устройства, под управлением Windows невозможно, так как символ ":", разделяющий букву, обозначающую устройство, и следующий за ним путь используются также интерфейсом DBI в качестве разделителя строки источника данных. Самый грубый обходной метод решения этой проблемы состоит в том, чтобы задать имя пути без буквы устройства, а затем команду `chdir()` к этому устройству перед вызовом метода `connect()`. Для того чтобы текущий каталог сохранился после операции, перед вызовом метода `connect()` его необходимо сохранить, а после операции вернуться в этот каталог с помощью команды `chdir()`.

Параметр `mysql_read_default_file` работает только начиная с версии MySQL 3.22.10 и версии DBD::mysql 1.21.06 и выше.

- `mysql_read_default_group=`*`group_name`*. При считывании конфигурационного файла для определения значений параметров соединения по умолчанию просматривается только группа `[client]`. С помощью параметра `mysql_read_default_group` можно задать еще одну дополнительную группу. Параметр `mysql_read_default_group=dbi`, например, определяет, что считываться вместе с параметрами группы `[client]` будут и параметры группы `[dbi]`. Формат конфигурационных файлов см. в приложении E, "Программный интерфейс приложений C".

  Параметр `mysql_read_default_group` работает начиная с версии MySQL 3.22.10 и версии DBD::mysql 1.2106 и выше.

- `mysql_socket=`*`socket_name`*. Этот параметр определяет путь к разъему домена UNIX, используемого для подключения к компьютеру `localhost` (если установленный по умолчанию путь к разъему не подходит). Под ОС Windows им обозначается имя именованного канала. Этот параметр игнорируется при TCP/IP-соединениях (например, при соединениях с узлами, отличными от `localhost` на UNIX).

  Параметр `mysql_socket` работает начиная с версии MySQL 3.22.15.

- `mysql_ssl=1`
  `mysql_ssl_ca_file=`*`file_name`*
  `mysql_ssl_ca_path=`*`dir_name`*
  `mysql_ssl_cipher=`*`str`*
  `mysql_ssl_client_cert=`*`file_name`*
  `mysql_ssl_client_key=`*`file_name`*

Эти параметры применяются для установления безопасного соединения с сервером с использованием протокола SSL. Установка параметра `mysql_ssl` в 1 разрешает использование SSL. При этом другие параметры используются для определения других характеристик соединения; их значения соответствуют аналогичным аргументам функции `mysql_ssl_set()` интерфейса C API. Подробнее о них см. в приложении E, "Программный интерфейс приложений C". Активизировав параметр `mysql_ssl`, необходимо задать значения для параметров `mysql_ssl_ca_file`, `mysql_ssl_client_cert` и `mysql_ssl_client_key`.

Эти параметры действуют с MySQL 4 и могут использоваться для DBD::mysql 2.1013.

Для определения параметров соединения интерфейс DBI просматривает следующие переменные.

- Если источник данных не определен или пуст, используется значение переменной `DBI_DSN`.
- Если в источнике данных отсутствует имя драйвера, используется значение переменной `DBI_DRIVER`.
- Если при вызове метода `connect()` не определены параметры *`user_name`* или *`password`*, используются значения переменных `DBI_USER` и `DBI_PASS` соответственно. Этого однако *не* происходит, если значениями параметров являются пустые строки.

  (Использование переменной `DBI_PASS` несет в себе определенный риск. Именно поэтому от нее следует отказаться, если требования повышенной безопасности являются наиболее приоритетными. Используя такие команды, как `ps`, одни пользователи могут просматривать значения переменных других.)

Даже если после исследования всех источников информации некоторые параметры соединения остаются неизвестными, интерфейс DBI использует параметры, установленные по умолчанию. Так, например, если не определено имя компьютера, по умолчанию выбирается имя `localhost`. Если же неопределенным осталось имя пользователя, в его качестве по умолчанию используется регистрационное имя в UNIX и ODBC в Windows. Значение по умолчанию пароля не устанавливается, поэтому если пароль не определен, его не нужно задавать.

- `$dbh = DBI->connect_cached ($data_source, $user_name, $password [, \%attr]);` Этот метод аналогичен методу `connect()`, за исключением того, что DBI-интерфейс внутренне кэширует дескриптор базы данных. Если делается последующий вызов `connect_cached()` с теми же параметрами соединения в то время, как соединение по-прежнему активно, DBI-интерфейс возвращает кэшированный дескриптор, не открывая при этом нового соединения. Если кэшированный дескриптор больше не действует, DBI-интерфейс устанавливает новое соединение, а потом кэширует и возвращает новый дескриптор.
- `@ary = DBI->data_sources ($driver [, \%attr]);` Возвращает список баз данных, которые доступны через указанный драйвер. Для MySQL

значение $driver должно быть mysql (в нижнем регистре). Если $driver имеет значение undef или пустую строку, DBI-интерфейс проверяет значение переменной среды DBI_DRIVER, чтобы получить имя драйвера.

Для большинства драйверов DBI-интерфейса data_sources() возвращает пустой или неполный список. В частности, для MySQL эта функция ничего не возвращает, если драйвер может подключиться к серверу на локальном узле без имени или пароля. Это вряд ли возможно, за исключением незащищенных серверных конфигураций.

- $drh = DBI->**install_driver** ($driver_name); Активизирует драйвер уровня DBD и возвращает для него дескриптор драйвера. Для MySQL используется имя драйвера "mysql" (которое должно быть записано строчными буквами). Необходимость в вызове этого метода зачастую отсутствует, поскольку интерфейс DBI активизирует соответствующий драйвер автоматически при вызове метода connect(). Хотя метод install_driver() может оказаться весьма полезным при выполнении административных операций с помощью метода func(). (Подробнее об этом — в разделе "Специальные административные методы MySQL".)

Методы дескрипторов баз данных

Описанные в этом разделе методы вызываются с помощью дескрипторов баз данных и используются после получения дескриптора с помощью методов connect() или connect_cached().

Атрибуты обработки этих методов могут задаваться в параметре %attr. (Значение атрибута undef означает "нет атрибута.") Как уже отмечалось выше, для MySQL наиболее важными являются атрибуты PrintError и RaiseError. Например, можно активизировать атрибут RaiseError для автоматического завершения выполнения сценария при возникновении ошибки DBI в процессе обработки запроса:

```
$rows = $dbh->do ($statement, {RaiseError => 1});
```

Атрибуты PrintError и RaiseError детально рассматриваются ниже, в разделе "Атрибуты интерфейса DBI".

- $rc = $dbh->**begin_work** (); Отключает режим автовыполнения отключением атрибута дескриптора базы данных AutoCommit. Это позволяет выполнять операторы, объединяющие несколько операторов. Режим автовыполнения остается отключенным до следующего вызова метода commit() или rollback(), после чего он активизируется снова. Использование метода begin_work() отличается от отключения метода автовыполнения вручную; после выполнения или отката транзакции нужно возобновить работу режима автовыполнения вручную.

  Метод begin_work() возвращает значение "true", если режим автовыполнения был успешно отключен, и возвращает значение "false", если он был отключен ранее. Если используемая версия MySQL не поддерживает транзакции, попытка отключения режима автовыполнения возвратит фатальную ошибку.

Функция `begin_work()` была добавлена в DBI 1.20. Поддержка транзакций требует DBD::mysql 1.2216 или выше и MySQL 3.23.17 или выше.

- `$rc = $dbh->commit ();` Выполнение текущей транзакции, если MySQL поддерживает транзакции и режим автовыполнения отключен. Если режим автовыполнения активизирован, вызов метода `commit()` не возымеет никакого действия и приводит к выводу предупреждения.

  Поддержка транзакций требует DBD::mysql 1.2216 или выше и MySQL 3.23.17 или выше.

- `$rc = $dbh->disconnect ();` Обрывает соединение, связанное с заданным дескриптором. Если по завершении работы сценария соединение все еще активно, на экран выводится предупреждение и соединение автоматически обрывается.

  Поведение метода `disconnect()` по отношению к активным транзакциям можно назвать неопределенным. Если транзакция активна, ее можно прервать явным образом, вызвав перед методом `disconnect()` метод `commit()` или метод `rollback()`.

- `$rows = $dbh->do ($statement [, \%attr [, @bind_values]]);` Подготавливает и выполняет запрос, заданный параметром `$statement`. Этот метод возвращает либо число измененных строк, либо значение -1, если число строк неизвестно, либо значение `undef`, если в процессе выполнения произошла ошибка. Если ни одна строка изменена не была, т.е. число измененных строк равно 0, возвращается строка "0E0". Это значение также равно 0, однако в контексте данного метода рассматривается как истинное.

  Метод `do()` используется чаще всего для операторов, которые не производят выборки строк, как например, `DELETE`, `INSERT`, `REPLACE` или `UPDATE`. Если попытаться применить его для оператора `SELECT`, дескриптор этого оператора не будет возвращен, что не позволит произвести выборку строк.

  Обычно методу `do()` параметры не передаются, поэтому параметр `%attr` может быть задан как `undef`. Параметр `@bind_values` используется в тех случаях, когда оператор содержит маркеры (выделенные внутри строки запроса символом "?"). Этот параметр содержит список значений, которые связаны с маркерами. Количество значений списка должно точно соответствовать числу маркеров.

  Если запрос не содержит маркеров, можно не задавать как параметр `%attr`, так и список значений:

```
$rows = $dbh->do ("UPDATE member SET expiration = NOW() WHERE
member_id = 39");
```

  Если запрос содержит маркеры, список должен содержать столько значений, сколько есть маркеров, и должен предваряться аргументом `%attr`.

  В следующем примере аргумент атрибута имеет значение `undef`, и за ним следуют два значения, которые привязываются к двум маркерам в строке запроса:

```
$rows = $dbh->do ("UPDATE member SET expiration = ? WHERE
member_id = ?",
                  undef,
                  "2005-11-30", 39);
```

- `$rc = $dbh->ping ();` Проверяет, активно ли еще соединение с сервером, и возвращает соответственно истинное значение, если соединение по-прежнему активно или было успешно восстановлено, или ложное значение.
- `$sth = $dbh->prepare ($statement [, \%attr]);` Подготавливает запрос, определенный параметром `$statement`, для последующего выполнения и возвращает дескриптор запроса. В дальнейшем этот дескриптор может использоваться для выполнения запроса с помощью метода `execute()`.
- `$sth = $dbh->prepare_cached ($statement [, \%attr [, $allow_active]]);` Этот метод аналогичен методу `prepare()`, за исключением того, что DBI-интерфейс кэширует дескриптор оператора внутренне. Если последующий вызов метода `prepare_cached()` производится с теми же самыми аргументами `$statement` и `%attr`, DBI-интерфейс возвращает кэшированный дескриптор, а не новый. Если кэшированный дескриптор активен, DBI-интерфейс вызывает метод `finish()`, выдает предупреждение перед возвращением дескриптора и передает значение для `$allow_active`.
- `$str = $dbh->quote ($value [, $data_type]);` Обрабатывает строку, заключая в кавычки и выделяя отдельные символы, которые относятся к специальным символам операторов SQL. Этот метод применяется для того, чтобы избежать возникновения синтаксической ошибки при выполнении запроса. Строка "`I'm happy`", например, возвращается этим методом как "`'I\'m happy'`" (без двойных кавычек). Если параметр `$value` имеет значение `undef`, он возвращается как строка "NULL" (без кавычек).

  Параметр `$data_type` в большинстве случаев не нужен, поскольку MySQL автоматически преобразует определенные в виде строк значения запросов в другие типы. Этот параметр можно использовать как подсказку, что заданное значение относится к определенному типу. Его значение `DBI::SQL_integer`, например, показывает, что параметр `$value` содержит целочисленное значение.

  Не рекомендуется использовать метод `quote()` для значений, которые планируется вставлять в запрос с помощью маркера. Интерфейс DBI автоматически обрабатывает такие значения.

  Параметр `$data_type` является необязательным, так как СУБД MySQL по необходимости преобразует строковые значения в запросы другого типа.

  Параметр `$data_type` может задаваться в качестве подсказки относительно типа значения. Например, `DBI::SQL_INTEGER` показывает, что значение `$value` имеет целый тип.
- `$rc = $dbh->rollback ();` Осуществляет откат текущей транзакции, если MySQL поддерживает транзакции и режим автовыполнения отключен. Если режим автовыполнения включен, вызов метода `rollback()` не имеет действия и в результате будет получено предупреждение.

  Поддержка транзакций требует DBD::mysql 1.2216 или выше и MySQL 3.23.17 или выше.
- `$ary_ref = $dbh->selectall_arrayref ($statement [, \%attr [, @bind_values]]);` Выполняет запрос, определенный параметром `$statement`, и возвращает результат. Этот метод совмещает возможности

методов prepare(), execute() и fetchall_arrayref(). Если параметр $statement определяет уже подготовленный ранее оператор, этап метода prepare() пропускается. Параметры %attr и @bind_values имеют то же значение, что и для метода do().

Возвращаемое значение представляет собой ссылку на массив. Каждый элемент массива является ссылкой на массив, содержащий значения для одного массива, содержащего значения одной строки результирующего набора. Массив будет пустым, если результирующий набор не содержит строк.

В случае ошибки метод selectall_arrayref() возвращает значение undef, если не был выбран частичный результирующий набор. В таком случае он возвращает строки, полученные до этого момента времени. Для того чтобы убедиться в том, что возвращаемое значение, не равное undef, отражает успешное или ошибочное завершение, проверьте значения $dbh->err() или $DBI::err.

- $hash_ref = $dbh->**selectall_hashref** ($statement, $key_col [, \%attr [, @bind_values]]); Выполняет запрос, определенный параметром $statement, и возвращает результат. Этот метод совмещает в себе возможности методов prepare(), execute() и fetchall_arrayref(). Если параметр $statement определяет ранее уже подготовленный оператор, этап метода prepare() пропускается. Параметры %attr и @bind_values имеют то же значение, что и для метода do().

  Возвращаемое значение является ссылкой на хэш, которая содержит один элемент строк результирующего набора. Ключи хэша — это значения столбца, обозначенные как $key_col, который может быть именем столбца, выделенным запросом или номером столбца (начиная с 1). Значения ключевого столбца должны быть уникальным, что поможет избежать потерь строк из-за коллизий с ключами хэша. Хэш будет пустым, если результирующий набор не содержит строк. С другой стороны, значение любого элемента хэша представляет собой ссылку на хэш, содержащий одну строку результирующего набора, отсортированную по именам столбцов, выбранных запросом.

  При ошибке метод selectall_hashref() возвращает значение undef, если уже не был выбран частичный результирующий набор. В этом случае он возвращает полученные до этого момента времени строки. Для того чтобы убедиться в том, что возвращаемое значение, не равное undef, отражает успешное или ошибочное завершение, проверьте $dbh->err() или $DBI::err.

  Метод selectall_hashref() был добавлен в DBI 1.20. (В действительности он появился в 1.15, но совсем с другими рабочими характеристиками.)

- @ary_ref = $dbh->**selectcol_arrayref** ($statement, [\%attr [, @bind_values]]); Выполняет определенный параметром $statement оператор и возвращает первый столбец результирующего набора, совмещая в процессе выполнения методы prepare() и execute(). Если параметр $statement содержит уже подготовленный оператор, этап метода prepare() пропускается. Параметры %attr и @bind_values имеют то же значение, что и для метода do().

Возвращаемый результат представлен в виде ссылки на массив, содержащий первый столбец из каждой строки.

Если в процессе выполнения имеет место ошибка, этот метод возвращает значение undef. В таком случае он возвращает строки, полученные до этого момента времени. Для того чтобы убедиться в том, что возвращаемое значение, не равное undef, отражает успешное или ошибочное завершение, проверьте $dbh->err() или $DBI::err.

Метод selectall_hashref() был добавлен в DBI 1.09.

- @row_ary = $dbh->**selectrow_array** ($statement [, \%attr [, @bind_values]]); Выполняет запрос, определенный параметром $statement, и возвращает результат. Этот метод совмещает в себе возможности методов prepare(), execute() и fetchall_arrayref(). Если параметр $statement определяет уже подготовленный ранее оператор, этап метода prepare() пропускается. Параметры %attr и @bind_values имеют то же значение, что и для метода do().

  При вызове в контексте списка метод selectrow_array() возвращает массив, представляющий значения в первой строке результирующего набора, или пустой массив, если не возвращается строка или произошла ошибка. В скалярном контексте метод selectrow_array() возвращает один элемент или значение undef, если строка не возвращается или происходит ошибка. Какой элемент возвращается, не определено (см. об этом поведении в описании метода fetchrow_array()).

  Для того чтобы убедиться в том, что возвращаемое значение, не равное undef, представляет успешное завершение или сбой, проверьте $dbh->err() или $DBI::err. Нулевое значение свидетельствует о том, что строка не возвращается. Однако при отсутствии ошибки неопределенное возвращаемое значение в скалярном контексте может означать то, что столбец имеет значение NULL, или то, что строка не была возвращена.

- @ary_ref = $dbh->**selectrow_arrayref** ($statement, [\%attr [, @bind_values]]); Выполняет определенный параметром $statement оператор и возвращает первую строку результирующего набора, совмещая в процессе выполнения методы prepare(), execute() и fetchrow_array(). Если параметр $statement определяет уже подготовленный оператор, этап метода prepare() пропускается. Параметры %attr и @bind_values имеют то же значение, что и для метода do().

  Метод selectrow_array() возвращает ссылку на массив, представляющий значения первой строки результирующего набора, либо значение undef, если имела место ошибка.

  Метод selectall_hashref() был добавлен в DBI 1.15.

- $hash_ref = $dbh->**selectrow_hashref** ($statement [, \%attr [, @bind_values]]); Выполняет запрос, определенный параметром $statement, и возвращает результат. Этот метод совмещает в себе возможности методов prepare(), execute() и fetchall_arrayref(). Если параметр $statement определяет ранее уже подготовленный оператор, этап

метода prepare() пропускается. Параметры %attr и @bind_values имеют то же значение, что и для метода do().

Возвращаемое значение является ссылкой на хэш, содержащий первую строку результирующего набора. В случае ошибки метод selectrow_hashref() возвращает значение undef. Элементы хэш отсортированы по именам выбранных запросом столбцов.

Метод selectrow_hashref() был добавлен в DBI 1.16.

Другие методы дескрипторов баз данных

В последних версиях интерфейса DBI появилось много дополнительных методов дескрипторов баз данных. Некоторые из них до сих пор находятся на стадии эксперимента и большей частью для СУБД MySQL не применяются. Среди них можно упомянуть такие методы, как column_info(), foreign_key_info(), get_info(), primary_key(), primary_key_info(), table_info(), tables(), type_info() и type_info_all(). Подробнее о них можно узнать в документации к интерфейсу DBI:

```
% perldoc DBI
```

Методы дескрипторов операторов

Описанные в этом разделе методы вызываются с помощью дескриптора оператора, который, в свою очередь, можно получить посредством вызова методов prepare() или prepare_cached().

- $rc = $sth->**bind_col** ($col_num, \$var); Привязывает определенный столбец запроса SELECT к переменной языка Perl, которая должна передаваться как ссылка. Значение параметра $col_num может находиться в диапазоне от 1 до количества столбцов, выбранных запросом. Всякий раз при выборке строки эта переменная автоматически обновляется с помощью нового значения столбца.

  Метод bind_col() следует вызывать после метода execute() и до выборки строк. Если заданный номер столбца выходит за пределы диапазона, начинающегося с 1 и до количества выбранных строк, метод bind_col() возвращает значение "False".

- $rc = $sth->**bind_columns** (\$var1, \$var2, ...); Привязывает возвращенные оператором SELECT столбцы к списку переменных. Более детально об этом рассказывается в описании метода bind_col(). Как и метод bind_col(), метод bind_columns() следует вызывать после метода execute() и до выборки строк.

  Если число ссылок не соответствует числу выбранных запросом столбцов, метод bind_colums() возвращает значение "False".

- $rv = $sth->**bind_param** ($n, $value [, \%attr]);

  $rv = $sth->**bind_param** ($n, $value [, $bind_type]); Привязывает значение к маркеру запроса (отображаются символом"?") до его посылки на

сервер. Этот метод следует вызывать после метода prepare() и до вызова метода execute().

Значение параметра $n определяет номер маркера, к которому следует привязать значение $value, и может находиться в диапазоне от 1 до заданного количества маркеров. Чтобы привязать значение NULL, задайте этому методу значение undef.

Параметр \%attr или $bind_type может выполнять роль подсказки относительно типа привязываемого значения. Для определения, например, целочисленного значения метод bind_param() можно вызвать одним из двух следующих способов:

```
$rv = $sth->bind_param ($n, $value, { TYPE => DBI::SQL_INTEGER});
$rv = $sth->bind_param ($n, $value, DBI::SQL_INTEGER);
```

По умолчанию переменная рассматривается как имеющая тип VARCHAR. Зачастую этого достаточно, поскольку MySQL в случае необходимости самостоятельно преобразует типы строковых значений запросов.

- $rows = $sth->**dump_results** ([$maxlen [, $line_sep [, $field_sep [, $fh]]]]); Извлекает все строки, используя дескриптор запроса $sth, форматирует их с помощью функции DBI::neat_list() и выводит, используя заданный дескриптор файла. Этот метод возвращает число выбранных строк.

  Для параметров $maxlen, $line_sep, $field_sep и $fh по умолчанию установлены значения 35, "\n", ", " и STDOUT соответственно.

- $rv = $sth->**execute** ([@bind_values]); Выполняет заранее подготовленный запрос. Возвращает истинное значение, если оператор выполняется успешно, либо значение undef, если имеет место ошибка. Для операторов, отличных от оператора SELECT, возвращаемое значение является числом задействованных строк, -1, если число строк неизвестно, и undef — в случае ошибки. Если количество задействованных строк равно 0, возвращаемое значение равно "0E0", что соответствует нулю в числовых контекстах, но представляет значение "True" в логическом контексте.

  Параметр @bind_values имеет то же значение, что и для метода do().

- $ary_ref = $sth->**fetch** (); Метод fetch() является псевдонимом метода fetchrow_arrayref().

- $tbl_ary_ref = $sth->**fetchall_arrayref** ([$slice_array_ref]);
  $tbl_ary_ref = $sth->**fetchall_arrayref** ([$slice_hash_ref]);
  Выполняет выборку всех строк, используя дескриптор оператора $sth, и возвращает ссылку на массив, содержащий по одной ссылке на каждую выбранную строку. Этот массив может быть пустым, если результирующий набор не содержит строк. Значение каждой ссылки массива зависит от заданного аргумента. Если не задан ни один аргумент или задан аргумент ссылки на массив, каждый элемент значения $tbl_ary_ref является ссылкой на массив, содержащий значения для одной строки результирующего набора. Значения диапазонов массива начинаются с нуля, так как они представляют собой индексы массивов языка Perl. Отрицательные значения начинают от-

счет с конца строки. Таким образом, чтобы выбрать первые и последние столбцы каждой строки, необходимо выполнить следующее:

```
$ary_ref = $sth->fetchall_arrayref ([0, -1]);
```

Если задан аргумент ссылки на хэш, каждый элемент значения `$tbl_ary_ref` является ссылкой на хэш, содержащий значения для одной строки результирующего набора.

Для задания диапазона хэша имена столбцов должны задаваться как ключи хэша, а каждый ключ должен иметь значение, равное 1:

```
$ary_ref = $sth->fetchall_arrayref ({id => 1, name => 1});
```

Для выборки столбцов как хэша передайте пустую ссылку на хэш:

```
$ary_ref = $sth->fetchall_arrayref ({});
```

При возникновении ошибки метод `fetchall_arrayref()` возвращает строки, полученные до этого момента времени. Для того чтобы убедиться в том, что возвращаемое значение отражает успешное или ошибочное завершение, проверьте `$dbh->err()` или `$DBI::err`.

- `$hash_ref = $sth->fetchall_hashref ($key_col);` Производит выборку результирующего набора и возвращает ссылку на хэш, содержащий по одному элементу из каждой строки результирующего набора. Ключи хэша представляют собой значения столбца, обозначенного как `$key_col`, который должен быть именем столбца, выбранного запросом, или номером столбца (начиная с 1).

  Значения ключевого столбца должны быть уникальными, это позволит избежать потери строк из-за коллизий столбцов в хэше. Хэш будет пустым, если результирующий набор не содержит строк. С другой стороны, значение каждого элемента хэша представляет собой ссылку на хэш, содержащий одну выбранную запросом строку результирующего набора, отсортированную по именам столбцов.

  В случае ошибки из-за ошибочного аргумента ключевого столбца метод `fetchall_hashref()` возвращает значение `undef`. Иначе он возвращает строки, полученные до этого момента времени. Для того чтобы убедиться в том, что возвращаемое значение отражает успешное или ошибочное завершение, проверьте значения `$dbh->err()` или `$DBI::err`.

- `@ary = $sth->fetchrow_array ();` При вызове в контексте списка метод `fetchrow_array()` возвращает массив со значениями столбцов для следующей строки результирующего набора либо пустой массив, если строк больше нет или имела место ошибка. При вызове в скалярном контексте метод `fetchrow_array()` возвращает значение первого элемента массива (т.е. первого столбца строки) либо значение `undef`, если строк больше нет или имела место ошибка. До версии 1.29 документация DBI утверждает, что метод `fetchrow_array()` возвращает значение первого столбца в скалярном контексте. Приложения больше не должны полагаться на такое поведение этого метода, так как возвращаемый элемент сейчас является неопределенным; это можно со всей определенностью утверждать только для запросов, возвращающих одно значение.

Чтобы отличить нормальное завершение от ошибочного, достаточно проверить значение метода `$sth->err()`. Нулевое значение отражает нормальное завершение, а соответственно и отсутствие ошибки.

Однако в случае отсутствия ошибки возвращаемое значение `undef` в скалярном контексте может быть неоднозначным и представлять значение столбца NULL или окончание результирующего набора.

- `$ary_ref = $sth->fetchrow_arrayref ();` Возвращает ссылку на массив со значениями столбца для следующей строки результирующего набора. Если строк больше нет или имела место ошибка, этот метод возвращает значение `undef`.

  Чтобы отличить нормальное завершение от ошибочного, достаточно проверить значение метода `$sth->err()`. Нулевое значение отражает нормальное завершение, а соответственно и отсутствие ошибки.

- `$hash_ref = $sth->fetchrow_hashref ([$name]);` Возвращает ссылку на хэш со значениями столбцов для следующей строки результирующего набора. Если строк больше нет или имела место ошибка, этот метод возвращает значение `undef`. В качестве значений индексов хэша выступают имена столбцов, а элементы хэша содержат значения столбцов.

  Для определения атрибута дескриптора оператора, используемого для ключевых значений хэша, может задаваться аргумент `$name`. По умолчанию он имеет значение "NAME" (использовать имена столбцов, указанные в запросе). В этой связи может возникнуть проблема, поскольку регистр символов в именах столбцов в запросах не учитывается, но учитывается в именах ключей хэша. Чтобы определить запись имен ключей символами нижнего или верхнего регистра, задайте значение аргумента `$name` как "NAME_lc" или "NAME_uc" соответственно. По умолчанию он имеет значение "NAME". (Еще один способ управления регистром букв хэша заключается в использовании атрибута `FetchHashKeyName`, который обсуждается ниже, в разделе "Атрибуты интерфейса DBI".)

  Чтобы отличить нормальное завершение от появления ошибки, достаточно проверить значение метода `$sth->err()` или `$DBI::err`. Нулевое значение отражает нормальное завершение, а соответственно и отсутствие ошибки.

- `$rc = $sth->finish ();` Освобождает все ресурсы, связанные с дескриптором оператора. Как правило, какая-нибудь острая необходимость в вызове этого метода отсутствует. Однако при выборке только части результирующего набора вызов метода `finish()` позволит интерфейсу DBI понять, что выборка данных завершена.

  Вызов метода `finish()` может аннулировать атрибуты оператора. Поэтому лучше всего получить доступ к ним сразу после вызова метода `execute()`.

- `$rv = $sth->rows ();` Возвращает число строк, измененных заданным с помощью дескриптора `$sth` оператором, либо значение -1, если имела место ошибка. Этот метод применяется чаще всего для таких не возвращающих строк операторов, как `UPDATE` или `DELETE`. Полагаться на метод `rows()` для операторов `SELECT` не стоит. В подобных случаях необходимо подсчитывать количество выбранных строк.

Методы общих дескрипторов

Описанные в этом разделе методы не относятся к каким-либо отдельным типам дескрипторов и могут вызываться с помощью дескриптора драйвера, базы данных или оператора.

- `$h->err ();` Возвращает числовой код ошибки для последней операции драйвера. Значение 0 указывает на отсутствие ошибки.
- `$h->errstr ();` Возвращает строку сообщения об ошибке для последней операции драйвера. Пустая строка или значение undef указывает на отсутствие ошибки.
- `DBI->trace ($trace_level [, $trace_filename]);`

  `$h->trace ($trace_level [, $trace_filename]);` Устанавливает уровень трассировки, которая обеспечивает информацию о взаимодействии интерфейса DBI. Уровень трассировки может задаваться в диапазоне от 0 (отключено) до 9 (максимальный объем информации). Трассировку можно активизировать для всех операций DBI внутри сценария, вызвав этот метод как метод класса интерфейса DBI, или для отдельного дескриптора.

  `DBI->trace (2);` Активизация трассировки внутри сценария.

  `$sth->trace (2);` Активизация трассировки для дескриптора.

  Установив переменную среды `DBI_TRACE`, можно выполнять трассировку на глобальном уровне для всех сценариев DBI.

  Протоколируемая информация по умолчанию записывается в STDERR. Для прямого вывода ее в отдельный файл можно задать имя файла с помощью параметра `$filename`. Новая информация будет добавляться к существующему содержимому файла.

  При каждом вызове этого метода информация обо *всех* дескрипторах заносится в один файл. Если имя этого файла определено пользователем, то именно в него выполняется запись. Во всех остальных случаях данные записываются в STDERR.
- `DBI->trace_msg ($str [, $min_level])`

  `$h->trace_msg ($str [, $min_level])`. Записывает сообщение в данные трассировки, если трассировка выполняется для отдельного дескриптора, либо на уровне интерфейса DBI. В последнем случае метод `trace_msg()` может вызываться как `DBI->trace_msg()`.

  С помощью параметра `$min_level` можно определить минимальный уровень трассировки, при достижении которого начинают записываться сообщения.

Специальные административные методы MySQL

Этот раздел посвящен методу `func()`, который используется в интерфейсе DBI как средство непосредственного доступа к специальным операциям драйвера.

```
$rc = $drh->func("createdb",
        $db_name, $host_name, $user_name, $password, "admin");
$rc = $drh->func("dropdb",
```

```
        $db_name, $host_name, $user_name, $password, "admin");
$rc = $dbh->func("reload",
        $host_name, $user_name, $password, "admin");
$rc = $drh->func("shutdown",
$rc = $dbh->func("createdb", db_name, "admin");
$rc = $dbh->func("dropdb",   db_name, "admin");
            $host_name, $user_name, $password, "admin");
$rc = $drh->func("reload", "admin");
$rc = $dbh->func("shutdown", "admin");
```

Метод `func()` может вызываться с помощью дескриптора драйвера либо дескриптора базы данных. Дескриптор драйвера не связан с открытым соединением, поэтому если метод `func()` задается именно с его помощью, необходимо определить соответствующие аргументы с именем компьютера и пользователя и паролем. Только наличие всех этих данных позволит методу установить соединение. Если же доступ с помощью метода `func()` обеспечивается через дескриптор базы данных, необходимость в этих аргументах отпадает. В случае необходимости дескриптор драйвера можно получить следующим образом:

```
$drh = DBI->install_driver("mysql"); # (аргумент "mysql" должен быть
записан прописными буквами)
```

Метод `func()` "понимает" следующие действия.

- `createdb`. Создает базу данных с именем `$db_name`. Для выполнения этой операции пользователь должен обладать привилегией `CREATE`.
- `dropdb`. Удаляет базу данных с именем `$db_name`. Для выполнения этой операции пользователь должен обладать привилегией `DROP`.

  Этот параметр следует использовать с особой осторожностью, поскольку восстановить удаленную базу данных будет невозможно.
- `reload`. Указывает серверу перезагрузить таблицы разрешений. Эта операция необходима, если содержимое таблиц изменяется непосредственно с помощью операторов `DELETE`, `INSERT` или `UPDATE`, а не посредством обычных операторов `GRANT` и `REVOKE`. Для выполнения этой операции пользователь должен обладать привилегией `RELOAD`.
- `shutdown`. Завершает работу сервера. Для выполнения этой операции пользователь должен обладать привилегией `SHUTDOWN`.

Единственным действием метода `func()`, которое не может выполняться с помощью обычного механизма обработки запросов DBI, является `shutdown`. Для других действий предпочтительней воспользоваться оператором `CREATE DATABASE`, `DROP DATABASE` или `FLUSH PRIVILEGES`, чем методом `func()`.

Функции программ интерфейса DBI

Описываемые в этом разделе функции необходимо вызывать с применением синтаксиса `DBI::`*`func_name`*`()`, а не `DBI->`*`func_name`*`()`.

- `@bool = DBI::looks_like_number (@ary)`. Считывает список значений и возвращает массив с одним членом для каждого элемента списка. Каждый член массива, в свою очередь, отражает, выглядит ли соответствую-

щий аргумент как число. Соответственно, член массива имеет истинное значение, если аргумент выглядит как число, и значение undef, если аргумент не выглядит так или вовсе пуст.

- `$str = DBI::neat ($value [, $maxlen]);` Возвращает строку с отформатированным представлением аргумента `$value`. Строки при форматировании заключаются в кавычки, чего нельзя сказать о числовых значениях. (Поэтому заключенные в кавычки числа интерпретируются как строки.) Результатом задания неопределенных значений станет значение undef, а непечатаемые символы отображаются символом ".". Например, выполнение цикла

```
for my $val ("a", "3", 3, undef, "\x01\x02")
{
    print DBI::neat ($val) . "\n";
}
```

даст следующие результаты:

```
'a'
'3'
3
undef
'..'
```

Аргумент `$maxlen` определяет максимальную длину результирующей строки. Если ее длина превышает `$maxlen`, результат укорачивается до длины `$maxlen-4` и дополняется многоточием (...). Если аргумент `$maxlen` равен 0, или значению undef, или не задан, по умолчанию функция задается как `$DBI::neat_maxlen(400)`.

Не рекомендуется использовать эту функцию для создания запросов. Чтобы выделить и заключить в кавычки отдельные символы запроса, лучше воспользоваться методом `quote()`.

- `$str = DBI::neat_list (\@ary [, $maxlen [, $sep]]);` Вызывает функцию `neat()` для каждого элемента списка, на который указывает первый аргумент, затем объединяет результаты с помощью строки-разделителя `$sep` и возвращает результат в виде одной строки.

  Аргумент `$maxlen` определяет максимальную длину для отдельных аргументов, а не результирующей строки, возвращаемой функцией `neat()`.

  Если аргумент `$sep` не определен, в качестве разделителя по умолчанию используется символическая последовательность ", ".

Атрибуты интерфейса DBI

Информация об атрибутах обеспечивается на нескольких уровнях интерфейса DBI. Большинство атрибутов связано с дескрипторами баз данных или дескрипторами операторов, однако не с теми и другими одновременно. Некоторые атрибуты, такие, например, как `PrintError` и `RaiseError`, могут связываться как с дескрипторами баз данных, так и с дескрипторами операторов. В общем, каждый дескриптор имеет свои собственные атрибуты. Некоторые атрибуты, содержащие информацию об ошибках, например `err` и `errstr`, являются динамическими, поскольку всегда связаны с дескриптором, используемым последним.

Атрибуты передаются методам connect() или connect_cached() и становятся частью дескриптора результирующего набора, возвращенного этими методами.

Атрибуты дескрипторов баз данных

Атрибуты, заданные в этом разделе, связаны с дескрипторами баз данных.

- $dbh->{**AutoCommit**}; Этот атрибут может быть установлен в значение "true" или "false" для того, чтобы включить или отключить режим автовыполнения транзакций. По умолчанию этот атрибут имеет значение "true". Установка атрибута AutoCommit в значение false позволяет осуществлять транзакции, каждая из которых должна завершаться вызовом метода commit() для успешного завершения или rollback() для отката неуспешной транзакции. Посмотрите также описание метода обработки баз данных begin_work().

  Если используемая версия MySQL не поддерживает транзакций, попытка отключения AutoCommit приводит к фатальной ошибке.

  Для поддержки транзакций требуется DBD::mysql 1.2216 или выше и MySQL 3.23.17 или выше.

- $dbh->{**Statement**}; Хранит самую последнюю строку запроса, переданную методу prepare() через заданный дескриптор базы данных.

Общие атрибуты дескрипторов

Общие атрибуты могут применяться к отдельным дескрипторам либо задаваться в аргументе %attr для методов, которые их принимают. Используемые вместе с методом connect() атрибуты влияют глобально на DBI-обработку во всем сценарии.

- $h->{**ChopBlanks**}; Определяет, должны ли методы выборки строк удалять замыкающие пустые символы из значений столбцов типа CHAR. По умолчанию атрибут ChopBlanks деактивизирован для большинства драйверов баз данных, однако для данных типа CHAR или VARCHAR это все равно не играет никакой роли, поскольку сервер всегда убирает пустые символы из значений этих типов. Однако для столбцов типа BLOB и TEXT установка значений ChopBlanks в "true" приводит к удалению завершающих пробелов.

- $h->{**FetchHashKeyName**}; Управление регистром букв, используемых в ключах хэша в строках результирующего набора, возвращаемых методом fetchrow_hashref() или другими методами, косвенно вызывающими метод fetchrow_hashref(). Значение по умолчанию равно "NAME" (использование имен столбцов, заданных в операторе SELECT). Другими допустимыми значениями являются значения "NAME_lc" и "NAME_uc", позволяющие переводить имена столбцов, используемых в качестве ключей хэша, в верхний или нижний регистр. Этот атрибут имеет отношение только к дескрипторам баз данных и драйверам.

  Атрибут FetchHashKeyName был добавлен в DBI 1.19.

- $h->{**HandleError**}; Этот атрибут используется для обработки ошибок. Он может указывать на подпрограмму, которая вызывается при возникно-

вении ошибки до обычной обработки методами RaiseError и PrintError. При возвращении этой подпрограммой значения true, обработка методами RaiseError и PrintError не осуществляется, в противном случае обработка осуществляется в обычном порядке. (Конечно, программа error может остановить выполнение сценария без возвращения.)

Интерфейс DBI передает программе обработки ошибок три аргумента: текст сообщения об ошибке, дескриптор DBI, который используется в точке обнаружения ошибки, и первое значение, которое возвращается сбойным методом.

Атрибут HandleError был добавлен в DBI 1.21.

- $h->{**PrintError**}; При активизации этого атрибута появление связанной с интерфейсом DBI ошибки приводит к выводу предупреждения. Этот атрибут отключен по умолчанию.
- $h->{**RaiseError**}; При активизации этого атрибута появление ошибки, связанной с интерфейсом DBI, приводит к автоматическому завершению работы сценария. Этот атрибут отключен по умолчанию.
- $h->{**ShowErrorStatement**}; Если этот атрибут активизирован, сообщения об ошибках снабжаются соответствующими текстами запросов. Атрибут ShowErrorStatement по умолчанию отключен. Действие этого атрибута ограничено дескрипторами операторов и методами prepare() и do().

  Метод ShowErrorStatement был добавлен в DBI 1.19.
- $h->{**TraceLevel**}; Установка уровня трассировки для заданного дескриптора. Этот атрибут является альтернативой методу trace().

  Метод TraceLevel был добавлен в DBI 1.21.

Специальные атрибуты дескрипторов баз данных MySQL

Описанные в этом разделе атрибуты используются исключительно для MySQL-драйвера интерфейса DBI, DBD::mysql. Как сказано в описаниях атрибутов, каждый из них связан с определенной функцией интерфейса MySQL C API. Подробнее об этом см. в приложении E, "Программный интерфейс приложений C".

- $rv = $dbh->{**mysql_errno**}; Подобно функции интерфейса C API mysql_errno(), возвращает код последней ошибки.
- $str = $dbh->{**mysql_error**}; Подобно функции интерфейса C API mysql_error(), возвращает диагностическое сообщение последней ошибки.
- $str = $dbh->{**mysql_hostinfo**}; Подобно функции интерфейса C API mysql_get_host_info(), возвращает строку описания данного соединения.
- $str = $dbh->{**mysql_info**}; Подобно функции интерфейса C API mysql_info(), возвращает информацию о запросах, которые воздействуют сразу на несколько строк.
- $rv = $dbh->{**mysql_insertid**}; Подобно функции интерфейса C API mysql_insert_id(), возвращает значение AUTO_INCREMENT, которое было недавно сгенерировано на соединении, связанном с $dbh.

Этот атрибут может использоваться также и с дескрипторами операторов.

- `$rv = $dbh->{mysql_protoinfo};` Подобно функции интерфейса C API `mysql_get_proto_info()`, возвращает число, соответствующее версии протокола "клиент/сервер", которая используется для заданного соединения.
- `$str = $dbh->{mysql_serverinfo};` Возвращает строку описания версии сервера, например "4.0.2-alpha-log". Это значение состоит из номера версии с последующими одним или несколькими суффиксами. Этот атрибут возвращает информацию, подобную информации, возвращаемой функцией интерфейса C API `mysql_get_server_info()`.

  Значения суффиксов см. в описании функции `VERSION()` в приложении B, "Операторы и функции".
- `$str = $dbh->{mysql_stat};` Подобно функции интерфейса C API `mysql_stat()`, возвращает строку с описанием состояния сервера.
- `$rv = $dbh->{mysql_thread_id};` Подобно функции интерфейса C API `mysql_thread_id()`, возвращает номер процесса соединения, связанного с `$dbh`.

Начиная с DBD::mysql 2.0900 несколько специальных атрибутов дескрипторов баз данных MySQL было переведено в разряд устаревших и больше не используется. В табл. Ж.2 приводится перечень таких атрибутов с соответствующими им используемыми атрибутами. Если ваша версия DBD::mysql еще не поддерживает новые атрибуты, есть два пути: использовать устаревшие атрибуты или обновить версию DBD::mysql. Существует также возможность написать программный код, который использует новейшую версию атрибута, если она поддерживается, или использовать старую версию, если она не поддерживается. Например:

```
my $info = $dbh->{mysql_info};
# применить старую форму, если новая форма отсутствует
$info = $dbh->{info} unless defined ($info);
```

Таблица Ж.2. Исключенные специальные атрибуты дескрипторов баз данных MySQL

Исключенный атрибут	Заменивший его атрибут
`errno`	`mysql_errno`
`error`	`mysql_error`
`hostinfo`	`mysql_hostinfo`
`info`	`mysql_info`
`protoinfo`	`mysql_protoinfo`
`serverinfo`	`mysql_serverinfo`
`stats`	`mysql_stat`
`thread_id`	`mysql_thread_id`

Атрибуты дескрипторов операторов

Атрибуты дескрипторов операторов чаще всего применяются к запросам `SELECT` (или запросам аналогичного типа) и не действуют, если предварительно не был задан запрос методу `prepare()` для получения дескриптора оператора и

не был вызван метод execute(). Кроме того, сделать недействительными некоторые атрибуты операторов может и метод finish() (или по достижении окончания результирующего набора, который косвенно вызывает метод finish()).

Многие описанные далее атрибуты имеют значения, которые являются ссылкой на массив значений (по одному значению на столбец запроса). Количество элементов в массиве определяется атрибутом $sth->{NUM_OF_FIELDS}. Для атрибутов оператора stmt_attr, который является ссылкой на массив, можно ссылаться на целый массив как на @{$sth->{*stmt_attr*}} или последовательно проходить по элементам массива следующим образом:

```
for (my $i = 0; $i < $sth->{NUM_OF_FIELDS}; $i++)
{
   my $value = $sth->{stmt_attr}->[$i];
}
```

Атрибуты NAME_hash, NAME_lc_hash и NAME_uc_hash возвращают ссылку на хэш. Последовательно обработать элементы массива можно следующим образом:

```
foreach my $key (keys (%{$sth->{stmt_attr}}))
{
   my $value = $sth->{stmt_attr}->{$key};
}
```

- $ary_ref = $sth->{**NAME**}; Ссылка на массив строк, отображающих названия столбцов. Регистр символов названий задается оператором SELECT.
- $ary_ref = $sth->{**NAME_hash**}; Ссылка на хэш, содержащий строки с именами всех столбцов. Регистр имен задается оператором SELECT. Значение всех элементов хэша отображает положение соответствующего столбца в строках результирующего набора (начиная с 0).

  Атрибут NAME_hash был добавлен в DBI 1.20.
- $ary_ref = $sth->{**NAME_lc**}; Ссылка на массив строк, отображающих названия столбцов. Имена возвращаются как строки в нижнем регистре.
- $ary_ref = $sth->{**NAME_lc_hash**}; Аналогичен атрибуту NAME_hash, но имена возвращаются как строки в нижнем регистре.

  Атрибут NAME_lc_hash был впервые добавлен в DBI 1.20.
- $ary_ref = $sth->{**NAME_uc**}; Ссылка на массив строк, отображающих названия столбцов. Имена возвращаются как строки в верхнем регистре.
- $ary_ref = $sth->{**NAME_uc_hash**}; Аналогичен атрибуту NAME_hash, но имена возвращаются как строки в верхнем регистре.

  Атрибут NAME_uc_hash был впервые добавлен в DBI 1.20.
- $ary_ref = $sth->{**NULLABLE**}; Ссылка на массив значений, отображающих возможность столбцов содержать значение NULL. Элемент массива может быть равен 0 (не может), 1 (может) или 2 (не определено).
- $ary_ref = $sth->{**NUM_OF_FIELDS**}; Число столбцов, которое возвратит готовый запрос, либо нулевое значение (для операторов, не использующих SELECT).
- $ary_ref = $sth->{**NUM_OF_PARAMS**}; Количество маркеров в подготовленном запросе.

- `$ary_ref = $sth->{PRECISION};` Ссылка на массив значений, отражающих точность столбцов. Понятие "точность" для интерфейса DBI означает то же, что и для ODBC, в то время как в MySQL это значение называется "максимальной шириной столбца". Для числовых столбцов под точностью понимается ширина отображения на экране, а для строковых — максимальная длина столбца в соответствии с определением оператора CREATE TABLE.
- `$ary_ref = $sth->{SCALE};` Ссылка на массив значений, отражающих масштаб каждого столбца. Понятие "масштаб" для интерфейса DBI означает то же, что и для ODBC, в то время как в MySQL это значение называется "числом десятичных знаков" для столбцов значений с плавающей запятой. Для других столбцов значение масштаба равно нулю.
- `$ary_ref = $sth->{Statement};` Текст запроса, связанного с дескриптором `$sth`. Именно этот текст считывается до замены маркеров методом `prepare()`.
- `$ary_ref = $sth->{TYPE};` Ссылка на массив значений, отражающих числовой тип каждого столбца. До DBD::mysql 1.1919 этот атрибут был аналогом `mysql_type`. Начиная с 1.1919 он содержит переносимые номера типов, а `mysql_type` применяется для получения номеров типов в классификации MySQL.

Специальные атрибуты дескрипторов операторов MySQL

Большинство описанных в этом разделе атрибутов дескрипторов операторов следует рассматривать как доступные только для чтения. Обрабатывать их следует только после вызова метода `execute()`. Исключение из этих двух правил составляют лишь атрибуты `mysql_store_result` и `mysql_use_result`. Драйвер DBD::mysql предоставляет возможность управлять стилем обработки результирующего набора, используемого сценарием. Атрибуты дескрипторов операторов `mysql_store_result` и `mysql_use_result` определяют вид обработки результирующего набора API-функций `mysql_store_result()` и `mysql_use_result()` языка C. Более детально об этих функциях и отличиях между ними см. в приложении E, "Программный интерфейс приложений C". В интерфейсе DBI по умолчанию используется функция `mysql_store_result()`, однако пользователь имеет возможность активизировать атрибут `mysql_use_result`, который указывает интерфейсу DBI применять функцию `mysql_use_result()`. Устанавливать этот атрибут следует после вызова метода `prepare()`, но до вызова метода `execute()`:

```
$sth = $dbh->prepare (...);
$sth->{mysql_use_result} = 1;
$sth->execute();
```

- `$rv = $dbh->{mysql_insertid};` Значение AUTO_INCREMENT, которое было последним сгенерировано в соединении, связанном с дескриптором `$dbh`.

  Этот атрибут также может использоваться с дескрипторами баз данных.

- `$ary_ref = $sth->{mysql_is_auto_increment};` Ссылка на массив значений, отражающих, может ли быть каждый столбец столбцом AUTO_INCREMENT.

  Этот атрибут может использоваться начиная с драйвера DBD::mysql 2.1014.

- `$ary_ref = $sth->{mysql_is_blob};` Ссылка на массив значений, показывающих, является ли тип столбца типом BLOB или TEXT.

- `$ary_ref = $sth->{mysql_is_key};` Ссылка на массив значений, отражающий принадлежность столбца к ключу.

- `$ary_ref = $sth->{mysql_is_num};` Ссылка на массив значений, отражающий, имеет ли столбец числовой тип.

- `$ary_ref = $sth->{mysql_is_pri_key};` Ссылка на массив значений, отражающий принадлежность столбца к ключу PRIMARY KEY.

- `$ary_ref = $sth->{mysql_length};` Этот атрибут аналогичен атрибуту PRECISION.

- `$ary_ref = $sth->{mysql_max_length};` Ссылка на массив значений, отражающих реальную максимальную длину столбцов в результирующем наборе.

- `$ary_ref = $sth->{mysql_store_result};` Если атрибут `mysql_store_result` активизирован (установлен равным 1), результирующие наборы извлекаются из сервера MySQL с помощью функции языка C `mysql_store_result()`, а не с помощью `mysql_use_result()`. Более детально об этих функциях и различиях между ними рассказывается в приложении E, "Программный интерфейс приложений C".

  Если планируется установить атрибут `mysql_store_result`, эту установку необходимо сделать после вызова метода `prepare()` и до вызова `execute()`.

- `$ary_ref = $sth->{mysql_table};` Ссылка на массив значений с именами таблиц, из которых считываются столбцы. В качестве имени таблицы для вычисляемого столбца выступает пустая строка.

- `$ary_ref = $sth->{mysql_type};` Ссылка на массив значений, отражающих количество типов MySQL для каждого столбца результирующего набора.

- `$ary_ref = $sth->{mysql_type_name};` Ссылка на массив значений, отражающих название типа MySQL для каждого столбца результирующего набора.

- `$ary_ref = $sth->{mysql_use_result};` Если атрибут `mysql_use_result` активизирован (установлен равным 1), результирующие наборы извлекаются из сервера MySQL с помощью функции языка C `mysql_use_result()`, а не с помощью `mysql_store_result()`. Более детально об этих функциях и отличиях между ними см. в приложении E, "Программный интерфейс приложений C".

  Обратите внимание, что использование этого атрибута может сделать недействительными некоторые другие атрибуты, такие, например, как `mysql_max_length`. *Этот атрибут также не позволяет использовать метод* `rows()`, хотя в любом случае подсчитывать число строк лучше всего после полной выборки.

Если планируется установить атрибут `mysql_use_result`, осуществите эту установку после вызова метода `prepare()` и до вызова `execute()`.

Некоторые специальные атрибуты MySQL были доступны в старых версиях драйвера DBD::mysql, однако в настоящее время исключены из него и заменены более новыми. Эти атрибуты и их современные заменители представлены в табл. Ж.3. При работе с немного устаревшей версией драйвера DBD::mysql, который не поддерживает новых атрибутов, воспользуйтесь устаревшими формами (либо же обновите версию драйвера). Можно также написать программный код, использующий новейшую версию атрибута, если она поддерживается, или использующий устаревшую версию в случае использования более старой версии драйвера. Например:

```
my $lengths = $sth->{PRECISION};
# применить старую форму, если новая форма отсутствует
$lengths = $sth->{length} unless defined ($lengths);
```

Обратите также внимание, что `insertid` является атрибутом дескриптора оператора, в то время как его новая форма, `mysql_insertid`, — атрибутом дескриптора базы данных.

Таблица Ж.3. Устаревшие специальные атрибуты дескрипторов операторов MySQL

Устаревший атрибут	Новый атрибут
`insertid`	`mysql_insertid`
`is_blob`	`mysql_is_blob`
`is_key`	`mysql_is_key`
`is_not_null`	`NULLABLE`
`is_num`	`mysql_is_num`
`is_pri_key`	`mysql_is_pri_key`
`length`	`PRECISION`
`max_length`	`mysql_max_length`
`table`	`mysql_table`

Динамические атрибуты

Динамические атрибуты связаны с используемым последним дескриптором, представленным в последующих описаниях как `$h`. Они используются сразу же после метода, который их установил, и перед вызовом другого метода, который может изменить их установку.

- `$rv = $DBI::err`. Аналогичен вызову `$h->err()`.
- `$str = $DBI::errstr`. Аналогичен вызову `$h->errstr()`.
- `$rows = $DBI::rows`. Аналогичен вызову `$h->rows()`.

Переменные среды интерфейса DBI

Интерфейс DBI анализирует несколько переменных среды, перечисленных в табл. Ж.4. Все эти переменные, за исключением `DBI_TRACE`, используются методом `connect()`. Переменная `DBI_TRACE` используется методом `trace()`.

Таблица Ж.4. Переменные среды DBI

Название	Значение
DBI_DRIVER	Имя драйвера уровня DBD ("mysql" для MySQL)
DBI_DSN	Имя источника данных
DBI_PASS	Пароль
DBI_TRACE	Уровень трассировки и/или файл вывода протоколируемой информации
DBI_USER	Имя пользователя

3

Программный интерфейс языка PHP

В этом приложении...

Написание сценариев PHP
Функции

Это приложение посвящено описанию программного интерфейса языка написания сценариев PHP для MySQL. Данный программный интерфейс состоит из набора функций, обеспечивающих взаимодействие с серверами MySQL и получение доступа к базам данных. Эти функции попадают в следующие категории:

- функции управления соединением;
- функции диагностирования ошибок;
- функции конструирования и выполнения запросов;
- функции обработки результатов, полученных от запросов;
- информационные функции;
- устаревшие функции.

Рекомендуется использовать это приложение как справочное руководство, поскольку оно включает только краткие фрагменты кода, иллюстрирующие использование программного интерфейса языка PHP. Полностью завершенные сценарии, а также рекомендации по их написанию см. в главе 8, "Интерфейс API для языка написания сценариев PHP." Здесь же описаны только функции, которые непосредственно связаны с MySQL. Руководство по PHP содержит сотни страниц справочного материала, поэтому данное приложение может только подытожить

определенную часть возможностей PHP. Для получения полного справочного руководства по PHP посетите Web-узел PHP по следующему адресу:
`http://www.php.net/`

Написание сценариев PHP

Сценарии PHP представляют собой простые текстовые файлы, которые могут содержать комбинацию HTML и PHP-кода. В процессе интерпретации таких сценариев создается Web-страница, которая отправляется клиенту. HTML-код копируется в выводимые данные без предварительной интерпретации. Код PHP обрабатывается и заменяется создаваемыми выходными данными (или удаляется при отсутствии вывода).

Средства PHP "приступают" к интерпретации файла в режиме HTML. Переключаться в режим кода PHP и обратно можно с помощью специальных тэгов, обозначающих соответственно начало и конец кода PHP. Количество переключений между режимами в одном файле не ограничено. Язык PHP распознает четыре типа тэгов, причем некоторые из них должны явным образом добавляться, если пользователь желает воспользоваться ими. Один из способов сделать это — включить тэги в файл инициализации PHP `php.ini` (или `php3.ini` в PHP3). Месторасположение этого файла зависит от системы, хотя зачастую его можно найти в каталоге `/usr/lib` или `/usr/local/lib`. В Windows его можно найти в системном каталоге.

PHP распознает тэги следующих типов.

- В стиле по умолчанию (the default style) используются тэги "`<?php`" и "`?>`":

  ```
  <?php print ("Hello, world."); ?>
  ```

- В стиле с коротким тэгом открытия (short-open-tag style) используются тэги "`<?`" и "`?>`":

  ```
  <? print ("Hello, world."); ?>
  ```

  В PHP 4 этот стиль также позволяет использовать тэги "`<?=`" и "`?>`" в качестве краткой формы для отображения результата выражения без использования оператора печати:

  ```
  <?= "Hello, world." ?>
  ```

  Этот стиль тэгов можно активизировать с помощью установки в файле инициализации PHP:

  ```
  short_open_tag = On;
  ```

- В совместимом с Active Server Page стиле (Active Server Page-compatible style) используются тэги "`<%`" и "`%>`":

  ```
  <% print ("Hello, world."); %>
  ```

  В PHP 4 этот стиль также позволяет использовать тэги "`<%=`" и "`%>`" в качестве краткой формы для отображения результата выражения без использования оператора печати:

  ```
  <%= "Hello, world." %>
  ```

  Тэги этого стиля также можно активизировать с помощью установки в файле инициализации PHP:

```
asp_tags = On;
```

Поддержка ASP-тэгов была добавлена в PHP версии 3.0.4.

- При работе с HTML-редактором, который не распознает другие стили, можно использовать тэги `<script>` и `</script>`:

```
<script language="php"> print ("Hello, world."); </script>
```

В этом приложении во всех примерах для открытия и закрытия PHP-кода будут использоваться исключительно тэги типа `<?php` и `?>`.

Функции

Этот раздел посвящен описанию связанных с MySQL функций языка PHP. Периодически повторяющиеся в последующем изложении имена параметров несут следующую смысловую нагрузку.

- Многие описанные далее функции считывают необязательный параметр `conn_id` (идентификатор соединения), который определяет соединение с сервером MySQL. Если при вызове функции идентификатор связи не задается, вместо него используется последнее открытое соединение. Некоторые функции попытаются установить соединение самостоятельно, если идентификатор соединения не задан и нет ни одного открытого соединения.
- Параметр `result_id` представляет собой идентификатор результирующего набора, обычно возвращаемый функцией `mysql_query()`.
- Параметры `row_num` и `col_num` отражают номер строки и столбца в результирующем наборе. Оба нумеруются начиная с нуля.
- Необязательные параметры заключаются в квадратные скобки (`[]`).

Некоторые функции в случае возникновения ошибки, помимо возвращения значения состояния, выводят еще и сообщение об ошибке. При работе в Web это сообщение появляется в отправляемых клиентскому броузеру данных вывода, что не всегда желательно. Чтобы исключить вывод сообщения об ошибке (возможно, закодированного), перед именем вызываемой функции необходимо установить символ "@". Например, чтобы отменить вывод сообщения об ошибке функцией `mysql_pconnect()`, можно задать ее следующим образом:

```
<?php
   $conn_id = @mysql_connect ("cobra.snake.net", "sampadm", "secret")
     or die ("Невозможно установить соединение \n");
   print ("Соединение установлено успешно \n");
?>
```

В примерах этого приложения при соединении используется оператор "@", и в случае ошибки печатается собственное диагностическое сообщение.

Второй способ избежать вывода сообщения об ошибке — воспользоваться функцией `error_reporting()`:

```
<?php
   error_reporting (0); #отменить все сообщения об ошибках
   $conn_id = @mysql_connect ("cobra.snake.net", "sampadm", "secret")
     or die ("Невозможно установить соединение \n");
   print ("Соединение установлено успешно \n");
?>
```

Многие сценарии из приведенных примеров выводят после подключения к серверу MySQL сообщение "Соединение установлено успешно". Это позволяет убедиться в том, что сценарий выводит данные, если, конечно, он используется исключительно в личных целях. (Для сценариев PHP, которые ничего не выводят, некоторые броузеры выдают предупреждение "page contains no data" (страница не содержит данных). Это может создать впечатление, что сценарий не работает.)

Как правило, примеры распечатывают сообщения и результаты запросов в виде простого текста. Это сделано для того, чтобы сделать сценарии более "читабельными". Однако в сценариях, предназначенных для выполнения в Web-среде, выводимую информацию необходимо кодировать с помощью функции `htmlspecialchars()`, если она может содержать символы, являющиеся в HTML специальными, такие как '<', '>' или '&'.

В последующих описаниях выражение "запрос SELECT" означает запрос SELECT или любой другой, возвращающий строки, например DESCRIBE, EXPLAIN или SHOW.

Функции управления соединением

Описываемые в этом разделе функции используются для подключения и отключения от сервера MySQL или для изменения имени текущего пользователя MySQL.

- `int` **`mysql_change_user`** `(string user_name, string password [, string db_name [, resource conn_id]]);` Изменить пользователя и базу данных по умолчанию для соединения, определенного параметром `conn_id`, или текущего соединения, если параметр `conn_id` не задан. База данных становится базой данных по умолчанию для ссылок на таблицы, не содержащих спецификации базы данных. Если `db_name` не задано, база данных по умолчанию не выбирается.

  Функция `mysql_change_user()` возвращает значение "true", если у пользователя есть право доступа к серверу и если база данных была определена и есть доступ к базе данных. Если функция завершается с ошибкой, и текущий пользователь, и текущая база данных остаются неизменными.

  Для работы функции `mysql_change_user()` требуется MySQL 3.23.3 или выше. Она была добавлена в PHP 3.0.13, но отсутствует в PHP 4.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    mysql_change_user ("unknown", "public")
        or die ("Нельзя изменить пользователя \n");
    print ("Пользователь изменен успешно \n");
 ?>
```

- `bool` **`mysql_close`** `([resource conn_id]);` Закрывает соединение с сервером MySQL, определенное идентификатором `conn_id`. Если идентификатор соединения не задан, функция `mysql_close()` закрывает соединение, открытое последним.

  Функция `mysql_close()` возвращает значение "true" в случае успешного завершения операции и значение "false", если имела место ошибка. Постоян-

ные соединения, открытые с помощью функции mysql_pconnect(), функция mysql_close() пропускает, однако возвращает значение "true". Поэтому если планируется в будущем закрыть новое соединение, рекомендуется открыть его с помощью функции mysql_connect(), а не с mysql_pconnect().

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    print ("Соединение установлено успешно \n");
    mysql_close ($conn_id);
?>
```

Неустойчивые соединения не требуется закрывать специально явным образом, так как интерпретатор PHP закроет его автоматически после завершения работы сценария. С другой стороны, если сценарий будет работать продолжительное время после того, как он завершил работу с сервером MySQL, закрытие соединения, если необходимость в нем отпала, облегчит работу сервера, который сможет потом использовать соединение для другого клиента.

- resource **mysql_connect** ([string host_name] [, string user_name [, string password [, new_conn]]]); Открывает для пользователя user_name с паролем password соединение с сервером MySQL, установленным на компьютере host_name. Возвращает идентификатор соединения (положительное число), указывающий на новое соединение. В случае возникновения ошибки возвращает значение "false".

  В версии PHP 3.0B4 и более поздних версиях параметр имени компьютера можно (но не необязательно) определять с номером порта, используя синтаксис "*host_name:port_num*". В версии PHP 3.0.10 и более поздних имя компьютера localhost можно (но необязательно) определять, указав разъем домена UNIX, используя синтаксис "localhost:*socket_name*". Путь к файлу разъема должен быть полным. При подключении к локальному узлу с использованием не разъема UNIX, а TCP/IP-адресации адрес задается значением узла "127.0.0.1".

  Если параметр имени компьютера отсутствует, по умолчанию используется имя localhost. Если же отсутствует или пуст параметр имени пользователя, по умолчанию выбирается пользователь, от имени которого запущен PHP. (Если PHP работает как модуль Apache, то выбирается пользователь, с полномочиями которого работает Web-сервер. Если PHP работает как автономная программа, это имя пользователя, под которым работает сценарий PHP.) Если отсутствует или имеет пустое значение параметр пароля, отправляется пустой пароль.

  В PHP 4 можно внести изменение в стандартные значения, которые используются для недостающих параметров соединения по определению директив настройки в файле инициализации PHP, php.ini. Эти директивы перечислены в табл. 3.1.

  Если функция mysql_connect() вызывается с параметрами (имя компьютера, имя пользователя и пароль) уже существующего открытого соединения, новое соединение не устанавливается. Функция mysql_connect()

возвращает существующий идентификатор соединения. Начиная с 4.2.0 эту ситуацию можно обойти и вынудить функцию mysql_connect() открыть новое соединение передачей значения "true" параметру new_conn.

Таблица 3.1. Директивы настройки для установки параметров соединения, заданных по умолчанию

Параметр соединения	Директива настройки
Hostname	mysql.default_host
Username	mysql.default_user
Password	mysql.default_password
Port number	mysql.default_port
Socket pathname	mysql.default_socket

Для закрытия открытого ранее соединения можно воспользоваться функцией mysql_close(). Открытое соединение закрывается автоматически по завершении работы сценария.

Функции PHP имеют необязательный аргумент идентификатора соединения. И если он не задан, то будет использован самый последний идентификатор соединения. Если соединение не открыто, функции смогут попробовать открыть его сами. Это функции, формирующие запросы, функции выборки, создания или удаления баз данных и функции, позволяющие осуществить выборку информации о соединении. Во всех этих случаях результат будет таким же, как при вызове функции mysql_connect() без указания аргументов с использованием параметров соединения по умолчанию. При неуспешной попытке неявного соединения функция, устанавливающая это соединение, также даст сбой.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    print ("Соединение установлено успешно \n");
    mysql_close ($conn_id);
?>
```

- resource **mysql_pconnect** ([string host_name] [, string user_name [, string password]]); Функция mysql_pconnect() аналогична функции mysql_connect(), с той лишь разницей, что открывает устойчивое соединение. Соответственно, такое соединение остается открытым даже после завершения работы сценария. Если функция mysql_pconnect() вызывается с параметрами (имя компьютера, имя пользователя и пароль) уже существующего открытого соединения, это соединение используется повторно. Это позволяет избежать закрытия и повторного открытия соединений и является более эффективным методом соединения, чем использование неустойчивых соединений.

  Использовать устойчивые соединения есть смысл только в том случае, когда PHP выполняется как модуль внутри Web-сервера, который, в свою очередь, продолжает работу после завершения работы сценария PHP. (Соединение может быть использовано многократно, если сервер получает другие запро-

сы для сценария PHP, который подключается к MySQL с аналогичными параметрами.) Если же сценарий выполняется автономной версией PHP, соединение закрывается по завершении работы сценария, поскольку процесс PHP также завершает свою работу.

Вызывать функцию `mysql_close()` для постоянного соединения бессмысленно. Эта функция возвращает значение "`true`", оставляя соединение открытым.

```
<?php
    $conn_id = @mysql_pconnect ("cobra.snake.net", "sampadm",
    "secret")
        or die ("Невозможно установить соединение \n");
    print ("Соединение установлено успешно \n");
?>
```

- `bool` **`mysql_ping`** `([resource conn_id]);` Повторная установка соединения с сервером, если время соединения вышло. Возвращает значение "`true`", если соединение по-прежнему активно или было восстановлено успешно, в противном случае возвращает значение "`false`".

  Функция `mysql_ping()` была добавлена в PHP 4.3.0.

Функции состояния и диагностики ошибок

Функции `mysql_errno()` и `mysql_error()` возвращают числовой код и информационное сообщение об ошибке в работе связанных с MySQL функций PHP. Однако до PHP 4.0.6 эта информация становится доступной только после задания функциям правильного идентификатора соединения. Другими словами, эти функции не могут использоваться для получения информации о результатах безуспешного завершения вызовов функций `mysql_connect()` или `mysql_pconnect()`, поскольку идентификатор связи становится доступным только после успешного установления соединения. Чтобы получить сообщение об ошибке MySQL при установлении соединения, активизируйте переменную `track_errors` посредством установки ее в файле инициализации PHP:

```
track_errors = On;
```

После этого перезапустите Web-сервер, если PHP работает как модуль Apache. Это позволит получить сообщение об ошибке при попытке установления соединения, сославшись на переменную `$php_errormsg`:

```
<?php
   $ conn_id = @mysql_connect("badhost","baduser", "badpass")
     or die ("Невозможно установить соединение: $php_errormsg \n");
   print ("Соединение установлено успешно \n");
?>
```

Можно прибегнуть и к комбинированному подходу, при котором функции обработки ошибок используются по возвращении информации об ошибочной попытке соединения и который в противном случае обращается к переменной `$php_errormsg`. Следующий пример иллюстрирует одно из таких решений. Если `mysql_error()` возвращает ненулевое значение, используются функции обработки ошибок. В противном случае для получения сообщения об ошибке вызывается переменная `$php_errormsg`.

```
<?php
   if (!($conn_id = @mysql_connect("badhost", "baduser", "badpass")))
   {
     # функция mysql_errno() возвращает ненулевые значения при
     обработке ошибок соединения
     if (mysql_errno ())
     {
         $msg = sprintf ("Невозможно установить соединение: %s (%d)\n",
         mysql_error (), mysql_errno ());
     }
     else # вызвать значение $php_errormsg
     {
         $msg = "Невозможно установить соединение: $php_errormsg\n";
     }
     die ($msg);
   }
   print ("Соединение установлено успешно \n");
?>
```

- int **mysql_errno** ([resource conn_id]); Возвращает код ошибки для связанной с заданным соединением MySQL функции, которая последней возвращала значение состояния. Нулевой код указывает на отсутствие ошибки.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение,
        error=" . mysql_errno () . "\n");
    print ("Соединение установлено успешно \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных,
        error=" . mysql_errno () . "\n");
    $query = "SELECT * FROM president";
    $result_id = mysql_query ($query)
        or die ("Неуспешное завершение запроса,
        error=" . mysql_errno () . "\n");
    mysql_free_result ($result_id);
?>
```

- string **mysql_error** ([resource conn_id]); Возвращает строку с сообщением об ошибке для связанной с заданным соединением MySQL функции, которая последней возвращала значение состояния. Пустая строка свидетельствует об отсутствии ошибки.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение,
        error=" . mysql_ error () . "\n");
    print ("Соединение установлено успешно \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных,
        error=" . mysql_error () . "\n");
    $query = "SELECT * FROM president";
    $result_id = mysql_query ($query)
        or die ("Неуспешное завершение запроса,
        error=" . mysql_error () . "\n");
    mysql_free_result ($result_id);
?>
```

Функции построения и выполнения запросов

Описанные в этом разделе функции используются для создания и отсылки запросов на сервер MySQL. Строки запроса должны состоять из одного оператора SQL и не должны заканчиваться символом "точка с запятой" (";") или последовательностью "\g". Окончания ";" и "\g" являются соглашениями клиентской программы mysql и не используются при выводе запросов через PHP.

- `string` **`mysql_escape_string`** `(string str);` Выделяет любые кавычки или другие специальные символы в аргументе `str`, предваряя каждый из них обратной косой чертой. Результатом этой функции является новая строка. Она используется для подготовки операции вставки данных в запросы SQL. В PHP 4.3.0 и более поздних версиях поддерживается также функция `mysql_real_escape_string()`, которая равносильна функции `mysql_escape_string()`, но принимает аргумент с идентификатором соединения и при декодировании учитывает текущую кодировку. Подробнее об этом — ниже, в описании функции `mysql_real_escape_string()`.

  Перечень символов, выделяемых с помощью этой функции, см. в описании функции `mysql_real_escape_string()` в приложении Е, "Программный интерфейс приложений C".

  Функция `mysql_escape_string()` была добавлена в PHP 4.0.3. До этой версии для получения аналогичного результата использовалась функция `addslashes()`.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $last_name = mysql_escape_string ("O'Malley");
    $first_name = mysql_escape_string ("Brian");
    $query = "INSERT INTO member
(last_name,first_name,expiration)"
        . " VALUES('$last_name', '$first_name', '2002-6-3')";
    $result_id = mysql_query ($query)
        or die ("Неуспешное завершение запроса \n");
    printf ("членский номер для нового члена: %d\n",
        mysql_insert_id());
?>
```

- `resource` **`mysql_list_dbs`** `([resource conn_id]);` Возвращает идентификатор результата для результирующего набора с именами баз данных, которые сервер относит к заданному соединению. В результирующем наборе имя каждой базы данных размещается в отдельной строке. Если в процессе выполнения возникает ошибка, эта функция возвращает значение "`false`". Устанавливать базу данных по умолчанию не следует. Результирующий набор впоследствии можно обработать с помощью любой функции выборки строк или функции `mysql_tablename()`. Для любой функции, принимающей параметр с номером строки, номер лежит в диапазоне от 0 до `mysql_num_rows()-1`. Для освобождения ресурсов, связанных с идентификатором результата, вызывайте функцию `mysql_free_result()`.

фикатором результата, его необходимо передать функции mysql_free_result().

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    $result_id = mysql_list_dbs ()
        or die ("Неуспешное завершение запроса \n");
    print ("Базы данных (using mysql_fetch_row()):<br />\n");
    while ($row = mysql_fetch_row ($result_id))
        printf ("%s<br />\n", $row[0]);
    mysql_free_result ($result_id);
    $result_id = mysql_list_dbs ()
        or die ("Неуспешное завершение запроса \n");
    print ("Базы данных (using mysql_db_name()):<br />\n");
    for ($i = 0; $i < mysql_num_rows ($result_id); $i++)
        printf ("%s<br />\n", mysql_db_name ($result_id, $i));
    mysql_free_result ($result_id);
?>
```

- resource **mysql_list_fields** (string db_name, string tbl_name [,resource conn_id]); Возвращает идентификатор для результирующего набора, содержащего информацию о столбцах таблицы. Если в процессе выполнения возникает ошибка, эта функция возвращает значение "false". Устанавливать базу данных по умолчанию не нужно. Параметры db_name и tbl_name определяют соответственно базу данных и таблицу, интересующую пользователя. Возвращаемый идентификатор может впоследствии использоваться для вызова функций mysql_field_flags(), mysql_field_len(), mysql_field_name() и mysql_field_type(). Для освобождения ресурсов, связанных с идентификатором результата, его необходимо передать функции mysql_free_result().

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    $result_id = mysql_list_fields ("sampdb", "member")
        or die ("Неуспешное завершение запроса \n");
    print ("информация о столбце таблицы member:<br />\n");
    for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
    {
        printf ("column %d:", $i);
        printf (" name %s,", mysql_field_name ($result_id, $i));
        printf (" len %d,", mysql_field_len ($result_id, $i));
        printf (" type %s,", mysql_field_type ($result_id, $i));
        printf (" flags %s\n", mysql_field_flags ($result_id, $i));
        print ("<br />\n");
    }
    mysql_free_result ($result_id);
?>
```

- resource **mysql_list_processes** ([resource conn_id]); Возвращает результирующий идентификатор для результирующего набора, содержащего информацию о текущих процессах, работающих на сервере, или "false", если имеет место ошибка. При наличии у пользователя привиле-

гии PROCESS список содержит все текущие процессы сервера. В противном случае он содержит информацию только о процессе пользователя.

Идентификатор результирующего набора может передаваться любой из обычных функций выборки строк и на mysql_free_result() для освобождения ресурсов, связанных с результирующим набором.

Функция mysql_list_processes() была добавлена в PHP 4.3.0.

- resource **mysql_list_tables** (string db_name [,resource conn_id]); Возвращает идентификатор результирующего набора со списком названий таблиц заданной базы данных. В результирующем наборе название каждой таблицы размещается в отдельной строке. Если в процессе выполнения возникает ошибка, эта функция возвращает значение "false". Устанавливать базу данных по умолчанию не нужно. Результирующий набор впоследствии можно обработать любой функцией выборки строк или с помощью функции mysql_tablename(). Для любой такой функции, которая принимает параметр с номером строки, номер лежит в диапазоне от 0 до mysql_num_rows()-1. Для освобождения ресурсов, связанных с идентификатором результата, его необходимо передать функции mysql_free_result().

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    $result_id = mysql_list_tables ("sampdb")
        or die ("Неуспешное завершение запроса \n");
    print ("sampdb tables (using mysql_fetch_row()):<br />\n");
    while ($row = mysql_fetch_row ($result_id))
        printf ("%s<br />\n", $row[0]);
    mysql_free_result ($result_id);
    $result_id = mysql_list_tables ("sampdb")
        or die ("Неуспешное завершение запроса \n");
    print ("sampdb tables (using mysql_tablename()):<br />\n");
    for ($i = 0; $i < mysql_num_rows ($result_id); $i++)
        printf ("%s<br />\n", mysql_tablename ($result_id, $i));
    mysql_free_result ($result_id);
?>
```

- int **mysql_query** (string query [,resource conn_id [, int fetch_mode]]); Отправляет строку запроса к серверу MySQL для заданного соединения. Для операторов DELETE, INSERT, REPLACE и UPDATE функция mysql_query() возвращает значение "true" в случае успешного выполнения и значение "false", если имела место ошибка. После успешного выполнения запроса можно вызвать функцию mysql_affected_rows(), чтобы узнать число измененных строк. Начиная с PHP 4.2.0 может задаваться необязательный параметр fetch_mode, чтобы указать, что функция mysql_query() должна выбрать и сохранить в буфере результирующий набор сразу же после выполнения или задержать выборку строк "на потом". Допустимые значения fetch_mode: MYSQL_STORE_RESULT (сохраняется в буфере) и MYSQL_USE_RESULT (не сохраняется в буфере), по умолчанию — MYSQL_STORE_RESULT. Использование значения MYSQL_USE_RESULT заставляет mysql_query() действовать как mysql_unbuffered_query(); подробнее об этом можно узнать из описания этой функции.

Для операторов SELECT функция mysql_query() возвращает положительный идентификатор результирующего набора в случае успешного выполнения и значение "false", если имела место ошибка. Возвращаемый после успешного выполнения запроса идентификатор результата может использоваться самыми разными функциями обработки результирующих наборов, считывающими аргумент result_id. Для освобождения занятых результирующих набором ресурсов системы этот идентификатор необходимо передать функции mysql_free_result().

Успешным считается запрос, который выполняется без единой ошибки. При этом на успех не влияет, возвращает ли данный запрос строки. Следующий запрос, например, является правильным, хотя и не возвращает никаких данных:

```
SELECT * FROM president WHERE 1 = 0
```

Ошибка при выполнении запроса может возникнуть по разным причинам. Так, она может быть обусловлена синтаксической ошибкой, смысловым несоответствием или недостаточностью полномочий для доступа к таблицам, указанным в запросе.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    print ("Успешное соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT * FROM president";
    $result_id = mysql_query ($query)
        or die ("Неуспешное завершение запроса \n");
    mysql_free_result ($result_id);
?>
```

- string **mysql_real_escape_string** (string str [, resource conn_id]); Выделяет любые кавычки или другие специальные символы в аргументе str, предваряя каждый из них обратной косой чертой. Результатом этой функции является новая строка, которая используется для подготовки вставки данных в запросы SQL. Функция mysql_real_escape_string() отличается от функции mysql_escape_string() тем, что принимает аргумент с идентификатором соединения и при выполнении декодирования учитывает текущую кодировку. Посмотрите раздел mysql_escape_string() в этом приложении.

  Перечень символов, выделяемых с помощью этой функции, см. в описании функции mysql_real_escape_string() в приложении E, "Программный интерфейс приложений C".

  Для выполнения функции mysql_real_escape_string() требуется MySQL 3.23.14 или выше. Функция mysql_escape_string() была добавлена в PHP 4.0.3. До этой версии для получения аналогичного результата использовалась функция mysql_escape_string() или addslashes().

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
    "secret")
```

```
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $last_name = mysql_real_escape_string ("O'Malley");
    $first_name = mysql_real_escape_string ("Brian");
    $query = "INSERT INTO member
(last_name,first_name,expiration)"
        . " VALUES('$last_name', '$first_name', '2002-6-3')";
    $result_id = mysql_query ($query)
        or die ("Неуспешное завершение запроса \n");
    printf ("членский номер для нового члена: %d\n",
mysql_insert_id());
?>
```

- `bool` **`mysql_select_db`** `(string db_name [, resource conn_id]);` Выборка указанной базы данных для того, чтобы сделать ее текущей базой данных для заданного соединения. Таблицы, на которые будут делаться ссылки в последующих запросах, по умолчанию будут принадлежать этой же самой базе данных, если не будет указана другая база данных.

  Возвращает значение "`true`" при успешном завершении и "`false`", если появилась ошибка.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
    "secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    print ("База данных выбрана успешно \n");
?>
```

- `int` **`mysql_unbuffered_query`** `(string query [, resource conn_id [, int fetch_mode]]);` Функция `mysql_unbuffered_query()` аналогична функции `mysql_query()`, но по умолчанию она не производит немедленной выборки строк результирующего набора в оперативную память. Различие этих двух функций аналогично различиям функций интерфейса C API `mysql_store_result()` и `mysql_use_result()`. Сравнение издержек и уместности применения этих двух функций см. в приложении E, "Программный интерфейс приложений C". Аналогичные соображения относятся и к функциям PHP `mysql_query()` и `mysql_unbuffered_query()`. В частности, некоторые из функций PHP работают правильно только с запросами, функционирующими в режиме буферизации. Это такие функции, как `mysql_data_seek()`, `mysql_num_rows()`, `mysql_result()`, и их псевдонимы.

  Начиная с PHP 4.2.0 может задаваться необязательный параметр `fetch_mode`, чтобы указать, что функция `mysql_unbuffered_query()` должна выбирать и сохранять в буфер результирующий набор сразу же после выполнения или отложить выборку строк "на потом". Допустимые значения параметра `fetch_mode`: `MYSQL_STORE_RESULT` (сохраняется в буфере) и `MYSQL_USE_RESULT` (не сохраняется в буфере), по умолчанию — `MYSQL_STORE_RESULT`. В режиме без буферизации все строки, возвращенные запросом, должны выбираться до запуска следующего запроса. Для ос-

вобождения ресурсов, связанных с идентификатором результата, его необходимо передавать функции mysql_free_result().

Функция mysql_unbuffered_query() была добавлена в PHP 4.0.6.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT first_name, last_name FROM member";
    $result_id = mysql_unbuffered_query ($query)
        or die ("Неуспешное завершение запроса \n");
    while (list ($first_name, $last_name)
        = mysql_fetch_row ($result_id))
    print ("$first_name $last_name\n");
    mysql_free_result ($result_id);
?>
```

Функции обработки результирующих наборов

Описанные в данном разделе функции используются для выборки результатов выполнения запросов, а также для предоставления информации о результатах, например количества измененных строк или метаданных для столбцов результирующего набора.

- int **mysql_affected_rows** ([resource conn_id]); Возвращает количество строк, измененных последним запросом DELETE, INSERT, REPLACE или UPDATE в заданном соединении. Функция mysql_affected_rows() возвращает значение 0, если ни одна строка изменена не была, и значение -1, если имела место ошибка.

  Вызванная после оператора SELECT функция mysql_affected_rows() возвращает число строк, полученных в результате выборки. Хотя обычно для операторов SELECT используется функция mysql_num_rows().

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "INSERT INTO member
(last_name,first_name,expiration)"
        . " VALUES('Brown', 'Marcia', '2002-6-3')";
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    $count = mysql_affected_rows ();
    printf ("%d row%s вставлено \n", $count, $count == 1 ? "" :
"s");
?>
```

- bool **mysql_data_seek** (resource result_id, int row_num); Каждый возвращаемый запросом SELECT результирующий набор содержит специальный курсор, указывающий на строку, которая будет возвращена при следующем вызове функций выборки строк (mysql_fetch_array(),

mysql_fetch_object() или mysql_fetch_row()). Функция mysql_data_seek() устанавливает курсор для заданного результирующего набора на заданную строку. Задаваемый номер должен находиться в диапазоне от 0 до mysql_num_rows()-1. Функция mysql_data_seek() возвращает значение "true", если задается разрешенный номер строки, и значение "false" во всех остальных случаях.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT last_name, first_name FROM president";
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    # fetch rows in reverse order
    for ($i = mysql_num_rows ($result_id) - 1; $i >= 0; $i--)
    {
        if (!mysql_data_seek ($result_id, $i))
        {
            printf ("Невозможно произвести выборку
                    строки %d\n", $i);
            continue;
        }
        if(!($row = mysql_fetch_object ($result_id)))
            continue;
        printf ("%s %s<br />\n", $row->last_name,
        $row->first_name);
    }
    mysql_free_result ($result_id);
?>
```

- string **mysql_db_name** (resource result_id, int row_num [, mixed field]);. Для заданного идентификатора результата, полученного с помощью функции mysql_list_dbs() и номера строки внутри результата, функция mysql_db_name() в заданной строке набора возвращает имя базы данных или значение "false" в случае возникновения ошибки. Для любой такой функции, которая принимает параметр с номером строки, номер лежит в диапазоне от 0 до mysql_num_rows()-1. Функция mysql_db_name() была добавлена в PHP 3.0.6. Фактически это псевдоним mysql_result().

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
    "secret")
        or die ("Невозможно установить соединение \n");
    $result_id = mysql_list_dbs ()
        or die ("Запрос не выполнен \n");
    print ("Базы данных:<br />\n");
    for ($i = 0; $i < mysql_num_rows ($result_id); $i++)
        printf ("%s<br />\n", mysql_db_name ($result_id, $i));
    mysql_free_result ($result_id);
?>
```

- array **mysql_fetch_array** (resource result_id [, int result_type]); Возвращает следующую строку данного результирующего

набора в виде массива. Возвращает значение "false", если строк больше не осталось. Возвращаемый массив содержит значения, идентифицируемые как по числовым индексам столбцов, так и по ключам имен столбцов. Другими словами, доступ к значению каждого столбца можно получить как по числовому индексу, так и по названию столбца. Учитывается регистр символов при написании ассоциативных индексов, поэтому их следует записывать в том же регистре, что и имена столбцов в запросе. Предположим, например, что отправлен следующий запрос:

```
SELECT last_name, first_name FROM president
```

Если строки извлекаются из результирующего набора в массив с именем $row, доступ к его элементам можно получить следующим образом.

`$row[0]`	Содержит значение столбца `last_name`.
`$row[1]`	Содержит значение столбца `first_name`.
`$row["last_name"]`	Содержит значение столбца `last_name`.
`$row["first_name"]`	Содержит значение столбца `first_name`.

Названия таблиц с соответствующими столбцами при определении ключей не используются, поэтому при выборе столбцов с одинаковыми именами из разных таблиц может произойти их наложение. Наивысшим приоритетом в этом случае будет обладать столбец, указанный последним в списке выбираемых запросом столбцов. Для доступа к скрытому под таким именем столбцу воспользуйтесь числовым индексом или напишите запрос, создающий уникальные псевдонимы для столбцов.

До PHP 3.0.7 функция `mysql_fetch_array()` возвращала значения как в виде цифровых, так и в виде ассоциативных индексов. Начиная с PHP 3.0.7 может указываться параметр `result_type`, который может иметь значения `MYSQL_ASSOC` (возвращаются значения только по индексам имен), `MYSQL_NUM` (возвращаются значения только по числовым индексам) или `MYSQL_BOTH` (возвращаются значения по индексам обоих типов). Если этот аргумент отсутствует, его значение по умолчанию устанавливается равным `MYSQL_BOTH`. Заметим, что вызов `mysql_fetch_array()` с параметром `result_type`, имеющим значение `MYSQL_ASSOC` или `MYSQL_NUM`, эквивалентно вызову `mysql_fetch_assoc()` или `mysql_fetch_row()`.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
    "secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT last_name, first_name FROM president";
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    while ($row = mysql_fetch_array ($result_id))
    {
    # печать каждого имени дважды, один раз с
    # помощью числовых индексов, а другой раз —
    # ассоциативных индексов (имен)
        printf ("%s %s<br />\n", $row[0], $row[1]);
```

```
        printf ("%s %s<br />\n", $row["last_name"],
        $row["first_name"]);
    }
    mysql_free_result ($result_id);
?>
```

- `array` **`mysql_fetch_assoc`** `(resource result_id);` Возвращает следующую строку данного результирующего набора в виде ассоциативного массива. Возвращает значение "`false`", если строк больше не осталось. Значения столбцов доступны с помощью ассоциативных имен, соответствующих именам столбцов, выбранных запросом.

  Вызов `mysql_fetch_assoc()` подобен вызову `mysql_fetch_array()` со вторым аргументом `MYSQL_ASSOC`. Значения столбцов с цифровыми индексами не возвращаются.

  Функция `mysql_fetch_assoc()` была добавлена в PHP 4.0.3.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT last_name, first_name FROM president";
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    while ($row = mysql_fetch_assoc ($result_id))
    {
        printf ("%s %s<br />\n", $row["last_name"],
$row["first_name"]);
    }
    mysql_free_result ($result_id);
?>
```

- `object` **`mysql_fetch_field`** `(resource result_id [, int col_num]);` Возвращает информацию метаданных о заданном столбце результирующего набора или значение "`false`", если заданный столбец отсутствует. Если параметр `col_num` пропущен, вызов функции `mysql_fetch_field()` приводит к возврату информации о следующих столбцах результирующего набора либо значения "`false`", если столбцов больше не осталось. Значение аргумента `col_num` должно находиться в диапазоне от 0 до `mysql_num_fields()-1`. Если это значение находится за пределами указанного диапазона, функция `mysql_fetch_field()` возвращает значение "`false`".

  Информация возвращается функцией в виде объекта со свойствами, описанными в табл. 3.2.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    $result_id = mysql_list_fields ("sampdb", "president")
        or die ("Запрос не выполнен \n");
    print ("Таблица: sampdb.president<br />\n");
    # получение метаданных столбца
    for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
    {
```

```
            printf ("Информация для столбца %d:<br />\n", $i);
            $meta = mysql_fetch_field ($result_id);
            if (!$meta)
            {
                print ("Информация отсутствует <br />\n");
                continue;
            }
            print ("<pre>\n");
            printf ("blob: %s\n", $meta->blob);
            printf ("def: %s\n", $meta->def);
            printf ("max_length: %s\n", $meta->max_length);
            printf ("multiple_key: %s\n", $meta->multiple_key);
            printf ("name: %s\n", $meta->name);
            printf ("not_null: %s\n", $meta->not_null);
            printf ("numeric: %s\n", $meta->numeric);
            printf ("primary_key: %s\n", $meta->primary_key);
            printf ("table: %s\n", $meta->table);
            printf ("type: %s\n", $meta->type);
            printf ("unique_key: %s\n", $meta->unique_key);
            printf ("unsigned: %s\n", $meta->unsigned);
            printf ("zerofill: %s\n", $meta->zerofill);
            print ("</pre>\n");
        }
        mysql_free_result ($result_id);
?>
```

Таблица 3.2. Свойства функции mysql_fetch_field()

Свойство	Значение
blob	1, если столбец имеет тип BLOB, иначе – 0
def	Значение столбца по умолчанию
max_length	Длина наибольшего значения столбца результирующего набора
multiple_key	1, если столбец является частью неуникального индекса, иначе – 0
name	Имя столбца
not_null	1, если столбец не может содержать значения NULL, иначе – 0
numeric	1, если столбец имеет числовой тип, иначе – 0
primary_key	1, если столбец является частью ключа PRIMARY KEY, иначе – 0
table	Имя таблицы, содержащей столбец (пустое для рассчитываемых столбцов)
type	Название типа столбца
unique_key	1, если столбец является частью индекса UNIQUE, иначе – 0
unsigned	1, если столбец имеет атрибут UNSIGNED, иначе – 0
zerofill	1, если столбец имеет атрибут ZEROFILL, иначе – 0

Элемент def объекта, возвращенного функцией mysql_fetch_field(), будет допустимым, только если результирующий набор был получен после вызова функции mysql_list_fields(). В противном случае будет возвращена пустая строка.

Элемент type будет одним из значений, перечисленных в табл. табл. 3.3. Для вычисляемого столбца значение type будет отображать тип выражения. Значение type, равное "unknown" скорее всего отражает то, что вер-

сия сервера MySQL новее, чем версия клиентской библиотеки MySQL, используемой PHP и то, что сервер "проинформирован" о новом типе, который библиотека не может распознать.

Таблица 3.3. Значения элементов type, возвращенные функцией mysql_fetch_field()

Значение	Тип столбца
blob	Столбец типа BLOB (или TEXT)
date	Столбец типа DATE
datetime	Столбец типа DATETIME
int	Столбец целого типа
null	Столбец, содержащий только значения NULL
real	Столбец вещественного типа
string	Столбец строкового типа, отличный от BLOB или TEXT
time	Столбец типа TIME
timestamp	Столбец типа TIMESTAMP
unknown	Столбец неизвестного типа
year	Столбец типа YEAR

- `array mysql_fetch_lengths (resource result_id);` Возвращает массив с длинами значений столбцов в строке, полученной последней, с помощью функции `mysql_fetch_array()`, `mysql_fetch_object()` или `mysql_fetch_row()`. Задаваемый индекс массива должен находиться в диапазоне от 0 до `mysql_num_rows()-1`. Эта функция возвращает значение "`false`", если еще не выбрана ни одна строка или произошла ошибка.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
    "secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT * FROM president";
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    $row_num = 0;
    while (mysql_fetch_row ($result_id))
    {
        ++$row_num;
        # получение длин значений столбцов
        printf ("Длины значений в строке %d:<br />\n", $row_num);
        $len = mysql_fetch_lengths ($result_id);
        if (!$len)
        {
            print ("Информация отсутствует <br />\n");
            break;
        }
        print ("<pre>\n");
        for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
            printf ("столбец %d: %s\n", $i, $len[$i]);
        print ("</pre>\n");
    }
```

```
    mysql_free_result ($result_id);
?>
```

- `object` **`mysql_fetch_object`** `(resource result_id [, int result_type]);` Возвращает следующую строку заданного результирующего набора в виде объекта либо значение "`false`", если строк больше не осталось. Доступ к значениям столбца осуществляется через свойства объекта. Соответственно, названия свойств аналогичны названиям столбцов, полученных из запроса, создающего результирующей набор.

  Параметр `result_type` может иметь значение `MYSQL_ASSOC` (значения возвращаются только по индексам имен), `MYSQL_NUM` (значения возвращаются только по числовым индексам) или `MYSQL_BOTH` (значения возвращаются по индексам обоих типов). Если значение этого параметра отсутствует, по умолчанию устанавливается `MYSQL_BOTH`.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT last_name, first_name FROM president";
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    while ($row = mysql_fetch_object ($result_id))
        printf ("%s %s<br />\n", $row->last_name, $row-
>first_name);
    mysql_free_result ($result_id);
?>
```

- `array` **`mysql_fetch_row`** `(resource result_id);` Возвращает следующую строку заданного результирующего набора в виде массива либо значение "`false`", если строк больше не осталось.

  Доступ к значениям столбцов можно получать через элементы массива, используя индексы столбцов в диапазоне от 0 до `mysql_num_fields()-1`.

  Вызов функции `mysql_fetch_row()` аналогичен вызову функции `mysql_fetch_array()` со вторым аргументом, равным `MYSQL_NUM`. Значения столбцов с ассоциативными индексами не возвращаются.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT last_name, first_name FROM president";
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    while ($row = mysql_fetch_row ($result_id))
        printf ("%s %s<br />\n", $row[0], $row[1]);
    mysql_free_result ($result_id);
?>
```

- `string` **`mysql_field_flags`** `(resource result_id, int col_num);` Возвращает строку с информацией метаданных о заданном столбце результи-

рующего набора. Если в процессе выполнения имела место ошибка, эта функция возвращает значение "false". Возвращаемая строка состоит из разделенных пробелами слов, отражающих, какие значения флагов столбцов истинны. Для флагов с ложными значениями слова в строке отсутствуют. В табл. 3.4 представлены все слова, которые могут включаться в возвращаемую строку.

```
Значение аргумента col_num должно находиться в диапазоне от 0 до
mysql_num_fields()-1.
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT * FROM member";
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
    {
        printf ("столбец %d:", $i);
        printf (" имя %s,", mysql_field_name ($result_id, $i));
        printf (" флаги %s\n", mysql_field_flags ($result_id,
$i));
        print ("<br />\n");
    }
    mysql_free_result ($result_id);
?>
```

Таблица 3.4. Значения функции `mysql_field_flags()`

Свойство	Значение
auto_increment	Столбец имеет атрибут AUTO_INCREMENT
binary	Столбец имеет атрибут BINARY
blob	Столбец имеет тип BLOB или TEXT
enum	Столбец имеет тип ENUM
multiple_key	Столбец является частью неуникального индекса
not_null	Столбец не может содержать значения NULL
primary_key	Столбец является частью ключа PRIMARY KEY
timestamp	Столбец имеет тип TIMESTAMP
unique_key	Столбец является частью индекса UNIQUE
unsigned	Столбец имеет атрибут UNSIGNED
zerofill	Столбец имеет атрибут ZEROFILL

Свойство "binary" устанавливается для столбцов строкового типа, чувствительных к регистру. Он включает столбцы, в которых ключевое слово BINARY задано явным образом (аналогично CHAR BINARY), так же как и столбцам BLOB.

Для того чтобы получить доступ к отдельным словам возвращенной функцией `mysql_field_flags()` строки, ее можно разбить, обработав с помощью функции `explode()`:

```
$words = explode (" ", mysql_field_flags ($result_id, $i));
```

- int **mysql_field_len** (resource result_id, int col_num); Возвращает максимально возможную длину значений заданного столбца результирующего набора.

```
Значение аргумента col_num должно находиться в диапазоне от 0 до
mysql_num_fields()-1.
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT * FROM member";
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
    {
        printf ("столбец %d:", $i);
        printf (" имя %s,", mysql_field_name ($result_id, $i));
        printf (" длина \n", mysql_field_len ($result_id, $i));
        print ("<br />\n");
    }
    mysql_free_result ($result_id);
?>
```

- string **mysql_field_name** (resource result_id, int col_num); Возвращает название заданного столбца результирующего набора.

```
Значение аргумента col_num должно находиться в диапазоне от 0 до
mysql_num_fields()-1.
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT * FROM president";
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    # получение названий столбцов
    for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
    {
        printf ("Наименование столбца %d: ", $i);
        $name = mysql_field_name ($result_id, $i);
        if (!$name)
            print ("Наименование отсутствует <br />\n");
        else
            print ("$name<br />\n");
    }
    mysql_free_result ($result_id);
?>
```

- int **mysql_field_seek** (resource resut_id, int col_num); Задает индекс для следующего вызова функции mysql_fetch_field(). Если при последующем вызове функции mysql_fetch_field() номер столбца не задан явно, возвращается информация для столбца col_num. Функция mysql_field_seek() возвращает значение "true", если поиск заданного столбца завершился удачно, и значение "false" при неудачном исходе.

Значение аргумента col_num должно находиться в диапазоне от 0 до mysql_num_fields()-1.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT * FROM president";
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    # get column metadata
    for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
    {
        printf ("Информация по столбцу %d:<br />\n", $i);
        if (!mysql_field_seek ($result_id, $i))
        {
            print ("Невозможно найти столбец <br />\n");
            continue;
        }
        $meta = mysql_fetch_field ($result_id, $i);
        if (!$meta)
        {
            print ("Информация отсутствует <br />\n");
            continue;
        }
        print ("<pre>\n");
        printf ("blob: %s\n", $meta->blob);
        printf ("max_length: %s\n", $meta->max_length);
        printf ("multiple_key: %s\n", $meta->multiple_key);
        printf ("name: %s\n", $meta->name);
        printf ("not_null: %s\n", $meta->not_null);
        printf ("numeric: %s\n", $meta->numeric);
        printf ("primary_key: %s\n", $meta->primary_key);
        printf ("table: %s\n", $meta->table);
        printf ("type: %s\n", $meta->type);
        printf ("unique_key: %s\n", $meta->unique_key);
        printf ("unsigned: %s\n", $meta->unsigned);
        printf ("zerofill: %s\n", $meta->zerofill);
        print ("</pre>\n");
    }
    mysql_free_result ($result_id);
?>
```

- `string` **`mysql_field_table`** `(resource result_id, int col_num);` Возвращает название таблицы, содержащей данный столбец результирующего набора. Для вычисляемых столбцов возвращается пустая строка. Значение аргумента col_num должно находиться в диапазоне от 0 до mysql_num_fields()-1.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT * FROM president";
```

```
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
    {
        printf ("столбец %d:", $i);
        printf (" имя %s,", mysql_field_name ($result_id, $i));
        printf (" таблица %s\n", mysql_field_table ($result_id, $i));
        print ("<br />\n");
    }
    mysql_free_result ($result_id);
?>
```

- `string` **`mysql_field_type`** `(resource result_id, int col_num);` Возвращает название типа заданного столбца результирующего набора. Список названий типов и их описание можно найти в описании типов, возвращаемых функцией `mysql_field_flags()`. Значение аргумента `col_num` должно находиться в диапазоне от 0 до `mysql_num_fields()-1`.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
     "secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT * FROM president";
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
    {
        printf ("столбец %d:", $i);
        printf (" имя %s,", mysql_field_name ($result_id, $i));
        printf (" тип %s\n", mysql_field_type ($result_id, $i));
        print ("<br />\n");
    }
    mysql_free_result ($result_id);
?>
```

- `bool` **`mysql_free_result`** `(resource result_id);` Освобождает системные ресурсы, занятые заданным результирующим набором, возвращенным одной из функций `mysql_query()`, `mysql_unbuffered_query()`, `mysql_list_dbs()`, `mysql_list_fields()`, `mysql_list_processes()` или `mysql_list_tables()`. Эти ресурсы автоматически освобождаются при завершении работы сценария. Поэтому необходимость в использовании этой функции может возникнуть только в том случае, когда рабочий сценарий создает большое количество результирующих наборов. Так, например, большой объем памяти потребуется для работы следующего сценария:

```
<?php
    for ($i = 0; $i < 10000; $i++)
    {
        $result_id = mysql_query ("SELECT * from president");
    }
?>
```

Добавление функции `mysql_free_resut()` после вызова `mysql_query()` позволяет значительно уменьшить объем памяти, выделяемой под результирующий набор, но практически неиспользуемой:

```
<?php
    for ($i = 0; $i < 10000; $i++)
    {
        $result_id = mysql_query ("SELECT * from president");
        mysql_free_result ($result_id);
    }
?>
```

Кроме того, необходимо явным образом вызывать функцию `mysql_free_result()` при преждевременной остановке цикла выборки после запроса с использованием функции `mysql_unbuffered_query()`. Функция `mysql_free_result()` позаботится о выборке и отмене оставшихся строк автоматически. Если сделать это невозможно, на следующем запросе произойдет ошибка "`out of sync`".

- `int mysql_insert_id ([resource conn_id]);` Возвращает значение AUTO_INCREMENT, сгенерированное последним запросом заданного соединения. Если в процессе всего соединения такое значение создано не было, эта функция возвращает нуль. В общем, функцию `mysql_insert_id()` следует вызывать сразу после запроса, который сгенерирует значение AUTO_INCREMENT. Если между этим запросом и вызовом функции будет выполнен промежуточный запрос, возвращаемое функцией значение может быть сброшено до нуля.

  Заметьте, однако, что функция `mysql_insert_id()` отличается от функции языка SQL `LAST_INSERT_ID()`, поскольку поддерживается на компьютере клиента и устанавливается для каждого запроса. Значение функции `LAST_INSERT_ID()`, напротив, поддерживается на сервере и постоянно для всех запросов.

  Возвращаемое функцией `mysql_insert_id()` значение относится к заданному соединению и не может изменяться значениями AUTO_INCREMENT для других соединений.

  Будьте осторожны при использовании этой функции для получения значений AUTO_INCREMENT для столбцов типа `BIGINT`. Диапазон значений типа `BIGINT` превышает диапазон значений, внутренне используемый PHP для такого типа при хранении возвращаемого значения `mysql_insert_id()`. Следует также быть осторожными с окружением PHP при получении этого значения с помощью запроса `SELECT LAST_INSERT_ID()`. Может оказаться, что использование результата в цифровом контексте приводит к преобразованию в вещественный тип значений.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
    "secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "INSERT INTO member
(last_name,first_name,expiration)"
        . " VALUES('Brown', 'Marcia', '2002-6-3')";
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    printf ("номер для нового члена: %d\n", mysql_insert_id());
?>
```

- int **mysql_num_fields** (resource result_id); Возвращает число столбцов в заданном результирующем наборе.

```
<?php
$conn_id = @mysql_connect ("cobra.snake.net", "sampadm", "secret")
or die ("Невозможно установить соединение \n");
mysql_select_db ("sampdb")
or die ("Невозможно выбрать базу данных \n");
$query = "SELECT * FROM president";
$result_id = mysql_query ($query)
or die ("Запрос не выполнен \n");
printf ("Количество столбцов: %d\n", mysql_num_fields
($result_id));
mysql_free_result ($result_id);
?>
```

- int **mysql_num_rows** (resource result_id); Возвращает число строк в заданном результирующем наборе.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
    "secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT * FROM president";
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    printf ("Количество строк: %d\n", mysql_num_rows ($result_id));
    mysql_free_result ($result_id);
?>
```

- mixed **mysql_result** (resource result_id, int row, mixed field); Возвращает значение строки row_num из заданной строки результирующего набора. Столбец идентифицируется параметром column, который может быть как числовым индексом столбца, так и именем столбца в формате *tbl_name.col_name*, где *tbl_name* — это имя таблицы, содержащей данный столбец. (В последнем случае вместо имени таблицы можно указать псевдоним *alias_name.col_name*.)

  Эта функция работает достаточно медленно, поэтому рекомендуется вместо нее использовать функции mysql_fetch_array(), mysql_fetch_object() или mysql_fetch_row(). Для улучшения производительности функции mysql_result() рекомендуется задавать цифровой индекс столбца, а не полное его имя.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    mysql_select_db ("sampdb")
        or die ("Невозможно выбрать базу данных \n");
    $query = "SELECT last_name, first_name FROM president";
    $result_id = mysql_query ($query)
        or die ("Запрос не выполнен \n");
    for ($i = 0; $i < mysql_num_rows ($result_id); $i++)
    {
        for ($j = 0; $j < mysql_num_fields ($result_id); $j++)
```

```
        {
        if ($j > 0)
            print (" ");
        print (mysql_result ($result_id, $i, $j));
        }
        print ("<br />\n");
    }
    mysql_free_result ($result_id);
?>
```

- `string` **`mysql_tablename`** `(resource result_id, int row_num);` Считывая идентификатор результата, полученный с помощью функции `mysql_list_tables()`, и номер строки, эта функция возвращает название, хранимое в заданной строке результирующего набора.

  Значение индекса строки должно находиться в диапазоне от 0 до `mysql_num_rows()-1`. Функция `mysql_tablename()` в действительности является псевдонимом функции `mysql_result()`.

```
<?php
    $conn_id = @mysql_connect ("cobra.snake.net", "sampadm",
"secret")
        or die ("Невозможно установить соединение \n");
    $result_id = mysql_list_tables ("sampdb")
        or die ("Запрос не выполнен \n");
    print ("таблицы sampdb:<br />\n");
    for ($i = 0; $i < mysql_num_rows ($result_id); $i++)
        printf ("%s<br />\n", mysql_tablename ($result_id, $i));
    mysql_free_result ($result_id);
?>
```

Информационные функции

Функции, приведенные в этом разделе, возвращают различную информацию о клиенте, сервере, версии протокола и текущем соединении. Следующий пример демонстрирует вызов этих функций:

```
<?php
  $conn_id = @mysql_connect ("cobra.snake.net", "sampadm", "secret")
    or die ("Невозможно установить соединение \n");
  $info = mysql_character_set_name ()
    or die ("Невозможно получить имя кодировки \n");
  print ("Имя кодировки: $info\n");
  $info = mysql_get_client_info ()
    or die ("Невозможно получить информацию о клиенте \n");
  print ("Информация о клиенте: $info\n");
  $info = mysql_get_host_info ()
    or die ("Невозможно получить информацию об узле \n");
  print ("Информация об узле: $info\n");
  $info = mysql_get_proto_info ()
    or die ("Невозможно получить информацию о протоколе \n");
  print ("Информация о протоколе: $info\n");
  $info = mysql_get_server_info ()
    or die ("Невозможно определить информацию о сервере \n");
  print ("Информация о сервере: $info\n");
  $info = mysql_stat ()
    or die ("Невозможно получить информацию о состоянии \n");
  print ("Информация о состоянии: $info\n");
```

```
  $info = mysql_thread_id ()
    or die ("Невозможно получить идентификатор процесса \n");
  print ("Идентификатор процесса: $info\n");
?>
```

Многие из этих функций соответствуют функциям интерфейса C API с аналогичным именем. Подробнее о них см. в приложении E, "Программный интерфейс приложений C".

- `string` **`mysql_get_client_info`** `(void);` Возвращает строку, отражающую версию клиентской библиотеки.

  Функция `mysql_get_client_info()` была добавлена в PHP 4.0.5.

- `string` **`mysql_get_host_info`** `([resource conn_id]);` Возвращает строку описания заданного соединения, например "Localhost via UNIX socket", "cobra.snake.net via TCP/IP" или ". via named pipe".

  Функция `mysql_get_host_info()` была добавлена в PHP 4.0.5.

- `int` **`mysql_get_proto_info`** `([resource conn_id]);` Возвращает число, отражающее версию протокола "клиент/сервер", используемую для данного соединения.

  Функция `mysql_get_proto_info()` была добавлена в PHP 4.0.5.

- `string` **`mysql_get_server_info`** `([resource conn_id]);` Возвращает строку, содержащую информацию о версии сервера, например "4.0.2-alpha-log". Значение состоит из номера версии с последующими одним или несколькими суффиксами (см. описание функции `VERSION()` в приложении B, "Операторы и функции").

  Функция `mysql_get_server_info()` была добавлена в PHP 4.0.5.

- `string` **`mysql_info`** `([resource conn_id]);` Возвращает информацию о запросах, обрабатывающих несколько строк. Формат строки см. в описании функции `mysql_info()` в приложении E, "Программный интерфейс приложений C".

  Функция `mysql_info()` была добавлена в PHP 4.3.0.

- `string` **`mysql_stat`** `([resource conn_id]);` Возвращает строку, содержащую информацию о статусе сервера. Формат этой строки см. в описании функции `mysql_stat()` в приложении E, "Программный интерфейс приложений C".

  Функция `mysql_stat()` была добавлена в PHP 4.3.0.

- `int` **`mysql_thread_id`** `([resource conn_id]);` Возвращает идентификатор процесса, который сервер связывает с данным соединением.

  Функция `mysql_thread_id()` была добавлена в PHP 4.3.0.

- `string` **`mysql_character_set_name`** `([resource conn_id]);` Возвращает имя стандартной кодировки для заданного соединения.

- Для функции `mysql_character_set_name()` требуется MySQL 3.23.21 или выше.

  Данная функция была добавлена в PHP 4.3.0.

Устаревшие функции

Описанные в этом разделе функции устарели и заменены альтернативными функциями, нашедшим отражение в описаниях функций.

- `bool` **`mysql_create_db`** `(string db_name [,resource conn_id]);` Указывает серверу MySQL, определенному идентификатором `conn_id`, создать базу данных с указанным именем. Эта функция возвращает значение "`true`", если создание базы данных завершено успешно, и значение "`false`", если имела место ошибка. Для создания базы данных пользователь должен обладать привилегией `CREATE`.

  Вместо функции `mysql_create_db()` рекомендуется применять функцию `mysql_query()` с оператором `CREATE DATABASE`.

- `resource` **`mysql_db_query`** `(string db_name, string query [,resource conn_id]);` Функция `mysql_db_query ()` подобна функции `mysql_query()` и отличается от нее лишь тем, что считывает дополнительный аргумент имени базы данных и делает заданную базу установленной по умолчанию перед выполнением запроса.

  Выбирать базу данных каждый раз при выводе запроса считается неэффективным, поэтому функцию `mysql_db_query()` лучше не использовать. Рекомендуется воспользоваться функцией `mysql_select_db()` для выбора базы данных и `mysql_query()` выпуска запроса после этого.

- `bool` **`mysql_drop_db`** `(string db_name [,resource conn_id]);` Указывает серверу MySQL, определенному идентификатором `conn_id`, удалить базу данных с указанным именем. Эта функция возвращает значение "`true`", если удаление базы данных завершено успешно, и значение "`false`", если произошла ошибка. Для удаления базы данных пользователь должен обладать привилегией удаления таблиц `DROP`.

  Эту функцию следует применять с большой осторожностью, поскольку после удаления базы данных восстановить ее будет невозможно.

  Вместо функции `mysql_drop_db()` рекомендуется применять функцию `mysql_query()` с оператором `DROP DATABASE`.

Язык PHP также поддерживает работу со старыми функциями для обеспечения обратной совместимости. Как видно в табл. 3.5, существуют псевдонимы, обеспечивающие связь с более новыми функциями. (Но обратите внимание, что новейшие функции тоже уже устарели.)

Таблица 3.5. Псевдонимы функций для обратной совместимости

Старое имя	Новое имя
`mysql_createdb()`	`mysql_create_db()`
`mysql_dbname()`	`mysql_db_name()`
`mysql_dropdb()`	`mysql_drop_db()`
`mysql_fieldflags()`	`mysql_field_flags()`
`mysql_fieldlen()`	`mysql_field_len()`
`mysql_fieldname()`	`mysql_field_name()`

Окончание табл. 3.5

Старое имя	Новое имя
mysql_fieldtable()	mysql_field_table()
mysql_fieldtype()	mysql_field_type()
mysql_freeresult()	mysql_free_result()
mysql_listdbs()	mysql_list_dbs()
mysql_listfields()	mysql_list_fields()
mysql_listtables()	mysql_list_tables()
mysql_numfields()	mysql_num_fields()
mysql_numrows()	mysql_num_rows()
mysql_selectdb()	mysql_select_db()

И

Провайдеры услуг Internet

В этом приложении...

Выбор провайдера
Оценка провайдера
Различные аспекты работы с MySQL

Сейчас пользователи имеют круглосуточный доступ к Internet и MySQL. В частности, такие возможности есть во многих университетах и больших компаниях, которые имеют собственные компьютерные отделы, хотя для многих пользователей подобные возможности еще недоступны, и доступ к Internet-службам они получают через провайдеров услуг Internet (Internet service providers). Помимо базового доступа к Internet для получения таких служб, как электронная почта и Web, многие провайдеры Internet предлагают и дополнительные услуги, например доступ к базам данных MySQL, Web-хостинг, такие программные языки, как Perl и PHP.

Это приложение представляет собой справочное руководство по выбору провайдера услуг Internet, который бы полностью соответствовал устанавливаемым пользователем критериям. Вполне возможно, что пользователям придется подключаться не только к Internet, но и к отдельному серверу, на котором установлена MySQL. Возможен и другой вариант, когда MySQL уже установлена на компьютере пользователя, и необходим лишь провайдер, который обеспечит подключение к этому компьютеру через Internet.

Один из способов найти подходящих провайдеров — посетить Web-узлы, посвященные инсталляционным пакетам, и посмотреть, не включают ли они страницы со списком провайдеров, поддерживающих нужные службы. Web-узел PHP, например, включает специальную страницу поиска, позволяющую найти поддерживающих службу PHP провайдеров Internet. Манипулируя критериями поиска,

можно найти и провайдеров, поддерживающих MySQL. Для поиска последних можно обратиться также по следующим адресам:

```
http://hosts.php.net/search.php
http://www.wix.com/mysql-hosting/
```

Каждый провайдер, как правило, имеет свои особенности, которые отличают его от остальных, поэтому перед окончательным выбором обязательно нужно провести сравнительный анализ. Необходимо с особой тщательностью подходить к выбору, поскольку плохой провайдер сведет на нет все преимущества использования MySQL. Помощь же хорошего провайдера, напротив, может быть просто неоценимой. Поэтому лучше не спешить и досконально проанализировать все имеющиеся варианты. Иначе можно остановить свой выбор на плохом провайдере, из-за чего впоследствии еще больше времени будет потрачено впустую.

Большинство описанных в этом приложении критериев следует оценивать без спешки. Вполне вероятно, что найти провайдера, полностью отвечающего установленным требованиям, будет невозможно. В такой ситуации следует поискать провайдера, который соответствует этим критериям.

Далеко не все провайдеры услуг Internet только и думают, как "содрать" с клиента побольше денег, предложив ему некачественные услуги. Поэтому необходимо будет провести небольшую "подготовительную работу", чтобы найти провайдера, который отлично знает свое дело и действительно помогает клиентам достичь их цели. Если речь идет о немалых инвестициях, следует с особой осторожностью относиться к провайдерам, которые уклоняются от ответов на вопросы об их коммерческой деятельности и технических возможностях. Ищите провайдера, который дает исчерпывающие ответы даже на самые необычные вопросы.



Вниманию провайдеров

MySQL представляет собой службу, которую также можно предложить клиентам, хотя затраты на нее гораздо ниже, чем на любую другую систему управления реляционными базами данных. Конечно, использовать MySQL можно и для своих внутренних потребностей (например, для хранения данных о клиентах). Читателям, которые сами предполагают инсталлировать и предлагать услуги по MySQL-хостингу, следует посмотреть на положение вещей с другой стороны. Другими словами, если где-то в тексте встречаются предложения типа "Спросите провайдера о том-то и том-то" или "Обеспечивает ли провайдер то-то и то-то", попытайтесь сами ответить на эти вопросы. Если описываемые услуги вами не предлагаются, возможно, пришло время задуматься о том, нужны ли они вообще.

Выбор провайдера

Первый этап выбора — поиск провайдера услуг, соответствующего потребностям пользователя. Для этого, собственно, необходимо сначала очень точно определить требования к необходимым службам и пропускной способности. Хотя, на первый взгляд, это кажется очевидным, но довольно многие пользователи выбирают провайдера, зная лишь, что они хотят "подключиться". Этой цели достичь довольно трудно, поскольку очень многие вопросы остаются нерешенными по той причине, что не были вовремя заданы. Подключиться для чего? Какие действия пользователь желает выполнять в Internet? Какую цену он готов за это заплатить?

Определив общие требования, можно задать выбранным провайдерам множество вопросов, что в итоге позволит провести маркетинг. В любом другом случае

полученная от провайдеров информация будет неполной, что не замедлит отразиться на правильности выбора.

В качестве одного из вариантов подобной оценки можно попытаться обзвонить провайдеров и выделить тех из них, чья служба продаж помогает определить, что же нужно вам на самом деле. Если сотрудники провайдера действительно помогают определить основные требования (а не пытаются предложить другие услуги, которые вам не нужны), значит, по меньшей мере есть один "плюс" — хорошая служба поддержки у провайдера уже есть.

Не следует также забывать, что некоторые провайдеры, предлагающие такие службы, как MySQL и Web-хостинг, не предоставляют сразу и услуги доступа по коммутируемой линии связи. Вполне возможно, что наилучшим вариантом будет выбор одного провайдера для подключения к Internet по коммутируемой линии, а второго — для работы с базами данных и Web-хостинга. (Более того, если пользователь уже работает с каким-либо провайдером услуг Internet, который полностью его устраивает, то остается лишь найти провайдера для управления базами данных и работы в Web.) В последующих двух разделах приведены основные виды вопросов, которые можно задать практически любому провайдеру.

Пропускная способность

Провайдеры зачастую предлагают самые разные варианты подключения: от выделенной линии (быстрый доступ) до коммутируемого модемного соединения. В общем, всегда существует компромисс между стоимостью и скоростью, ведь быстрый доступ стоит дороже. Соответственно, если планируется значительный обмен данными между компьютером пользователя и сервером провайдера, модемное соединение наверняка не подходит, так как оно является медленным. Иногда же и его может быть достаточно. Например, если большая часть поддержки, предоставляемой провайдером, заключается в обеспечении доступа клиентов к базе данных (например, через Web-узел), основная часть трафика будет происходить между провайдером и пользователями базы данных, что не требует наличия высокоскоростного соединения между владельцем базы данных и провайдером.

Службы

Очевидно, что предполагаемый кандидат в провайдеры должен обеспечивать работу службы MySQL, однако потребоваться могут также и следующие службы.

- **Электронный адрес.** Необходим для обмена данными со службой технической поддержки провайдера, а также может потребоваться для подключения к одному или нескольким спискам рассылки MySQL. Многие провайдеры предлагают несколько электронных адресов на одну учетную запись, что может оказаться очень кстати при наличии у вас наемных работников или при активном сотрудничестве с другими людьми.
- **Telnet-доступ к учетной записи.** Эта служба позволит запускать стандартные утилиты UNIX и работать из командной строки с такими программами MySQL, как `mysql`, `mysqldump` и `mysqlimport`. Кроме того, провайдеры иногда позволяют клиентам инсталлировать свое собственное программное обеспечение. Доступ к оболочке обычно осуществляется через утилиты SSH

(secure shell) и Telnet. SSH является более предпочтительным вариантом, чем Telnet, так как она безопаснее.

- **Дополнительные службы, связанные с MySQL.** К таким службам относятся Web-хостинг, PHP- и CGI-средства.
- **scp или FTP.** Эти службы весьма необходимы для обмена файлами между вашим компьютером и сервером провайдера. Так, например, вполне возможно, что со временем появится необходимость загрузить определенные файлы данных в базу данных для ее расширения или, напротив, извлечь определенные данные из базы для обработки их на локальном компьютере. Эти возможности можно реализовать именно с помощью FTP. Более того, можно даже позволить другим пользователям загружать свои собственные файлы в базу данных с помощью FTP.
- **Регистрация доменного имени и виртуальный хостинг.** Это позволит использовать адреса электронной почты и Web-узлов со своим доменным именем, а не именем провайдера. Особенно эта возможность эффективна, когда необходимо, чтобы устанавливаемый Web- или FTP-узел ассоциировался с конкретной организацией или видом деятельности. Почтовый адрес вида `you@yourbiz.com` является более информативным, чем `you@yourbiz.some-isp.com`, а URL `http://www.yourbiz.com` более конкретен, чем `http://yourbiz.some-isp.com`. Наличие личного домена также свидетельствует о том, что смена провайдера в будущем потребует меньше затрат. Если ваше имя привязано к имени провайдера, смена провайдера потребует изменить ваше имя на `you@yourbiz.other-isp.com` в электронном адресе и `http://yourbiz.other-isp.com` в URL вашего узла. Это нарушит стабильность, к которой стремятся люди, когда пытаются написать вам письмо или попасть на ваш узел.

Оценка провайдера

В этом разделе описываются критерии, позволяющие оценить возможности провайдера услуг Internet. Следующий раздел посвящен рассмотрению вопросов, непосредственно относящихся к MySQL. Итак, анализируя возможности того или иного провайдера, попытайтесь ответить на следующие вопросы.

- Насколько легко получить информацию?
- Какова стоимость услуг?
- Какое клиентское программное обеспечение предоставляет провайдер в пользование?
- Какую начальную поддержку или помощь предоставляет персонал провайдера?
- Какие виды подключения предлагает провайдер клиентам?
- Обеспечивает ли провайдер качественную связь и высокую пропускную способность?
- Какие виды технической поддержки обеспечивает провайдер?
- Существуют ли какие-либо ограничения? Если они существуют, то как это может отразиться на вашей работе?

- Какие аппаратные средства использует провайдер?
- Обеспечивает ли провайдер защиту информации и другие меры безопасности?
- Обладает ли провайдер хорошей репутацией?
- Как долго он работает на рынке телекоммуникационных услуг?
- Какие дальнейшие планы у данного провайдера?

А теперь рассмотрим каждый из этих вопросов отдельно.

- **Насколько легко получить информацию?** Сразу ли сотрудники компании-провайдера отвечают на все заданные вопросы по телефону? Существуют ли трудности с дозвоном, либо по указанному телефону ответил автоответчик? Перезвонили ли впоследствии представители компании?

  Ответил ли представитель отдела продаж на все заданные вопросы? Если не ответил, то каким образом данный вопрос был закрыт: кто-то другой из этой организации ответил на вопрос или сотрудники компании проигнорировали его. Если сотрудники компании не ответили на заданные вопросы до того, как были заплачены деньги, то как они поведут себя после оплаты?

  Общается ли персонал компании с клиентами на "понятном" языке? Если собеседник болтает о совершенно не относящихся к делу вещах без устали, то наверняка весьма мало знаком с технической терминологией. Однако очень трудно разговаривать, если собеседник использует совсем непонятный профессиональный жаргон и незнакомые термины.

  Посетите Web-узел провайдера. Является ли его страница информативной и понятной? Если провайдер не в состоянии сделать хорошую собственную страницу, то нет никакой гарантии, что свои услуги он обеспечивает на высоком уровне.

- **Какова стоимость услуг?** Стоимость предлагаемых услуг может в значительной степени изменяться, поскольку зависит от большого количества факторов. Узнайте, существует ли у провайдера, помимо ежемесячной оплаты, еще и плата за подключение или установку? Является ли ежемесячная плата фиксированной или "плавающей"? Время доступа не ограничено, или существует определенный лимит? Что произойдет, если этот лимит будет превышен? Обрывает ли провайдер соединение (что весьма неудобно, но позволяет избежать дополнительной платы), или к ежемесячному счету будет прибавлена дополнительная оплата? Включается ли в стоимость оплаченных услуг предоставление определенного дискового пространства?

  Для коммутируемых соединений провайдеры услуг Internet, как правило, выделяют локальные номера внутри определенного региона.

  Включают ли эти номера код 800, который можно использовать бесплатно за пределами региона? Существует ли дополнительная плата за такой номер? Пользователи, проживающие в сельской местности, имеют более ограниченный выбор и должны обязательно убедиться, что доступ к провайдеру не будет слишком дорогим.

  В основную стоимость провайдеры зачастую включают предоставление небольшого объема свободного дискового пространства (несколько мегабайт).

За дополнительный объем приходится платить отдельно. Постарайтесь получить всю информацию о стоимости дискового пространства у провайдера, если планируется работать с большой базой данных или пользоваться услугами Web-хостинга (особенно если пользовательские страницы включают большой объем графической информации).

Если же речь идет об установке Web- или FTP-узла на ресурсах провайдера услуг Internet, обязательно нужно поинтересоваться, какую плату берет провайдер за объем данных, который внешние клиенты загружают с Web-страниц? Некоторые провайдеры устанавливают определенные ограничения на такой объем и в случае его превышения запрещают дальнейшую загрузку до наступления следующего отчетного периода. Другие провайдеры вводят переменную ставку за определенную единицу объема. Постарайтесь проанализировать все альтернативы, чтобы выбрать наиболее подходящий для вас вариант оплаты.

Необходимо ли платить за оказание технической поддержки со стороны провайдера? Если эта поддержка предоставляется бесплатно, то не ограничена ли она во времени, например только первыми 30 днями работы? Если же за поддержку необходимо платить, то взимается плата постоянно или за каждое отдельное обращение? Существует ли возможность получать дополнительную поддержку за отдельную плату?

С особой осторожностью следует относиться к провайдерам, требующим полной предоплаты за все будущие услуги. Оплата такого рода иногда используется для увеличения оборотного капитала начинающими компаниями, и вероятность разорения таких компаний намного выше, что, собственно, и повышает риск невозврата денег.

Подписывается договор на обслуживание на долгий срок, или он будет краткосрочным? Не рекомендуется заключать долгосрочный договор, не владея полной информацией о возможностях провайдера. Ведь пользователь должен иметь возможность отказаться от услуг данного провайдера. Подписать договор не трудно. Труднее впоследствии его разорвать.

Некоторые провайдеры предоставляют дешевую или бесплатную тестовую учетную запись, которая позволяет оценить уровень возможностей и технической поддержки провайдера. Следует помнить, что такая учетная запись накладывает определенные ограничения на операции. Так, например, небольшой объем выделяемого пространства ограничивает размер создаваемой тестовой базы данных.

- **Какое клиентское программное обеспечение предоставляет провайдер?** Поддерживает ли провайдер используемую пользователем платформу? Некоторые провайдеры сразу сообщают, что они поддерживают только определенные платформы, например Windows. Такие провайдеры не подходят для пользователей, чьи программы работают под управлением ОС UNIX или Mac OS.

  Предоставляет ли провайдер при необходимости средства почтового клиента и Web-броузера? Необходимо ли для этих целей использовать определенные программы, предоставленные провайдером, или можно использовать свои?

- **Какую начальную поддержку или помощь предоставляет персонал провайдера?** Оказывает ли персонал компании-провайдера помощь в подключении и активизации учетной записи? Делается это бесплатно или за дополнительную плату (при наличии высоких требований у пользователя последнее более вероятно).
- **Какие виды подключения предлагает провайдер клиентам?** Мало кто захочет при подключении через модем и установлении соединения с провайдером постоянно слышать сигнал "занято". Поэтому узнайте у потенциальных провайдеров, какое у них соотношение количества клиентов к числу модемов. Хотя, надо признать, свобода доступа зависит еще и от активности клиентов. Частные лица, подключающиеся ненадолго для проверки своей электронной почты, конечно же, не так сильно загружают модемный пул, как коммерческие пользователи, которые целый день работают в сети с целью расширения бизнеса. Лучше спросить, как часто дозванивающиеся клиенты слышат сигнал "занято", особенно в то время суток, когда вы планируете подключаться.

  Вполне возможно, что даже приобретение высокоскоростного модема ничего не даст, так как провайдер не сможет поддерживать такую скорость либо ограничит пользователя еще более низкой скоростью. Узнайте у провайдера, в состоянии ли он предложить такие высокоскоростные варианты подключения, как DSL? Если между установленным узлом и сервером провайдера планируется передавать огромные объемы данных, возможно, есть смысл задуматься об установке выделенной линии. Может ли провайдер оказать помощь в подобной установке?

  При наличии всегда подключенной линии появляется возможность поддержки входящих подключений к вашему компьютеру и, таким образом, размещения своего собственного Web-узла на этом компьютере. Но для того, чтобы это все работало, другие пользователи должны убедиться в его надежности и подключиться к вашему серверу. Многие провайдеры сейчас обеспечивают динамические IP-адреса по умолчанию и могут фильтровать ваш входящий трафик. Статический IP-адрес является обязательным условием при необходимости поддерживать входящие соединения, поскольку для пользователей будет очень трудно добраться до вас, если ваш адрес периодически меняется. Вам также потребуется проверить то, что входящие соединения к портам, используемым вашими серверам, не будут заблокированы. Например, для провайдеров обычно достаточной является фильтрация портов SMTP, HTTP и FTP. (Обычно это делается с целью предотвращения попыток пользователей установить серверы и изменить пропускную способность.) Есть провайдеры, предлагающие услугу статической IP-адресации без фильтрации, но это может стоить дороже; провайдеры хорошо знают, что пользователи, которые заказывают такой тип соединения, обычно имеют и больший трафик.
- **Обеспечивает ли провайдер качественную связь и высокую пропускную способность?** Насколько близко расположен провайдер к магистральному Internet-каналу? Подключен ли он к этому каналу напрямую или через другого провайдера? Имеет ли компания-провайдер резервные каналы на случай сбоя?

Какова пропускная способность канала у провайдера услуг Internet? Тип соединения определяет *пропускную способность* (*bandwidth*), т.е. максимальный объем трафика, который в состоянии пропускать провайдер. Линии OC3, T-3 и T-1 являются высокоскоростными и обеспечивают быстрые соединения. Если же внешние соединения провайдера установлены через модемы, то скорости передачи окажутся на порядок ниже.

Оцените соотношение пропускной способности и пользовательского пула, который обслуживается данным провайдером. Обязательно спросите представителей провайдера, какой трафик они на самом деле передают через свои линии и насколько загружен общий канал провайдера? Если общее соединение с магистральным каналом делится на количество активных пользователей, доступная каждому из них пропускная способность будет со временем уменьшаться. В свою очередь, полностью загруженное соединение замедлит работу пользователей, подключающихся к установленным базам данных или Web-узлу, если таковой имеется.

- **Какие виды технической поддержки обеспечивает провайдер?** Техническая поддержка является круглосуточной или обеспечивается только в рабочее время? Конечно же, большинство коммерческих пользователей подключаются только в рабочее время. Однако если пользователь пожелает подключиться в нерабочее время, кто ответит на возникающие у него вопросы? Можно ли связаться с персоналом отдела технической поддержки по телефону, по электронной почте? Последняя возможность может оказаться весьма кстати, например, когда можно не объяснять специалистам все по телефону, а просто отправить по почте техническую информацию или распечатку программы. Обещает ли служба поддержки ответить на запрос в течение определенного периода времени? (Возможно, проблема в столь короткий срок и не может быть решена, однако ответ на запрос дает понять, что персонал предпринимает попытки ее решить.)

  Предоставляют ли сотрудники службы помощь в реальном масштабе времени? Понятна ли она? Легко ли найти необходимую информацию и поддержку в Internet?

  Проверить доступность технической поддержки можно так же, как и доступность телефонных номеров провайдера, — позвонить им. Возможно, и не совсем разумно пытаться дозвониться в службу поддержки до получения счета, однако это позволит на собственном примере убедиться, как быстро сотрудники обслуживают клиентов, трудно ли к ним дозвониться, и т.п. Задайте вопрос относительно количества постоянно работающих сотрудников. А еще лучше, попытайтесь узнать соотношение числа клиентов к числу сотрудников отдела поддержки.

- **Существуют ли какие-либо ограничения? Если они существуют, то как это может отразиться на вашей работе?** Устанавливает ли провайдер какие-либо ограничения на объем доступного пространства и время обработки? Лимитируется ли время обработки сценариев Web-сервером? Некоторые провайдеры действительно устанавливают определенные ограничения на деятельность клиентов, чтобы исключить захват одним клиентом ресур-

сов, совместно используемых большим количеством пользователей. И настоятельно рекомендуется сразу определиться с размерами подобных ограничений. Что происходит при достижении ограничения? Уведомляет ли провайдер специально об этом или без предварительного уведомления начинает высчитывать с вас за перерасход, о чем вы с удивлением узнаете из следующего счета?

Накладывает ли провайдер ограничение на размеры сообщений электронной почты? Если вам приходится постоянно отсылать или получать большие вложения, убедитесь в том, что для них хватит места и они не будут усечены в момент прохождения сервера?

- **Какие аппаратные средства использует провайдер?** Простейшие Internet-службы могут работать и на старых компьютерах с 386-процессором, однако для большинства пользователей подобного доступа недостаточно, не говоря уже о таких интенсивно использующих компьютерные ресурсы системах, как MySQL. Кроме того, MySQL – далеко не самая интенсивно использующая ресурсы система управления базами данных, однако все равно стоит остановить свой выбор на провайдерах, обладающих достаточным резервом мощностей аппаратных ресурсов. Попытайтесь узнать, какой объем загрузки провайдер может обрабатывать без особых проблем?

  Где размещается сервер провайдера? Ведь если оборудование провайдера расположено в деловой части города, он сможет в случае необходимости гораздо быстрей расширить свои телефонные каналы.

  Что собирается делать персонал провайдера в случае сбоя в работе оборудования? Имеется ли на этот случай какой-нибудь план восстановления? Или пользователям придется ожидать, пока поврежденное оборудование не будет исправлено или заменено? Какими значениями отказоустойчивости характеризуется работа провайдера в последние несколько месяцев? Как оплачиваются такие инциденты? Выполняют ли сотрудники компании-провайдера резервирование и как часто? Или, может, провайдер считает, что за резервирование отвечает пользователь?

  Бывают ли в работе провайдера простои, обусловленные, например, выполнением процедуры резервирования или поддержки? Информирует ли провайдер своих клиентов о таких простоях заблаговременно? Разумеется, что запустившие Web-сервер клиенты захотят, чтобы их страницы были доступны 24 часа в сутки.

- **Обеспечивает ли провайдер защиту информации и другие меры безопасности?** Следуют ли сотрудники компании-провайдера какой-либо политике защиты файлов клиентов? Какие действия предпринимаются персоналом для защиты учетных записей и личных данных?

- **Обладает ли провайдер хорошей репутацией?** Какую репутацию имеет провайдер? Какие еще клиенты обслуживаются у потенциального провайдера и насколько они удовлетворены его работой? Весьма полезными могут оказаться отзывы знакомых и друзей о работе конкретного провайдера. Подобные отзывы можно найти и в списке рассылки MySQL. Кроме того, даже провайдер может направить в случае необходимости вас к своим клиентам,

но это, скорее всего, будут клиенты, которые удовлетворены уровнем предлагаемых им услуг. В почтовом же списке можно увидеть как положительные, так и отрицательные отзывы. Опытные пользователи с удовольствием расскажут, как они работали с тем или иным провайдером.

- **Как долго он работает на рынке телекоммуникационных услуг?** Как долго провайдер работает в телекоммуникационном бизнесе? Индустрия Internet-услуг помнит множество впечатляющих взлетов и быстрых падений, поскольку в ней достаточно легко начать бизнес и весьма трудно удержаться на плаву. Провайдер с "долгой историей" гораздо надежнее новичка, а следовательно, с ним можно планировать долгосрочное сотрудничество.
- **Какие дальнейшие планы у данного провайдера?** Насколько велика компания-провайдер? Не каждый захочет пользоваться услугами небольшой компании-провайдера, однако и огромные компании иногда оказываются не самым лучшим вариантом. Ведь небольшие компании быстрее перестраиваются под потребности своих клиентов.

  Насколько быстро развивается компания-провайдер? Сколько клиентов она обслуживает в данный момент? А сколько обслуживала месяц назад? Конечно же, желательно подключаться к Internet с помощью быстро развивающейся компании, однако быстрый рост числа клиентов оказывает значительное влияние на пропускную способность, качество доступа клиентов и технической поддержки. Как провайдер планирует справиться с быстро возрастающей нагрузкой?

Не ожидайте от провайдера многого

Не следует думать, что можно получить неограниченный доступ, неограниченное дисковое пространство и неограниченную техническую поддержку за 20 долларов в месяц. Согласны ли вы платить большие деньги за качественное обслуживание и поддержку? Ведь провайдер желает окупить средства, вложенные в предоставляемые пользователям услуги.

Различные аспекты работы с MySQL

В предыдущем разделе рассматривались и анализировались возможности провайдеров услуг Internet. Настоящий раздел посвящен критериям оценки MySQL-служб провайдеров. В последующем материале под MySQL следует понимать не только собственно MySQL, но также все другие связанные с этой системой службы, например Apache, Perl и PHP.

- Какие связанные с MySQL службы предлагает провайдер?
- Какую версию MySQL использует провайдер?
- Какова политика провайдера в отношении обновления MySQL?
- Существуют ли какие-либо проблемы в работе MySQL, связанные с установленной у провайдера операционной системой?
- Могут ли клиенты инсталлировать свое собственное программное обеспечение?
- Заботится ли провайдер о защите и безопасности баз данных клиентов?
- Оказывает ли провайдер помощь на начальном этапе работы?

- Какие виды связанной с MySQL технической поддержки предоставляются провайдером?
- Сколько клиентов провайдера используют службу MySQL?

Теперь рассмотрим каждый из этих вопросов более детально.

- **Какие связанные с MySQL службы предлагает провайдер?** Прежде всего, потенциальный провайдер должен установить у себя MySQL и все необходимые для работы этой СУБД пакеты. Исключение из этого правила могут составлять лишь провайдеры, которые предоставляют все остальные услуги, кроме MySQL, и полностью удовлетворяют клиентов для подключения к Internet. Такого провайдера достаточно спросить, планирует ли он в будущем инсталлировать MySQL. Ведь нельзя не согласиться, что MySQL — не настолько распространенная служба, как электронная почта или Web-хостинг.

  Посетите Web-страницу провайдера. Посмотрите, предлагается ли на ней подробная информация о предоставляемых службах MySQL? Можно ли из стиля изложения сделать вывод, что персонал компании действительно разбирается в MySQL, или предоставленная информация больше похожа на рекламу? Легко ли перемещаться по Web-узлу компании? Включает ли Web-страница провайдера специальный раздел с ответами на наиболее часто задаваемые клиентами вопросы о предлагаемых услугах? Такие разделы иногда содержат весьма полезную информацию.
- **Какую версию MySQL использует провайдер?** Какая версия установлена у провайдера: одна из последних или очень старая, не обновляемая с момента установки? Обратитесь к разделу справочного руководства по MySQL, посвященному изменению версий, и узнайте, какие возможности были добавлены в более новых, чем использует провайдер, версиях? Необходимы ли эти возможности вам?
- **Какова политика провайдера в отношении обновления MySQL?** Персонал компании-провайдера вынужден постоянно балансировать между проверенной стабильностью давно используемой версии MySQL и желанием некоторых клиентов использовать новые возможности самых последних версий. Эту дилемму разрешить не так легко, поэтому не стоит относиться к политике провайдера с предубеждением. Однако по крайней мере поинтересуйтесь, можно ли поработать с более новыми версиями MySQL, запущенными, например, на отдельном компьютере в режиме тестирования. Подобное предложение может заинтересовать провайдера, поскольку некоторые клиенты действительно сами вызываются на роль "подопытного кролика" с целью тестирования новых версий программных продуктов.
- **Существуют ли какие-либо проблемы в работе MySQL, связанные с установленной у провайдера операционной системой?** Обратитесь к главе справочного руководства по MySQL и узнайте, существуют ли какие-либо ограничения, связанные с операционной системой, которые могут отрицательно отразиться на вашей работе? Иногда сотрудники компании-провайдера сами предлагают способы устранения подобных проблем.

- **Могут ли клиенты инсталлировать свое собственное программное обеспечение?** Это позволит расширить возможности MySQL с помощью утилит независимых разработчиков и собственных программ.
- **Заботится ли провайдер о защите и безопасности баз данных клиентов?** Использует ли провайдер, помимо общих мероприятий и средств, направленных на защиту данных клиентов, какие-либо дополнительные меры защиты данных MySQL? Выделяется ли каждому пользователю отдельный сервер MySQL? С точки зрения клиента подобная услуга весьма ценна, поскольку появляется возможность отследить, кто получает доступ к содержимому базы данных. Для провайдера же она означает лишь увеличение объемов работ и увеличивает нагрузку на ресурсы сервера. Аналогичные факторы являются существенными для Web-сервера. Значительно безопаснее иметь свой собственный сервер, но это приводит к возрастанию нагрузки на него. Поэтому здесь придется решать проблему выбора между недорогой общей базой данных и дорогим выделенным только для вас сервером.
- **Оказывает ли провайдер помощь на начальном этапе работы?** Не следует ожидать, что персонал компании-провайдера будет учить клиентов основам MySQL и показывать, как писать SQL-запросы. Но по крайней мере желательно, чтобы сотрудники компании помогли подключиться к серверу MySQL, сообщив всю необходимую для этого информацию. Хорошо бы было также, чтобы они предоставили простой сценарий, позволяющий проверить правильность доступа к базе данных.
- **Какие виды технической поддержки, связанной с MySQL, предоставляются провайдером?** В настоящее время немало пользователей отправляют в почтовый список сообщения о проблемах со своими провайдерами и неправильной работе MySQL. Замечено, что основным источником проблем провайдеры всегда считают клиента. Однако, как показывает опыт, ошибок достаточно с обеих сторон, и зачастую оказывается, что сам провайдер плохо понимает, как работать с MySQL, и его персонал решает появляющиеся проблемы методом "научного тыка". Поэтому узнайте, имеется ли в компании специальный отдел, занимающийся поддержкой пользователей MySQL. Попытайтесь узнать уровень знаний сотрудников этого отдела. Если провайдер хорошо знаком с пакетом, необходимым для работы клиентов, он сможет оказать ценную помощь. Если же его сотрудники не могут дать конкретного ответа, можно сделать вывод, что они лишь поверхностно ознакомились с материалом части III этого издания, которая целиком посвящена проблемам администрирования MySQL.

  Провайдер должен быть знаком с материалом, изложенным в главе 3, "Синтаксис и использование языка SQL". Узнайте, имеют ли они контракт на поддержку MySQL (это свидетельствует о том, что они воспринимают MySQL всерьез и всегда готовы помочь в разрешении возникающих проблем) или ограничиваются рекомендацией "Об этом можно узнать в списке рассылки"? Уровень технической поддержки должен соответствовать уровню управления ресурсов, который провайдер предоставляет клиенту. Ведь если клиент сам хочет иметь полный контроль над своим сервером MySQL, про-

вайдер может предоставить ему такой контроль. Однако со своей стороны он вполне обоснованно ожидает, что в этом случае вся ответственность за администрирование сервера ложится на клиента. Клиент же может обратиться к провайдеру за сценарием, который позволит автоматически запускать сервер MySQL в процессе загрузки системы, что исключит необходимость запуска сервера вручную каждый раз при перезагрузке компьютера.

- **Сколько клиентов провайдера используют службу MySQL?** Активное использование службы MySQL большим количеством пользователей является наилучшим доказательством качества предоставляемых услуг и хорошей технической поддержки со стороны провайдера.

Остерегайтесь неопытных администраторов

Настоятельно рекомендуется избегать провайдеров, инсталлирующих MySQL исключительно для расширения списка предоставляемых услуг с целью привлечения клиентов. Очень трудно впоследствии решить проблемы с администратором, обладающим минимальным объемом знаний в области выполнения задач администрирования.

Предметный указатель

C

CGI-программы, *309*

O

ODBC (Open Bata Base Connectivity), *125*

W

Web-сервер Apache, *309*

A

Автоматическая разблокировка сервера, *283*
Административные функции, *958*
Аргумент Perl
  %attr, *967*
Арифметические операторы, *733*
Архитектура
  СУБД MySQL, *47*
  клиент/сервер, *47*
Атрибуты
  AUTO_INCREMENT, *129*; *721*
    в таблицах BDB, *154*
    в таблицах HEAP, *154*
    в таблицах InnoDB, *155*
    в таблицах ISAM, *150*
    в таблицах MyISAM, *152*
  BINARY, *141*; *159*
  CHARACTER SET, *141*
  DEFAULT, *143*
  NOT NULL, *60*; *143*
  NULL, *60*; *143*
  PrintError, *384*; *387*
  RaiseError, *384*; *387*
  UNSIGNED, *128*; *129*; *721*
  ZEROFILL, *128*; *720*
  дескрипторов баз данных, *984*
    AutoCommit, *984*
    Statement, *984*
  дескрипторов общие
    ChopBlanks, *984*
    FetchHashKeyName, *984*
    HandleError, *984*
    PrintError, *985*
    RaiseError, *985*
    ShowErrorStatement, *985*
    TraceLevel, *985*
дескрипторов операторов, *986*
    NAME, *987*
    NAME_hash, *987*
    NAME_lc, *987*
    NAME_lc_hash, *987*
    NAME_uc, *987*
    NAME_uc_hash, *987*
    NULLABLE, *987*
    NUM_OF_FIELDS, *987*
    NUM_OF_PARAMS, *987*
    SCALE, *988*
    Statement, *988*
динамические
    err, *990*
    errstr, *990*
    rows, *990*
интерфейса DBI, *983*
метаданных
    mysql_max_length, *414*
операторов
    PRECISION, *988*
специальные дескрипторов баз данных
    mysql_errno, *985*
    mysql_error, *985*
    mysql_hostinfo, *985*
    mysql_info, *985*
    mysql_insertid, *985*
    mysql_protoinfo, *986*
    mysql_serverinfo, *986*
    mysql_stat, *986*
    mysql_thread_id, *986*
специальные дескрипторов операторов
    mysql_insertid, *988*
    mysql_is_auto_increment, *989*
    mysql_is_blob, *989*
    mysql_is_key, *989*
    mysql_is_num, *989*
    mysql_is_pri_key, *989*
    mysql_length, *989*
    mysql_max_length, *989*
    mysql_store_result, *989*
    mysql_table, *989*
    mysql_type, *989*
    mysql_type_name, *989*
    mysql_use_result, *989*
    TYPE, *988*

Б

Библиотека
  libmysqld, *316*; *874*
Битовые операции, *739*

В

Вектор аргументов, *328*
Версия, *705*
Вещественное число, *115*
Вещественный тип, *118*; *121*
  DECIMAL, *122*
  DOUBLE, *122*
  FLOAT, *122*
Включаемые файлы PHP, *473*
Внешний ключ, *249*
Восстановление
  после сбоя, *521*
  таблиц баз данных, *681*
Временные таблицы, *206*
Выбор
  диапазона допустимых значений, *169*
  типа данных столбца, *166*
  типа столбца, *165*
Выборка полных результирующих наборов данных, *398*
Вызов сценария
  mysql.server, *557*
  mysqld_safe, *557*
Выражение SET, *73*

Г

Генерация последовательностей, *158*
Глобальные параметры баз данных, *198*
Группа
  [client], *329*; *331*
  mysqlgrp, *704*

Д

Дата и время , *117*
Двоичная строка, *131*
Действия после
  инсталляции MySQL, *548*
  первой инсталляции MySQL, *548*
Декартово произведение, *226*
Дескриптор
  $SIG{__DIE__}, *388*
  "$dbh->", *967*
  "$h->", *967*
  "$sbh->", *967*
  соединения, *321*
  языка Perl, *381*
Диапазон значений, *126*
Динамические атрибуты, *990*
Дистрибутив
  ActiveState Perl, *716*
  в RPM-файлах, *702*
  в двоичном формате, *702*
  в исходных текстах, *702*
Длина
  имени, *194*
  отображаемых значений, *126*
Допустимые символы, *194*
Другие методы дескрипторов баз данных, *977*

Ж

Журнал
  медленных запросов, *578*
  обновлений, *539*
  общих запросов, *578*

З

Завершение работы сервера, *554*
Запись, *44*
Запуск сервера, *519*; *554*
  MySQL непривилегированным пользователем, *555*; *584*; *652*
Зарезервированные слова, *120*
Защита
  сервера, *521*
  файловой системы, *521*
Значение NULL, *80*

И

Изменение структуры базы данных, *198*
Индекс, *212*; *264*
  PRIMARY KEY, *214*; *721*
  UNIQUE, *214*; *721*
  обычный (неуникальный), *213*
  типа FULLTEXT, *135*; *200*; *213*
  уникальный, *213*
Индексирование, *174*
Индексно-последовательный метод доступа, *199*
Инициализация каталога данных и таблиц разрешений, *708*
Инсталляция
  MySQL в Windows, *713*
  RPM-дистрибутива, *705*
  двоичного дистрибутива, *705*
  дистрибутива в исходных текстах, *707*
  модуля PHP, *701*
  поддержки Perl DBI, *710*

СУБД MySQL, *700*
Интегрированная среда разработки, *319*
Интерпретация календарных данных
функция TO_DAYS, *186*
Информационные функции, *956*; *1019*
Исключенные функции, *960*

К

Календарные типы, *119*
Каталог данных, *524*
MySQL, *519*; *527*
Квалификатор имени, *194*
Клиентская программа mysql, *49*
Ключевое слово
BINARY, *173*
DELAYED (операторы INSERT и REPLACE), *284*
DISTINCT, *91*
EXPLAIN (оператором SELECT), *271*
HIGH_PRIORITY (оператор SELECT), *284*
LOW_PRIORITY (операторы DELETE, INSERT, LOAD DATA, REPLACE и UPDATE), *283*
PROMPT, *111*
Кодировка, *117*; *141*; *160*
Unicode, *162*
UTF-8, *164*
на уровне
базы данных, *160*; *162*
сервера, *160*; *162*
столбцов, *160*
таблицы, *163*
строковых констант, *160*
таблиц, *160*
таблицы, *163*
Команды
chmod, *709*
chown, *709*
configure, *707*
CREATE TABLE, *120*
make, *318*
mysql, *888*
clear, \c, *888*
connect, \r, *888*
edit, \e, *888*
ego, \G, *888*
exit, *888*
go, \g, ;, *888*
help, \h, ?, *888*
nopager, \n, *888*
notee, \t, *888*
pager, \P, *888*
print, \p, *888*
prompt, \R, *889*
quit, \q, *889*
rehash, \#, *889*
source file_name, \. file_name, *889*
status, \s, *889*
system, \!, *889*
tee, \T, *889*
use, \u, *889*
mysqladmin, *892*
create, *893*
debug, *893*
drop, *893*
extended-status, *893*
flush-hosts, *893*
flush-logs, *893*
flush-privileges, *893*
flush-status, *893*
flush-tables, *893*
flush-threads, *893*
kill, *893*
password, *893*
ping, *893*
processlist, *894*
refresh, *894*
reload, *894*
shutdown, *894*
start-slave, *894*
status, *894*
stop-slave, *894*
variables, *894*
version, *894*
mysqladmin variables, *524*
mysqlshow, *222*
OPTIMIZE TABLE, *172*
Комментарии, *260*
языка Perl, *383*
Компиляция, *317*
и редактирование связей клиентской программы, *317*
Конструкция
AS name, *85*
qq{}, *402*
языка Perl
switchbox, *421*
Конфигурационный файл, *328*; *868*
.my.cnf, *410*
Apache, *713*
my-example.cnf, *868*
Курсор, *262*

Л

Левое объединение, *227*
Логические операторы, *740*
Логическое

И (AND или &&), *741*
ИЛИ (OR или ||), *741*
отрицание (NOT или !), *740*

M

Макросы, *936*
IS_BLOB, *936*
IS_NOT_NULL, *936*
IS_NUM, *358*; *936*
IS_PRI_KEY, *936*
Массив
client_groups[], *331*
Метаданные результирующего набора, *355*
Методы
дескрипторов
баз данных, *972*
общих, *981*
операторов, *977*
интерфейса DBI, *966*
класса DBI, *968*
языка Perl, *381*
available_drivers, *968*
begin_work, *972*
bind_col, *406*; *977*
bind_columns, *407*; *977*
bind_param, *977*
commit, *973*
connect, *384*; *968*
connect_cached, *971*
data_sources, *971*
defined, *401*
die, *387*
disconnect, *973*
do, *390*; *973*
dump_results, *978*
err, *981*
errstr, *981*
escape, *446*
escapeHTML, *446*
execute, *385*; *978*
fetch, *978*
fetchall_arrayref, *399*; *978*
fetchall_hashref, *979*
fetchrow_array, *385*; *392*; *979*
fetchrow_arrayref, *394*; *980*
fetchrow_hashref, *395*; *980*
finish, *386*; *980*
func действие, *982*
createdb, *982*
dropdb, *982*
reload, *982*
shutdown, *982*
install_driver, *972*
map, *393*
ping, *974*
prepare, *385*; *974*
prepare_cached, *974*
quote, *403*; *974*
rollback, *974*
rows, *980*
selectall_arrayref, *400*; *974*
selectall_hashref, *975*
selectcol_arrayref, *975*
selectrow_array, *397*; *976*
selectrow_arrayref, *976*
selectrow_hashref, *976*
trace, *412*; *981*
trace_msg, *981*
warn, *387*
Модуль
CGI.pm, *441*; *443*
DBI, *42*
mod_perl, *309*
mod_php, *309*

H

Настройка сервера, *520*
Научная нотация чисел, *115*
Недвоичная строка, *131*
Недостатки индексирования, *267*
Неиндексированная таблица, *264*
Непосредственный вызов mysqld, *557*
Нескалярные типы данных, *932*
MYSQL, *932*
MYSQL_FIELD, *932*
Несовместимость типов, *134*
Нулевое значение, *143*

O

Обновление MySQL, *521*
Обработка ошибок в языке Perl, *386*
Общие атрибуты дескрипторов, *984*
Общий
журнал запросов, *539*
регистрационный журнал, *539*
Объединение
CROSS_JOIN, *226*
JOIN, *226*
LEFT OUTER JOIN, *229*
NATURAL LEFT JOIN, *229*
STRAIGHT_JOIN, *226*
Объекты языка Perl, *381*
Оператор, *732*
%, *734*
*, *734*
/, *734*
_charset, *742*

+, *733*
ALTER DATABASE, *162*; *197*; *198*; *784*
ALTER TABLE, *162*; *166*; *214*; *217*; *784*
предложение CHANGE, *218*
предложение MODIFY, *218*
спецификатор action, *218*
ANALYZE TABLE, *788*
BACKUP TABLE, *788*
BEGIN, *245*; *789*
BINARY, *178*; *742*
CASE, *737*
CHANGE MASTER, *789*
CHECK TABLE, *790*
COLLATE, *162*; *742*
COMMIT, *245*; *791*
CREATE DATABASE, *55*; *197*; *791*
CREATE FUNCTION, *791*
CREATE INDEX, *198*; *214*; *281*; *792*
CREATE TABLE, *59*; *162*; *204*; *793*
CREATE TABLE ... SELECT, *207*
CREATE TABLE IF NOT EXISTS, *205*
CREATE TEMPORARY TABLE, *206*
DELETE, *103*; *799*
DESCRIBE, *221*; *800*
DO, *801*
DROP DATABASE, *197*; *198*; *801*
DROP FUNCTION, *801*
DROP INDEX, *198*; *216*; *281*; *801*
DROP TABLE, *211*; *802*
EXPLAIN, *221*; *271*; *802*
FLUSH, *803*
GRANT, *567*; *804*
HANDLER, *808*
IN, *738*
INSERT, *72*; *279*; *808*
IS NOT NULL, *738*
IS NULL, *738*
KILL, *809*
LIKE, *89*; *179*; *743*
LOAD ... FROM MASTER, *814*
LOAD DATA, *72*; *279*; *810*
LOAD DATA LOCAL, *74*; *279*
LOCK TABLES, *283*; *814*
NOT IN, *738*
NOT LIKE, *89*; *179*
NOT REGEXP, *179*
NOT RLIKE, *746*
OPTIMIZE TABLE, *136*; *815*
PURGE MASTER LOGS, *816*
REGEXP, *179*; *180*; *743*; *744*
RENAME TABLE, *816*
REPAIR TABLE, *816*
REPLACE, *817*
RESET, *817*
RESTORE TABLE, *818*
REVOKE, *576*; *818*
RLIKE, *180*; *746*
ROLLBACK, *245*; *819*
SELECT, *75*; *819*
SET, *246*; *825*
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL, *830*
SHOW, *831*
SHOW [FULL] COLUMNS, *832*
SHOW [GLOBAL | SESSION] VARIABLES, *844*
SHOW BINLOG EVENTS, *831*
SHOW CHARACTER SET, *162*; *831*
SHOW COLUMN TYPES, *832*
SHOW CREATE DATABASE, *833*
SHOW CREATE TABLE, *833*
SHOW DATABASES, *833*
SHOW GRANTS, *833*
SHOW INDEX, *834*
SHOW INNODB STATUS, *835*
SHOW LOGS, *835*
SHOW MASTER LOGS, *835*
SHOW MASTER STATUS, *835*
SHOW PRIVILEGES, *835*
SHOW PROCESSLIST, *836*
SHOW SLAVE HOSTS, *836*
SHOW SLAVE STATUS, *836*
SHOW STATUS, *837*
SHOW TABLE STATUS, *842*
SHOW TABLES, *843*
SHOW TABLES TYPES, *843*
SLAVE, *855*
SQL, *784*
switch, *495*
TRUNCATE, *855*
UNION, *234*; *856*
UNLOCK TABLES, *283*; *856*
UPDATE, *104*; *856*
USE, *197*; *857*
группировки, *733*
сравнения, *735*
Операционная система, *43*
Операция объединения, *98*
Оптимизатор запросов СУБД MySQL, *271*
Основные требования к системе, *317*
Основы языка PHP, *467*
Останов сервера, *519*
Отладка с применением трассировки, *412*
Отмена привилегий пользователя, *576*
Отсутствие предложения FROM в операторе SELECT, *732*

П

Параллельный доступ, *47*
Параметры, *120*
  --datadir= dir_name, *524*
  AUTO_INCREMENT, *114*; *118*
  AUTOCOMMIT, *246*
  BINARY, *120*
  --character-sets-dir, *865*
  --compress, -C, *865*
  D, *124*
  --datadir, -h, *902*
  --debug, -#, *865*
  DEFAULT, *121*
  --default-character-set, *161*; *866*
  DELAYED_KEY_WRITE, *282*
  --help, -?, *866*
  --host, -h, *866*
  --host=host_name (mysqladmin), *525*
  isamchk и myisamchk
    --analyse, -a, *876*
    --block-search, -b, *876*
    --description, -d, *876*
    --extend-check, -e, *876*
    --force, -f, *876*
    --information, -i, *876*
    --keys-used, -k, *876*
    --no-symlinks, -l, *876*
    --quick, -q, *877*
    --recover, -r, *877*
    --safe_recover, -o, *877*
    --sort-index, -S, *877*
    --sort-records, -R, *877*
    --unpack, -u, *877*
    --wait, -W, *877*
  M, *124*
  max_allowed_packet, *136*
  myisamchk
    --backup, -B, *877*
    --check, -c, *878*
    --check-only-changed, -C, *878*
    --correct-checksum, *878*
    --data-file-length, -D, *878*
    --fast, -F, *878*
    --medium-check, -m, *878*
    --parallel-recover, -p, *878*
    --read-only, -T, *878*
    --set-auto-increment, -A, *878*
    --set-character-set, *878*
    --sort-recover, -n, *879*
    --start-check-pos, *879*
    --tmpdir, -t, *879*
    --update-state, -U, *879*
  myisampack и pack_isam
    --backup, -b, *881*
    --force, -f, *881*
    --join, -j, *881*
    --test, -t, *881*
    --tmpdir, -T, *881*
    --wait, -W, *881*
mysld_safe
    --core-file-size, *915*
    --err-log, *915*
    --ledir, *915*
    --mysqld, *916*
    --open-files-limit, *915*
    --timezone, *916*
mysqadmin
    --count, -c, *892*
    --force, -f, *892*
    --relative, -r, *892*
    --sleep, -i, *892*
    --timeout, -t, *892*
    --vertical, -E, *892*
    --wait, -W, *892*
mysql
    --auto-rehash, *883*
    --batch, B, *884*
    --column-names, *884*
    --database, -D, *884*
    --debug-info, -T, *884*
    --execute, -e, *884*
    --force, -f, *884*
    --html, -H, *884*
    --i-am-a-dummy, *884*
    --ignore-space, -i, *884*
    --line-numbers, *885*
    --local-infile, *885*
    --named-commands, -G, *885*
    --no-auto-rehash, -A, *885*
    --no-beep, -b, *885*
    --no-named-commands, -g, *885*
    --no-pager, *885*
    --no-tee, *885*
    --one-database, -o, *885*
    --pager, *886*
    --prompt, *886*
    --quick, -q, *886*
    --raw, -r, *886*
    --safe-updates, -U, *886*
    --skip-column-names, -N, *886*
    --skip-line-numbers, -L, *886*
    --table, -t, *886*
    --tee, *887*
    --unbuffered, -n, *887*
    --vertical, -E, *887*
    --wait, -W, *887*
    --xml, -X, *887*
mysql_config
    --cflags, *899*

--embedded, --embedded-libs, *899*
--libmysqld-libs, *899*
--libs, *899*
--port, *899*
--socket, *899*
--version, *899*
mysql_install_db
--basedir, *926*
--datadir, *926*
--force, *926*
--user, *927*
mysqlaccess
--brief, -b, *890*
--commit, *890*
--copy, *891*
--db, -d, *891*
--debug, *891*
--howto, *891*
--old_server, *891*
--plan, *891*
--preview, *891*
--relnotes, *891*
--rhost, -H, *891*
--rollback, *891*
--spassword, -P, *891*
--superuser, -U, *891*
--table, -t, *891*
mysqlbinlog
--database, -d, *895*
--offset, -o, *895*
--position, -j, *895*
--result-file, -r, *895*
--short-form, -s, *895*
--table, -t, *895*
mysqlcheck
--all-databases, -A, *897*
--all-in-1, -1, *897*
--analyze, -a, *897*
--auto-repair, *897*
--check, -c, *897*
--check-only-changed, -C, *897*
--databases, -B (boolean), *897*
--extended, -e, *897*
--fast, -F, *897*
--force, -f, *897*
--medium-check, -m, *897*
--optimize, -o, *898*
--quick, -q, *898*
--repair, -r, *898*
--tables, *898*
--use-frm, *898*
mysqld
--ansi, -a, *901*
--basedir, -b, *901*
--bdb-home, *908*
--bdb-lock-detect, *908*
--bdb-logdir, *908*
--bdb-no-recover, *908*
--bdb-no-sync, *909*
--bdb-shared-data, *909*
--bdb-tempdir, *909*
--big-tables, *901*
--bind-address, *901*
--bootstrap, *901*
--chroot, -r, *901*
--concurrent-insert, *901*
--console, *907*
--core-file, *901*
--default-table-type, *902*
--delay-key-write, *902*
--des-key-file, *902*
--enable-locking, *902*
--enable-named-pipe, *908*
--enable-pstack, *902*
--exit-info, -T, *902*
--external-locking, *902*
--flush, *902*
--init-file, *903*
--innodb_data_file_path, *909*
--innodb_data_home_dir, *909*
--innodb_fast_shutdown, *909*
--innodb_flush_log_at_trx_commit, *909*
--innodb_flush_method, *910*
--innodb_log_arch_dir, *910*
--innodb_log_archive, *910*
--innodb_log_group_home_dir, *910*
--install, *908*
--install-manual, *908*
--language, -L, *903*
--local-infile, *903*
--log, -l, *903*
--log-bin, *903*
--log-bin-index, *903*
--log-isam, *903*
--log-long-format, *904*
--log-slow-queries, *903*
--log-update, *904*
--low-priority-updates, *904*
--memlock, *904*
--myisam-recover, *904*
--new, -n, *904*
--old-passwords, *904*
--old-protocol, -o, *904*
--one-thread, *905*
--pid-file, *905*
--remove, *908*
--safe, *905*
--safemalloc-mem-limit, *905*
--safe-show-database, *905*
--safe-user-create, *905*

--skip-bdb, *909*
--skip-concurrent-insert, *905*
--skip-grant-tables, -Sg, *905*
--skip-host-cache, *906*
--skip-innodb, *910*
--skip-locking, *906*
--skip-name-resolve, *906*
--skip-networking, *906*
--skip-new, *906*
--skip-safemalloc, *906*
--skip-show-database, *906*
--skip-stack-trace, *906*
--skip-symlink, *906*
--skip-thread-priority, *906*
--sql-bin-update-same, *906*
--sql-mode, *906*
ANSI_QUOTES, *906*
IGNORE_SPACE, *906*
NO_UNSIGNED_SUBTRACTION, *906*
ONLY_FULL_GROUP_BY, *907*
PIPES_AS_CONCAT, *907*
REAL_AS_FLOAT, *907*
SERIALIZE, *907*
--standalone, *908*
--temp-pool, *907*
--tmpdir, -t, *907*
--transaction-isolation, *907*
--use-symbolic-links, *908*
--warnings, *907*
mysqld_multi
--config-file, *914*
--example, *914*
--log, *914*
--mysqladmin, *914*
--mysqld, *915*
--no-log, *915*
--tcp-ip, *915*
mysqldump
--add-drop-table, *917*
--add-locks, *917*
--all, -a, *917*
--all-databases, -A, *917*
--allow-keywords, *917*
--complete-insert, -c, *917*
--databases, -B, *918*
--delayed-insert, *918*
--disable-keys, -K, *918*
--extended-insert, -e, *918*
--fields-enclosed-by, *920*
--fields-escaped-by, *920*
--fields-optionally-enclosed-by, *920*
--fields-terminated-by, *920*
--first-slave, -x, *918*
--flush-logs, -F, *918*
--force, -f, *918*
--lines-terminated-by, *920*
--lock-tables, -l, *918*
--master-data, *918*
--no-autocommit, *918*
--no-create-db, -n, *918*
--no-create-info, -t, *918*
--no-data, -d, *918*
--opt, *918*
--quick, -q, *919*
--quote-names, -Q, *919*
--result-file, -r, *919*
--single-transaction, *919*
--tab, -T, *919*
--tables, *920*
--where, -W, *920*
--xml, -X, *920*
mysqlhotcopy
--allowold, *922*
--checkpoint, *922*
--dryrun, -n, *922*
--flushlog, *922*
--keepold, *922*
--method, *922*
--noindices, *922*
--quiet, -q, *923*
--record_log_pos, *923*
--regexp, *923*
--resetmaster, *923*
--resetslave, *923*
--suffix, *923*
--tmpdir, *923*
mysqlimport
--columns, *924*
--delete, -d, *924*
--fields-enclosed-by, *925*
--fields-escaped-by, *925*
--fields-optionally-enclosed-by, *925*
--fields-terminated-by, *926*
--force, -f, *924*
--ignore, -i, *925*
--ignore-lines, *925*
--lines-terminated-by, *926*
--local, -L, *925*
--lock-tables, -l, *925*
--low-priority (boolean), *925*
--replace, -r, *925*
mysqlshow
--keys, -k, *928*
--status, -i, *928*
NOT NULL, *121*
NULL, *121*; *129*
pack_isam
--packlength, -p, *881*
PACK_KEYS=1, *200*
--password, -p, *866*

--pipe, -W, *866*
--port и --socket (mysqladmin), *525*
--port, -P, *866*
--quick, *161*
QUICK (оператор REPAIR TABLE), *161*
--recover, *161*
--setcharacter-set, *161*
--set-variable, -O, *866*
--silent, -s, *866*
--socket, -S, *867*
--ssl, *867*
--ssl-ca, *867*
--ssl-capath, *867*
--ssl-cert, *867*
--ssl-cipher, *867*
--ssl-key, *867*
UNSIGNED, *118*; *120*; *124*; *165*
--user, -u, *867*
--verbose, -v, *867*
--version, -V, *867*
ZEROFILL, *121*
драйвера MySQL
host, *969*
mysql_client_found_rows, *969*
mysql_compression, *969*
mysql_connect_timeout, *969*
mysql_local_infile, *969*
mysql_read_default_file, *969*
mysql_read_default_group, *970*
mysql_socket, *970*
mysql_ssl, *971*
mysql_ssl_ca_file, *971*
mysql_ssl_ca_path, *971*
mysql_ssl_cipher, *971*
mysql_ssl_client_cert, *971*
mysql_ssl_client_key, *971*
port, *969*
конфигурационные файлы
--defaults-extra-file, *869*
--defaults-file, *869*
--no-defaults, *869*
--print-defaults, *870*
программы
длинная форма, *863*
короткая форма, *863*
репликация
--abort-slave-event-count, *910*
--binlog-do-db, *910*
--binlog-ignore-db, *910*
--disconnect-slave-event-count, *910*
--init-rpl-role, *911*
--log-slave-updates, *911*
--master-connect-retry, *911*
--master-host, *911*
--master-info-file, *911*
--master-password, *911*
--master-port, *911*
--master-retry-count, *911*
--master-ssl, *911*
--master-ssl-cert, *911*
--master-ssl-key, *911*
--master-user, *911*
--max-binlog-dump-events, *912*
--old-rpl-compat, *912*
--reckless-slave, *912*
--relay-log, *912*
--relay-log-index, *912*
--relay-log-info-file, *912*
--replicate-do-db, *912*
--replicate-do-table, *912*
--replicate-ignore-db, *912*
--replicate-ignore-table, *912*
--replicate-rewrite-db, *912*
--replicate-wild-do-table, *913*
--replicate-wild-ignore-table, *913*
--server-id, *913*
--show-slave-auth-info, *913*
--skip-slave-start, *913*
--slave-load-tmpdir, *913*
--slave-skip-errors, *913*
--sporadic-binlog-dump-fail, *913*
форматирования данных, *917*; *919*; *920*; *924*; *925*
функции mysql_real_connect, *321*
Перекрестное объединение, *226*
Переменные
$DBI
err, *387*
errstr, *387*
$ENV{HOME}, *408*
DBI_DSN, *407*
DBI_PASS, *407*
DBI_USER, *407*
myisamchk
ft_max_word_len, *879*
ft_max_word_len_for_sort, *879*
ft_min_word_len, *880*
myisam_block_size, *880*
myisamchk и isamchk
decode_bits, *879*
key_buffer_size, *879*
read_buffer_size, *879*
sort_buffer_size, *879*
sort_key_blocks, *879*
write_buffer_size, *879*
mysqadmin
connect_timeout, *892*
shutdown_timeout, *892*
mysql, *887*
connect_timeout, *887*

max_allowed_packet, *887*
max_join_size, *887*
net_buffer_length, *887*
select_limit, *887*
MYSQL_DEBUG, *873*
MYSQL_HISTFILE, *873*
MYSQL_HOST, *873*
MYSQL_PS1, *874*
MYSQL_PWD, *873*
MYSQL_TCP_PORT, *873*
MYSQL_UNIX_PORT, *873*
mysqld, *913*
mysqldump, *920*
max_allowed_packet, *921*
net_buffer_length, *921*
PHP
$php_errormsg, *999*
TMPDIR, *873*
USER, *873*
среды, *873*
интерфейса DBI, *990*
Переносимость, *202*
BDB, *203*
InnoDB, *203*
ISAM, *203*
MERGE, *203*
MyISAM, *203*
Платформа, *705*
Побитовые операции
& (пересечение), *739*
| (объединение), *739*
<< (сдвиг влево), *739*
>> (сдвиг право), *740*
XOR (исключающее, *739*
Подавление механизма оптимизации
СУБД MySQL, *274*
Поддержка
регистрационных файлов, *519*
учетных записей пользователей, *519*
Полное объединение, *225*
Пользователь
mysql, *704*
mysqladm, *704*
Порождающая таблица, *249*
Порожденная таблица, *249*
Последовательность, *967*
Правила
индексирования, *268*
написания выражений, *175*
присвоения имен базам данных и таблицам, *532*
Правое объединение, *227*
Превентивная поддержка, *522*
Предложение
HAVING, *96*
IF NOT EXISTS, *205*
IS NOT NULL, *80*
IS NULL, *80*
LEFT JOIN, *100*
LIMIT, *83*
ORDER BY, *81*
ASC, *81*
DESC, *81*
WHERE, *77*
Преимущества индексирования, *264*
Преобразование типа таблицы, *220*
Преобразование типов
данных при выполнении операций, *183*
функция
FLOOR, *188*
функция CHAR, *188*
функция CONVERT, *188*
функция DATE_ADD, *188*
функция HEX, *188*
функция ROUND, *188*
Пример сценария
dump_members.pl, *382*
dump_members2.pl, *389*
Приоритет операторов, *732*
Проверка
пустых значений, *400*
таблиц баз данных, *681*
Программа
isamchk, *518*; *548*; *874*
load_defaults, *331*
myisamchk, *161*; *518*; *548*; *874*
myisampack, *880*
mysql, *518*; *882*
mysql_config, *899*
mysqladmin, *518*; *548*; *891*
mysqlbinlog, *894*
mysqlcheck, *896*
mysqld, *518*; *548*; *899*
mysqld_safe, *518*
mysqldump, *222*; *518*; *548*; *916*
mysqldump, *161*
mysqlhotcopy, *548*
mysqlimport, *924*
mysqlshow, *927*
pack_isam, *880*
ppm, *716*
Производительность СУБД MySQL, *171*
Проиндексированная таблица, *264*
Простой метод аутентификации, *507*
Протокол
CGI (Common Gateway Interface), *309*
SSL (Secure Sockets Layer), *316*
Процесс выбора кодировки, *142*
Псевдоним, *194*
Пустое значение, *118*

(NULL), *117*
Пустые аргументы
функция IFNULL, *182*
функция STRCMP, *182*

P

Распаковка дистрибутива, *705*
Расширение
.php, *466*
.phtml, *466*
Редактирование связей, *317*
Резервирование и копирование баз данных, *519*
Результирующий набор, *345*
Реляционная система управления базами данных (relational database management system, RDBMS), *44*
Репликация баз данных, *520*

C

Сервер
Apache, *700*
Синтаксис CHARACTER SET charset, *724*
Синтаксический анализ аргументов командной строки, *331*
Система управления, *44*
Скалярные типы данных, *931*
my_bool, *931*
my_ulonglong, *931*
MYSQL_FIELD_OFFSET, *931*
MYSQL_ROW_OFFSET, *932*
Смена типов столбцов, *133*
Создание
баз данных, *197*
и настройка учетных записей пользователей MySQL, *565*
индексов, *213*
таблиц, *204*
Специальные
административные методы MySQL, *981*
атрибуты дескрипторов
баз данных MySQL, *985*
операторов MySQL, *988*
параметры
`mysld_safe`, *915*
`mysql_install_db`, *926*
`mysqld`, *901*
`mysqld_multi`, *914*
`mysqldump`, *917*
`mysqlhotcopy`, *922*
`mysqlimport`, *924*
символы SQL-образцов, *743*
Сравнение значений столбцов, *173*
Ссылки языка Perl, *381*
Ссылочная целостность, *249*
Стандарт
ANSI SQL, *60*
Стандартные параметры
программ
`isamchk` и `myisanchk`, *875*
`myisampack` и `pack_isam`, *880*
MySQL, *865*; *883*
`mysqladmin`, *891*
`mysqlbinlog`, *894*
`mysqld`, *900*
`mysqldump`, *917*
`mysqlhotcopy`, *921*
`mysqlimport`, *924*
`mysqlshow`, *928*
сценария `mysqlaccess`, *890*
Столбец таблицы, *44*
Стратегия планирования, *282*
Строка
use DBI, *383*
use strict, *383*
таблицы, *44*
Строковые
(символьные) данные, *115*
типы, *119*; *724*
функции, *754*
Структура MYSQL_FIELD, *356*
СУБД (Система управления базами данных), *35*
Суммирующие функции, *772*
Сценарий
mysql.server, *538*; *548*; *927*
mysql_install_db, *549*; *926*
`mysqlaccess`, *890*
`mysqlbug`, *895*
mysqld.multi, *548*
`mysqld_multi`, *914*
mysqld_safe, *548*; *915*
`mysqlhotcopy`, *921*
`safe_mysqld`, *928*
score_entry.php, *495*
языка Perl, *380*

T

Таблица, *44*
типа HEAP, *201*
Типы данных
BIGINT, *118*; *722*
BLOB, *119*; *130*; *135*; *725*
CHAR, *119*; *130*; *132*; *724*
CHAR(n), *172*
DATE, *60*; *119*; *143*; *144*; *168*; *728*
DATETIME, *119*; *143*; *144*; *168*; *728*

DECIMAL, *119*; *124*; *126*; *724*
DECIMAL(M,2), *168*
DOUBLE, *119*; *124*; *723*
DOUBLE PRECISION, *124*
ENUM, *119*; *130*; *727*
ENUM, *136*
FLOAT, *119*
FLOAT(4), *125*
FLOAT(8), *125*
FLOAT[(M,D)], *723*
FLOAT[(p)], *722*
FLOAT4, *125*
FLOAT8, *125*
INT, *118*; *722*
LONGBLOB, *119*; *130*; *135*; *726*
LONGTEXT, *119*; *130*; *135*; *727*
MEDIUMBLOB, *119*; *130*; *135*; *726*
MEDIUMINT, *118*; *722*
MEDIUMTEXT, *119*; *130*; *135*; *726*
MYSQL_RES, *345*; *935*
MYSQL_ROW, *348*; *935*
number, *113*
NUMERIC, *124*
REAL, *124*
SET, *119*; *130*; *136*; *137*; *727*
SMALLINT, *118*; *721*
string, *113*
TEXT, *119*; *130*; *135*; *726*
TIME, *119*; *143*; *144*; *728*
TIMESTAMP, *119*; *143*; *145*; *728*
TINYBLOB, *119*; *130*; *135*; *725*
TINYINT, *118*; *721*
TINYTEXT, *119*; *130*; *135*; *726*
VARCHAR, *60*; *119*; *130*; *132*; *725*
YEAR, *119*; *143*; *147*; *729*
YEAR(2), *147*
YEAR(4), *147*
СУБД MySQL, *115*
Типы даты и времени, *727*
Типы таблиц
BDB, *201*
HEAP, *201*
InnoDB, *201*
ISAM, *199*
MERGE, *200*
MyISAM, *199*
Транзакция, *245*
Три основных метода запуска сервера, *557*
Тривиальное объединение, *225*
Триггер, *262*
Тэг, *467*; *468*

У

Увеличение производительности сервера, *291*
Удаление
базы данных, *198*
индекса, *216*
каскадного типа, *249*
таблиц, *211*
Унарные операторы, *733*
Управляющая последовательность, *116*
Управляющие HTML-последовательности, *446*
Упрощение работы оптимизатора, *272*
Условное создание таблиц, *205*
Условный оператор IF EXISTS, *211*
Установка нескольких серверов, *520*
Устаревшие функции, *1021*
Утилита
configure, *702*
параметр `--localstatedir=`*path_name*, *708*
параметр `--prefix=`*path_name*, *708*
параметр `--with-berkeley-db`, *707*
параметр `--with-embedded-server`, *708*
параметр `--with-innodb`, *707*
параметр `--with-low-memory`, *708*
параметр `--without-innodb`, *707*
параметр `--without-server`, *707*
mysql_config, *318*

Ф

Файл
.frm, *202*
db.opt, *198*
заголовка
myclient.h, *318*
Флаг CLIENT_FOUND_ROWS, *346*
Формат
RTF (Rich Text Format), *42*
календарных типов, *148*
Функции
ABS, *750*
ACOS, *750*
ADDDATE, *764*
AES_DECRYPT, *774*
AES_ENCRYPT, *774*
ASCII, *754*
ASIN, *750*
ATAN, *750*

ATAN2, *750*
AVG, *96*; *773*
BENCHMARK, *777*
BIN, *754*
BIT_AND, *773*
BIT_COUNT, *777*
BIT_LENGTH, *778*
BIT_OR, *773*
CAST, *209*; *749*
CEILING, *750*
CHAR, *754*
CHAR_LENGTH, *132*; *755*
CHARACTER_LENGTH, *754*
CHARSET, *755*
check_pass, *511*
COALESCE, *755*
CONCAT, *734*; *755*
CONCAT_WS, *755*
CONNECTION_ID, *778*
CONV, *755*
CONVERT, *132*; *163*; *749*
CONVERT(expr USING charset), *749*
COS, *750*
COT, *750*
COUNT, *92*; *773*
CRC32, *751*
CURDATE, *764*
CURRENT_DATE, *764*
CURRENT_TIME, *764*
CURRENT_TIMESTAMP, *764*
CURTIME, *764*
DATABASE, *778*
DATE_ADD, *88*; *765*
DATE_FORMAT, *117*; *766*
DATE_SUB, *88*; *767*
DAYNAME, *768*
DAYOFMONTH, *86*; *768*
DAYOFWEEK, *768*
DAYOFYEAR, *768*
DECODE, *775*
DEGREES, *751*
DES_DECRYPT, *775*
DES_ENCRYPT, *775*
display_entry, *512*
display_form, *504*
ELT, *756*
ENCODE, *776*
ENCRYPT, *776*
EXP, *751*
EXPORT_SET, *756*
EXTRACT, *768*
FIELD, *756*
FIND_IN_SET, *757*
FLOOR, *87*; *751*
FORMAT, *757*
FOUND_ROWS, *778*
FROM_DAYS, *768*
FROM_UNIXTIME, *769*
GET_LOCK, *778*
getopt_long, *332*
GREATEST, *747*
handle_options, *332*
HEX, *757*
HOUR, *769*
html_begin, *477*
IF, *747*
IFNULL, *748*
INET_ATON, *780*
INET_NTOA, *780*
INSERT, *757*
INSTR, *758*
INTERVAL, *748*
IS_FREE_LOCK, *780*
ISNULL, *748*
LAST_INSERT_ID, *152*; *780*
LAST_INSERT_ID() с аргументом, *158*
LCASE, *758*
LEAST, *748*
LEFT, *758*
LENGTH, *132*; *758*
LN(x), *751*
LOAD_FILE, *780*
load_image, *374*
LOCATE, *758*
LOG, *751*
LOG10, *752*
LOG2, *752*
LOWER, *758*
LPAD, *758*
LTRIM, *759*
MAKE_SET, *759*
MASTER_POS_WAIT, *781*
MATCH, *759*
MAX, *96*; *774*
MD5, *776*
MID, *760*
MIN, *96*; *774*
MINUTE, *769*
MOD, *752*
MONTH, *86*
MONTHNAME, *769*
mysql_close, *323*
mysql_data_seek, *371*
mysql_error, *324*; *945*
mysql_escape_string, *372*; *373*
mysql_fetch_lengths, *356*
mysql_fetch_row, *347*; *348*
mysql_fetch_seek, *358*
mysql_field_count, *352*
mysql_init, *321*

mysql_num_fields, *352*
mysql_num_rows, *349*
mysql_real_connect, *321*
mysql_real_escape_string, *373*; *945*
mysql_rows_affected, *346*
mysql_store_result, *347*; *353*
mysql_use_result, *347*; *353*
NOW, *147*; *769*
OCT, *761*
OCTET_LENGTH, *761*
ORD, *761*
param, *447*
PASSWORD, *777*
password_field, *510*
PERIOD_ADD, *769*
PERIOD_DIFF, *770*
PHP
  addslashes, *488*
  count, *484*
  die, *478*
  display_cell, *496*
  display_events, *495*
  each, *501*
  empty, *486*
  htmlspecialchars, *490*
  is_int, *501*
  isset, *486*
  MD5, *508*
  mysql_affected_rows, *1006*
  mysql_change_user, *996*
  mysql_character_set_name, *1020*
  mysql_close, *996*
  mysql_connect, *997*
  mysql_create_db, *1021*
  mysql_data_seek, *1006*
  mysql_db_name, *1007*
  mysql_db_query, *1021*
  mysql_drop_db, *1021*
  mysql_errno, *480*; *487*; *1000*
  mysql_error, *480*; *487*; *1000*
  mysql_escape_string, *489*; *1001*
  mysql_fetch_array, *485*; *1007*
  mysql_fetch_assoc, *484*; *1009*
  mysql_fetch_field, *1009*
  mysql_fetch_lengths, *1011*
  mysql_fetch_object, *485*; *1012*
  mysql_fetch_row, *483*; *1012*
  mysql_field_flags, *1012*
  mysql_field_len, *1014*
  mysql_field_name, *1014*
  mysql_field_seek, *1014*
  mysql_field_table, *1015*
  mysql_field_type, *1016*
  mysql_free_result, *472*; *483*; *1016*
  mysql_get_client_info, *1020*
  mysql_get_host_info, *1020*
  mysql_get_proto_info, *1020*
  mysql_get_server_info, *1020*
  mysql_info, *1020*
  mysql_insert_id, *1017*
  mysql_list_dbs, *1001*
  mysql_list_fields, *1002*
  mysql_list_processes, *1002*
  mysql_list_tables, *1003*
  mysql_num_fields, *483*; *1018*
  mysql_num_rows, *483*; *1018*
  mysql_pconnect, *470*; *998*
  mysql_ping, *999*
  mysql_query, *472*; *480*; *1003*
  mysql_real_escape_string, *489*; *1004*
  mysql_result, *1018*
  mysql_select_db, *1005*
  mysql_stat, *1020*
  mysql_tablename, *1019*
  mysql_thread_id, *1020*
  mysql_unbuffered_query, *1005*
  ORDER BY RAND, *503*
  preg_match, *498*
  quote_value, *489*
  sampdb_connect, *475*
  urlencode, *490*
PI, *752*
POSITION, *761*
POW, *752*
POWER, *752*
present_question, *504*
PROCEDURE ANALYSE, *166*
process_result_set, *347*
QUARTER, *770*
QUOTE, *761*
RADIANS, *752*
RAND, *752*
RELEASE_LOCK, *781*
REPEAT, *761*
REPLACE, *762*
REVERSE, *762*
RIGHT, *762*
ROUND, *753*
RPAD, *762*
RTRIM, *762*
script_name, *496*
SEC_TO_TIME, *770*
SECOND, *770*
SESSION_USER, *781*
SHA, *777*
SHA1, *777*
SIGN, *753*
SIN, *753*
SOUNDEX, *762*
SPACE, *763*

SQL, *746*
SQRT, *753*
STD, *774*
STDDEV, *774*
STRCMP, *748*
SUBDATE, *770*
SUBSTRING, *763*
SUBSTRING_INDEX, *763*
SUM, *96*; *774*
SYSDATE, *770*
SYSTEM_USER, *781*
TAN, *753*
TIME_FORMAT, *770*
TIME_TO_SEC, *770*
TO_DAYS, *87*; *771*
TRIM, *763*
TRUNCATE, *754*
UCASE, *764*
UNIX_TIMESTAMP, *771*
UPPER, *764*
USER, *781*
VERSION, *781*
WEEK, *771*
YEAR, *86*; *772*
YEARWEEK, *772*
даты и времени, *764*
интерфейса DBI
looks_like_number, *982*
neat, *983*
neat_list, *983*
обмена данными со встроенным сервером, *959*
обработки результирующих наборов, *947*; *1006*
отладки, *960*
поддержки процессовых клиентов, *958*
построения и выполнения запросов, *1001*
программ интерфейса DBI, *982*
программного интерфейса приложений C, *937*
C-интерфейса
mysql_affected_rows, *948*
mysql_change_user, *937*
mysql_character_set_name, *956*
mysql_close, *938*
mysql_connect, *961*
mysql_create_db, *961*
mysql_data_seek, *949*
mysql_debug, *960*
mysql_drop_db, *961*
mysql_dump_debug_info, *960*
mysql_eof, *961*
mysql_errno, *944*
mysql_escape_string, *962*
mysql_fetch_field, *949*
mysql_fetch_field_direct, *950*
mysql_fetch_fields, *950*
mysql_fetch_lengths, *951*
mysql_fetch_row, *951*
mysql_field_count, *952*
mysql_field_seek, *953*
mysql_field_tell, *953*
mysql_free_result, *953*
mysql_get_client_info, *956*
mysql_get_host_info, *957*
mysql_get_proto_info, *957*
mysql_get_server_info, *957*
mysql_info, *957*
mysql_init, *938*
mysql_insert_id, *953*
mysql_kill, *962*
mysql_list_dbs, *962*
mysql_list_fields, *962*
mysql_list_processes, *963*
mysql_list_tables, *963*
mysql_num_fields, *954*
mysql_num_rows, *955*
mysql_options, *939*
mysql_ping, *941*
mysql_real_connect, *941*
mysql_real_query, *947*
mysql_refresh, *963*
mysql_reload, *964*
mysql_row_seek, *955*
mysql_row_tell, *955*
mysql_select_db, *947*
mysql_server_end, *959*
mysql_server_init, *959*
mysql_shutdown, *958*
mysql_stat, *957*
mysql_store_result, *955*
mysql_thread_end, *958*
mysql_thread_id, *958*
mysql_thread_init, *959*
mysql_thread_safe, *959*
mysql_use_result, *956*
создания и выполнения запросов, *945*
состояния и диагностики ошибок, *944*; *999*
сравнения, *747*
управления соединением, *937*; *996*

X

Хранимая процедура, *262*

Ц

Целое число, *115*
Целочисленные типы, *121*
Целый тип, *118*
  BIGINT, *122*
  INT, *122*
  MEDIUMINT, *122*
  SMALLINT, *122*
  TINYINT, *122*
Цифровые
  данные, *115*
  типы данных, *121*

Ч

Числовые
  типы, *720*
  функции, *750*
Чувствительность к регистру
  имен
    базы данных, *196*
    псевдонимов, *196*
    столбцов и индексов, *196*
    таблиц, *196*
    функций, *196*
  ключевых слов, *196*

Ш

Шаблон, *89*
  '%', *179*
  '_', *179*, *180*
Шестнадцатеричный формат
  представления, *116*

Э

Эффективность запросов, *276*

Я

Языки
  SQL, *35*; *46*
  написания сценариев
    Perl, *42*; *301*
    PHP, *42*; *301*

Научно-популярное издание

Поль Дюбуа

MySQL, 2-е издание

Литературный редактор	*И.А. Попова*
Верстка	*О.В. Линник*
Художественный редактор	*С.А. Чернокозинский*
Корректоры	*З.В. Александрова, Л.А. Гордиенко, Т.А. Корзун, О.В. Мишутина, Л.В. Чернокозинская*

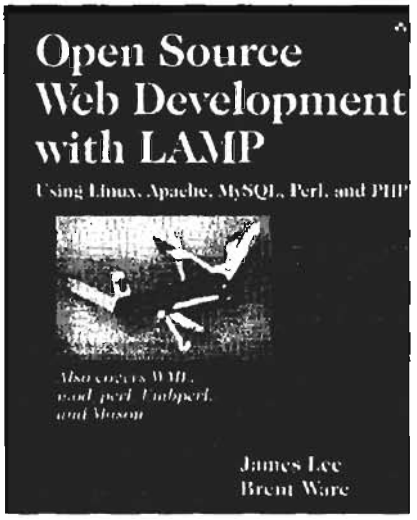
Издательский дом "Вильямс".
101509, Москва, ул. Лесная, д. 43, стр. 1.
Изд. лиц. ЛР № 090230 от 23.06.99
Госкомитета РФ по печати.

Подписано в печать 16.02.2004. Формат 70×100/16.
Гарнитура Times. Печать офсетная.
Усл. печ. л. 85,14. Уч.-изд. л. 61,5.
Тираж 5000 экз. Заказ № 2049.

Отпечатано с диапозитивов в ФГУП "Печатный двор"
Министерства РФ по делам печати,
телерадиовещания и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.

ИСПОЛЬЗОВАНИЕ LINUX, APACHE, MYSQL И PHP ДЛЯ РАЗРАБОТКИ WEB-ПРИЛОЖЕНИЙ

Джеймс Ли,
Брент Уар



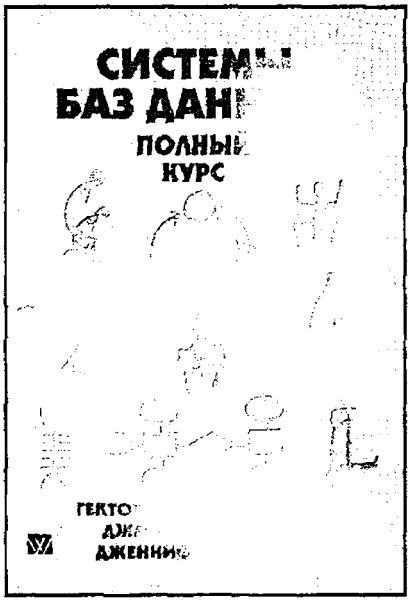
www.williamspublishing.com

Эта книга знакомит вас с основами технологий разработки приложений с открытым исходным кодом.
Такие технологии позволяют создавать интерактивные, основанные на базах данных, Web-узлы при относительно небольших затратах.
Книга предназначена для программистов знакомых с основами Unix и HTML.
В книге даны руководства по установке Linux, настройке Web-сервера, созданию базы данных MySQL. Рассказывается о том, как создавать статические Web-страницы при помощи языка Website META Language (WML) и динамические при помощи интерфейса CGI и mod-perl.

Плановая дата выхода
2 кв. 2004 г.

СИСТЕМЫ БАЗ ДАННЫХ. ПОЛНЫЙ КУРС

Гектор Гарсиа-Молина,
Джеффри Д. Ульман,
Дженнифер Уидом



Книга известного специалиста в области компьютерных наук Дж. Ульмана и его именитых коллег по Станфордскому университету является уникальным учебным и справочным пособием, которое отличается беспрецедентными широтой и глубиной охвата предмета и представляет несомненный интерес для всех, кто по роду своей профессиональной деятельности сталкивается с проблемами проектирования и использования современных систем баз данных.

www.williamspublishing.com

в продаже

ПРАКТИЧЕСКАЯ КРИПТОГРАФИЯ

Нильс Фергусон,
Брюс Шнайер



В современном деловом мире вопрос безопасности компьютерных систем приобретает все более решающее значение. Проигнорировав его, вы лишаете себя возможности заработать деньги, расширить свой бизнес и, как следствие, ставите под угрозу само существование вашей компании. Одной из наиболее многообещающих технологий, позволяющих обеспечить безопасность в киберпространстве, является криптография. Данная книга, написанная всемирно известными специалистами в области криптографии, представляет собой уникальное в своем роде руководство по практической разработке криптографической системы, устраняя тем самым досадный пробел, существовавший ранее между теоретическими основами криптографии и реальными криптографическими приложениями.

www.williamspublishing.com

Плановая дата выхода
II кв. 2004 г.

Мир книг по Linux от издательской группы *"ДИАЛЕКТИКА - ВИЛЬЯМС"*



ISBN 5-8459-0234-7



ISBN 5-8459-0491-



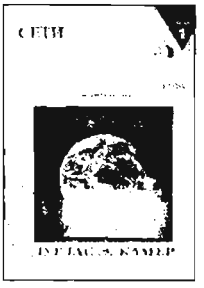
ISBN 5-8459-0405-6



ISBN 5-8459-0426-9



ISBN 5-8459-0243-6



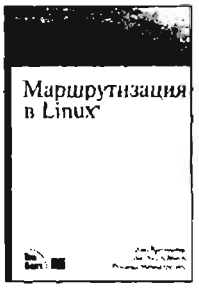
ISBN 5-8459-0419-6



ISBN 5-8459-0485-4



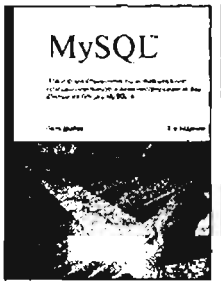
ISBN 5-8459-0244-4



ISBN 5-8459-0271-1



ISBN 5-8459-0348-3



ISBN 5-8459-0515-X



ISBN 5-8459-0516-8

... и много других книг Вы найдете на наших сайтах

www.dialektika.com

www.williamspublishing.com

www.ciscopress.ru