МАКСИМ КУЗНЕЦОВ, ИГОРЬ СИМДЯНОВ

MySQL

НА ПРИМЕРАХ

НОВЫЕ ВОЗМОЖНОСТИ MySQL 5.1

ХРАНИМЫЕ ПРОЦЕДУРЫ

ПРЕДСТАВЛЕНИЯ

ТРИГГЕРЫ

ПЛАНИРОВЩИК ЗАДАНИЙ

СЕГМЕНТИРОВАНИЕ

ООП-БИБЛИОТЕКА php_mysqli

+CD

bhv®

Максим Кузнецов
Игорь Симдянов

MySQL
НА ПРИМЕРАХ

Санкт-Петербург
«БХВ-Петербург»
2007

УДК 681.3.068
ББК 32.973.26-018.1
К89

Кузнецов, М. В.

К89 MySQL на примерах / М. В. Кузнецов, И. В. Симдянов. — СПб.: БХВ-Петербург, 2007. — 592 с.: ил. + CD-ROM

ISBN 978-5-9775-0066-1

Книга посвящена популярной СУБД MySQL 5.1. На практических примерах рассматриваются вложенные запросы, хранимые процедуры, представления, триггеры, курсоры, информационная схема, планировщик заданий, сегментирование. Большое внимание уделяется взаимодействию клиентского кода на PHP с СУБД MySQL. Подробно описаны две наиболее популярные библиотеки: классическая библиотека php_mysql и новая объектно-ориентированная библиотека php_mysqli. На компакт-диске содержатся дистрибутивы и исходные коды MySQL 5.0/5.1/5.2 для Windows и Linux и графические клиенты для MySQL.

Для программистов и разработчиков баз данных

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Наталья Таркова*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 24.04.07.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 47,73.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0066-1

© Кузнецов М. В., Симдянов И. В., 2007
© Оформление, издательство "БХВ-Петербург", 2007

Оглавление

Введение **1**

Благодарности 2

Глава 1. Установка MySQL **3**

1.1. Структура MySQL 3
1.1.1. Клиент-серверная архитектура 3
1.1.2. Загрузка дистрибутивов 5
1.1.3. Структура дистрибутива 7
1.1.4. Официальная документация 8
1.2. Установка MySQL под Windows 9
1.2.1. Процесс установки 9
1.2.2. Постинсталляционная настройка 15
1.2.3. Проверка работоспособности MySQL 23
1.3. Установка MySQL под Linux 25
1.4. Конфигурационный файл 29
1.5. Перемещение каталога данных 32
1.6. Обновление текущей версии 32
1.7. Обзор утилит MySQL 34
1.8. Утилита *mysql* 35
1.8.1. Командная строка 35
1.8.2. Установка соединения с локальным сервером 40
1.8.3. Установка соединения с удаленным сервером 45
1.8.4. Вертикальный режим вывода 46
1.9. Настройка кодировок 48
1.10. Графические клиенты MySQL 53
1.11. Перенос баз данных с одного сервера на другой 54
1.11.1. Копирование бинарных файлов 54
1.11.2. Создание SQL-дампа 56
1.11.3. Использование операторов *SELECT* и *LOAD DATA* 58
1.11.4. Операторы *BACKUP TABLE* и *RESTORE TABLE* 64

1.12. Журнальные файлы MySQL ... 67
1.12.1. Журнальные таблицы ... 68
1.12.2. Журнал ошибок ... 68
1.12.3. Общий журнал запросов ... 69
1.12.4. Бинарный журнал регистраций ... 69
1.12.5. Журнал медленных запросов ... 72
1.13. Лицензирование MySQL ... 73

Глава 2. Базы данных и таблицы ... 75

2.1. Создание базы данных ... 75
2.2. Выбор базы данных ... 79
2.3. Создание таблиц ... 81
2.3.1. Структура таблицы ... 82
Числовой тип данных ... 82
Строковый тип данных ... 85
Календарный тип данных ... 87
Тип данных *NULL* ... 89
2.3.2. Просмотр структуры таблицы ... 91
2.3.3. Параметры столбцов ... 94
2.3.4. Допустимые имена баз данных, таблиц и столбцов ... 98
2.3.5. Параметры таблицы ... 101
ENGINE (TYPE) ... 101
AUTO_INCREMENT ... 102
[DEFAULT] CHARACTER SET ... 103
CHECKSUM ... 103
COMMENT ... 104
DATA DIRECTORY ... 104
INDEX DIRECTORY ... 104
PACK_KEYS ... 105
ROW_FORMAT ... 105
2.3.6. Временные таблицы ... 105
2.3.7. Создание копии таблицы ... 106
2.3.8. Размещение таблиц в оперативной памяти ... 108
2.3.9. Объединение нескольких таблиц в одну ... 109
2.3.10. Размещение таблиц на удаленной машине ... 111
2.4. Удаление таблиц ... 113
2.5. Редактирование таблицы ... 114
2.5.1. Добавление столбца ... 115
2.5.2. Удаление столбца ... 116
2.5.3. Изменение уже существующих столбцов ... 116
2.5.4. Переименование таблицы ... 118
2.5.5. Преобразование параметров таблицы ... 119
2.6. Восстановление таблиц ... 121
2.6.1. Оператор *REPAIR TABLE* ... 121
2.6.2. Контрольная сумма таблицы ... 123

Глава 3. Индексы ... 127

3.1. Индексация столбцов при создании таблицы ... 127
3.1.1. Первичный ключ ... 128
3.1.2. Создание уникального индекса при помощи атрибута *AUTO_INCREMENT* ... 131
3.1.3. Обычный и уникальный индексы ... 132
3.2. Добавление и удаление индексов в существующую таблицу ... 134
3.2.1. Добавление индекса ... 134
3.2.2. Удаление индекса ... 136
3.3. Восстановление индексов ... 137

Глава 4. Заполнение, обновление и очистка таблиц ... 141

4.1. Вставка записи в таблицу. Оператор *INSERT* ... 141
4.1.1. Вставка числовых значений ... 141
4.1.2. Вставка строковых значений ... 143
4.1.3. Вставка календарных значений ... 145
4.1.4. Тип данных *TIMESTAMP* ... 148
4.1.5. Вставка данных в формате *UNIXSTAMP* ... 149
4.1.6. Вставка уникальных значений ... 150
4.1.7. Механизм *AUTO_INCREMENT* ... 151
4.1.8. Вставка вычисляемых значений ... 153
4.2. Многострочный оператор *INSERT* ... 154
4.3. Отложенная вставка записей ... 154
4.4. Пакетная загрузка данных. Оператор *INSERT ... SELECT* ... 155
4.5. Удаление данных ... 157
4.5.1. Оператор *DELETE* ... 157
4.5.2. Оператор *TRUNCATE* ... 158
4.5.3. Удаление из нескольких таблиц ... 158
4.5.4. Каскадное удаление из нескольких таблиц ... 162
4.6. Обновление записей ... 165
4.6.1. Оператор *UPDATE* ... 165
4.6.2. Многотабличный оператор *UPDATE* ... 166
4.6.3. Оператор *REPLACE* ... 168

Глава 5. Извлечение данных ... 171

5.1. Простейшая выборка из таблицы ... 171
5.2. Условная выборка ... 173
5.3. Комбинированные условия ... 175
5.4. *NULL* и логические операторы ... 177
5.5. Выборка из диапазона ... 179
5.6. Выборка множества ... 180
5.7. Использование строк в условии ... 181
5.8. Использование календарных значений в условии ... 186

5.9. Форматирование календарных значений ... 188
5.10. Модификация календарных значений ... 190
5.11. Псевдонимы столбцов ... 191
5.12. Сортировка записей ... 192
5.13. Вывод записей в случайном порядке ... 195
5.14. Ограничение выборки ... 195
5.15. Текущая версия MySQL ... 196
5.16. Количество записей в таблице ... 198
5.17. Поиск минимального и максимального значений ... 199
5.18. Вывод уникальных значений ... 200
5.19. Групповые условия. Условие *HAVING* ... 203
5.20. Объединение таблиц ... 205
5.21. Перекрестное объединение таблиц ... 208
5.22. Самообъединение таблиц ... 214
5.23. Объединение таблиц при помощи *JOIN* ... 215
5.24. Вложенный запрос, возвращающий одно значение ... 218
5.25. Вложенные запросы, возвращающие несколько строк ... 223
5.25.1. Ключевое слово *IN* ... 224
5.25.2. Ключевое слово *ANY (SOME)* ... 225
5.25.3. Ключевое слово *ALL* ... 227
5.26. Проверка результирующей таблицы на существование ... 228
5.27. Вложенные запросы, возвращающие несколько столбцов ... 230
5.28. Подзапросы в конструкции *FROM* ... 232

Глава 6. Встроенные функции ... 233

6.1. Математические функции ... 233
6.1.1. Вычисление площади треугольников ... 235
6.1.2. Округление результатов вычисления ... 237
6.1.3. Вывод случайной записи ... 239
6.2. Функции даты и времени ... 240
6.2.1. Сложение и вычитание интервалов ... 245
6.2.2. Заполнение календарных значений ... 249
6.2.3. Форматирование календарных значений ... 250
6.2.4. Вычисление возраста человека ... 250
6.2.5. Преобразование даты в UNIXSTAMP-формат ... 252
6.3. Строковые функции ... 253
6.3.1. Изменение кодировки строки ... 258
6.3.2. Первые несколько символов строки ... 259
6.3.3. Извлечение инициалов ... 259
6.3.4. Изменение регистра строки ... 260
6.3.5. Сжатие строки ... 262
6.3.6. Загрузка строки из файла ... 263
6.3.7. Извлечение строк, содержащих искомую подстроку в поле типа *SET* ... 265

6.4. Функции шифрования ... 266
6.4.1. Обратимое шифрование ... 268
6.4.2. Необратимое шифрование ... 269
6.5. Функции управления потоком ... 271
6.5.1. Статус записи ... 271
6.5.2. Замена *NULL* на эквиваленты ... 273
6.6. Информационные функции ... 274
6.6.1. Статусная информация о текущем сеансе ... 275
6.6.2. Информация о сервере ... 276
6.6.3. Фактическое количество строк в таблице ... 277
6.7. Разное ... 278
6.7.1. Преобразование IP-адреса ... 279
6.7.2. Блокировка ... 280
6.7.3. Универсальный уникальный идентификатор компьютера ... 282

Глава 7. Поиск ... 285

7.1. Регулярные выражения ... 285
7.2. Полнотекстовый поиск ... 295
7.2.1. Индекс *FULLTEXT* ... 296
7.2.2. Конструкция *MATCH (...) AGAINST (...)* ... 298
7.2.3. Модификаторы полнотекстового поиска ... 302
7.2.4. Логический режим ... 303
7.2.5. Режим расширения запроса ... 305

Глава 8. Функции, применяемые вместе с конструкцией *GROUP BY* ... 307

8.1. Среднее значение ... 308
8.2. Сортировка агрегатных значений ... 311
8.3. Подсчет количества записей в таблице ... 312
8.4. Объединение значений группы ... 315
8.5. Поиск минимального и максимального значений ... 316
8.6. Сумма столбца ... 318
8.7. Конструкция *WITH ROLLUP* ... 319

Глава 9. Переменные ... 321

9.1. Переменные SQL ... 321
9.2. Нумерация записей ... 325
9.3. Системные переменные сервера ... 325
9.4. Динамические запросы ... 330

Глава 10. Транзакции и блокировки ... 333

10.1. Транзакции ... 333
10.2. Когда можно обойтись без транзакций? ... 338
10.3. Блокировка таблиц ... 339

Глава 11. Администрирование MySQL 343

11.1. Создание и удаление пользователей 343
11.2. Назначение пароля 346
11.3. Восстановление утерянного пароля 346
11.4. Удаленный доступ к MySQL 346
11.5. Управление привилегиями пользователей 348
11.6. Просмотр существующих привилегий 356
11.7. Ограничение на число соединений с сервером и число запросов 356
11.8. Репликация 359

Глава 12. Хранимые процедуры и функции 365

12.1. Создание хранимой процедуры (функции) 365
12.2. Использование параметров 369
12.3. Локальные переменные 372
12.4. Присвоение значения локальной переменной 375
12.5. Форматирование временного интервала 376
12.6. Группа характеристик хранимых процедур 379
12.7. Операторы ветвления 382
12.7.1. Оператор *IF...THEN...ELSE* 382
12.7.2. Оператор *CASE* 385
12.7.3. Проверка правильности ввода 387
12.7.4. Вывод приветствия в зависимости от времени суток 388
12.8. Циклы 389
12.8.1. Оператор *WHILE* 389
12.8.2. Досрочный выход из цикла 391
12.8.3. Оператор *REPEAT* 394
12.8.4. Оператор *LOOP* 395
12.8.5. Отсутствующие идентификаторы 396
12.9. Получение списка созданных хранимых процедур и функций 397
12.10. Содержимое хранимой процедуры 402
12.11. Удаление хранимых процедур и функций 402
12.12. Редактирование хранимых процедур 404
12.13. Обработчики ошибок 405
12.14. Курсоры 410
12.15. Рекурсивные хранимые процедуры 413

Глава 13. Триггеры 419

13.1. Создание триггера 419
13.2. Удаление триггера 423
13.3. Список созданных триггеров 424

Глава 14. Представления 427

14.1. Создание представлений 427
14.2. Сокрытие столбцов 432

14.3. Сокрытие строк ... 434
14.4. Обновление и удаление записей при помощи представлений ... 435
14.5. Редактирование представления ... 436
14.6. Удаление представлений ... 437
14.7. Просмотр структуры представления ... 438

Глава 15. Информационная схема ... 439

15.1. Список баз данных ... 442
15.2. Список таблиц ... 444
15.3. Список столбцов ... 447

Глава 16. Планировщик заданий ... 453

16.1. Управление планировщиком задач ... 453
16.2. Размещение задания ... 456
16.3. Удаление задания ... 460
16.4. Редактирование задания ... 461
16.5. Список заданий ... 463

Глава 17. Сегментирование ... 465

17.1. Включено ли сегментирование? ... 465
17.2. Создание сегментированной таблицы ... 466
17.3. Типы сегментирования таблиц ... 467
17.3.1. Линейная сегментация *RANGE* ... 467
17.3.2. Списочная сегментация *LIST* ... 471
17.3.3. Хэш-сегментация *HASH* ... 474
17.3.4. Индексная сегментация *KEY* ... 475
17.4. Подразделы ... 476
17.5. Распределение подразделов по разным папкам и дискам ... 479
17.6. Удаление разделов ... 481
17.7. Добавление разделов ... 483
17.8. Перепланировка разделов ... 485

Глава 18. Взаимодействие с PHP ... 489

18.1. Расширение php_mysql ... 489
18.2. Расширение php_mysqli ... 497
18.3. Создание базы данных ... 505
18.4. Создание и заполнение таблицы ... 506
18.5. Заполнение связанных таблиц ... 508
18.6. Вывод данных ... 515
18.7. Повторное чтение результирующей таблицы ... 517
18.8. Количество строк в таблице ... 520
18.9. Вывод результатов в несколько столбцов ... 521
18.10. Постраничная навигация ... 524
18.11. Удаление данных ... 528

18.12. Сортировка 532
18.13. Размер базы данных 535

Заключение 537

Приложение 1. Системные переменные MySQL 539

Приложение 2. Описание компакт-диска 565

Предметный указатель 567

Введение

Книги по компьютерной тематике традиционно делятся на два типа: последовательно излагающие материал и рецептурно. Проводя аналогию, можно сказать, что первый тип книг относится к своду законов, а второй — к их комментариям. При последовательном изложении материала дается наиболее полный список функций и конструкций и краткие примеры их использования, при рецептурном изложении приводятся наиболее эффективные методики, удачные решения, традиционные методы, наиболее частые ошибки, даже фольклор, т. е. передается "дух" предметной области. Книга, которую вы держите в руках, относится ко второму типу.

Некоторое время назад нами были написаны две книги, посвященные MySQL и последовательно излагающие материал: "Самоучитель MySQL 5"[1] и "MySQL 5"[2], позволяющие читателю повысить свой уровень до разработчика и разработчика-администратора соответственно. Очень скоро опытные разработчики на нашем форуме **http://www.softtime.ru/forum/index.php?id_forum=3**, высказали мнение о необходимости создания книги, в которой бы обсуждались сложные приемы работы, вопросы, которые не рассматриваются в традиционных изданиях, но обсуждаются на форумах. Мы решили объединить наиболее интересные приемы, а также часто задаваемые вопросы в данной книге. Формально данная книга является продолжением книг "Самоучитель MySQL 5" и "MySQL 5", де-факто книга является независимой и ориентирована на читателей, которые знакомы с языком запросов SQL, однако желающие освоить диалект MySQL или повысить свой уровень, разобраться в вопросах, "туманно" освещаемых документацией и большинством книг. Мы делимся с читателями, в первую очередь, опытом и приемами работы с СУБД

---

[1] Кузнецов М., Симдянов И. Самоучитель MySQL 5. — СПб.: БХВ-Петербург, 2006.

[2] Кузнецов М., Симдянов И. MySQL 5. — СПб.: БХВ-Петербург, 2006.

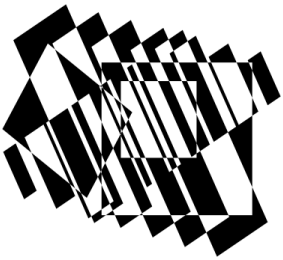
MySQL, которые приобрели при разработке приложений и сайтов в IT-студии SoftTime (**http://www.softtime.ru**) и при поддержке собственной хост-площадки (**http://www.st-host.ru**).

Кроме этого, в книге впервые рассматриваются нововведения версии 5.1, такие как планировщик заданий, динамические запросы, сегментирование. Подробно обсуждается взаимодействие с MySQL из PHP-программ с использованием как классической библиотеки php_mysql, так и нового улучшенного варианта php_mysqli, использующего объектно-ориентированный интерфейс.

Благодарности

Авторы выражают признательность сотрудникам издательства "БХВ-Петербург", благодаря которым наша рукопись увидела свет, а также компании MySQL AB и лично директору по связям с общественностью Стиву Кари (Steve Curry) за любезное разрешение поместить дистрибутивы MySQL на компакт-диске, прилагаемом к книге.

ГЛАВА 1



Установка MySQL

Как и любая другая СУБД (система управления базами данных), MySQL является сложным программным комплексом, от установки и настройки которого зависит производительность, устойчивость и безопасность. Данная глава посвящена установке и первичной настройки СУБД MySQL. Здесь рассматриваются вопросы, которые возникают практически сразу при работе с любой СУБД: как установить и наиболее эффективно настроить СУБД, как переносить базы данных с одного сервера на другой.

1.1. Структура MySQL

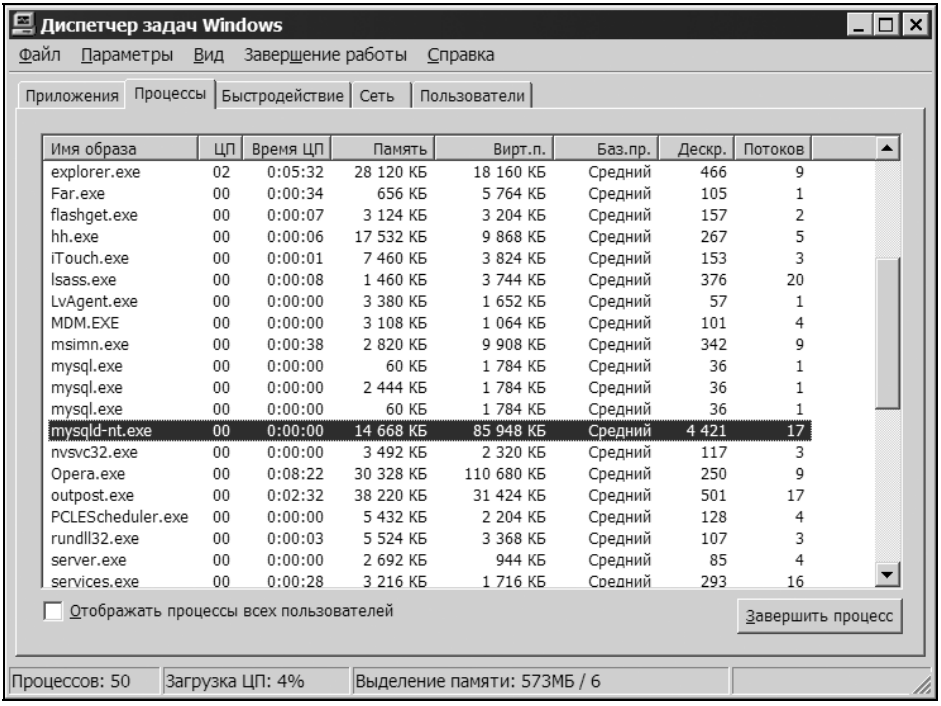
1.1.1. Клиент-серверная архитектура

СУБД MySQL построена по клиент-серверной технологии. Базы данных хранятся на сервере, который ожидает обращений клиентов. Клиенты посылают запросы, на которые после обработки сервером отсылается ответ. Примером клиентов сервера MySQL могут служить консольная утилита mysql, скрипт на языке программирования PHP или Perl, другой сервер, например, почтовый сервер. Для того чтобы программа имела возможность обращаться к серверу MySQL, она должна быть скомпилирована с клиентской библиотекой libmysql.

Замечание

Следует понимать, что клиенты и серверы СУБД — это программы, а не машины, на которых они выполняются. Помимо сервера базы данных, на машине может выполняться множество других программ, как серверов, так и клиентов. Сервер представляет собой сложную ресурсоемкую программу, поэтому для него часто выделяют отдельную машину, которую тоже называют сервером. Здесь и далее под сервером мы будем понимать программную часть, а не аппаратную.

Клиенты — это программы, которые обращаются к серверу (посылают запросы), а сервер — это программа, которая отвечает на эти запросы. Сервер постоянно прослушивает, не поступили ли запросы от клиентов, и если поступили — отвечает на эти запросы. Сервер и клиенты могут быть расположены на разных машинах или на одной. Например, при установке полной версии дистрибутива MySQL устанавливается как сервер MySQL, так и клиент. При помощи клиента можно обращаться как к локальному серверу, так и к удаленному по локальной сети или Интернету. Если MySQL-сервер соответствующим образом настроен, то клиент с другого узла сети может обратиться к MySQL-серверу на вашей машине. Клиент-серверная архитектура удобна, когда десяткам и сотням клиентов необходимо работать над единой копией базы данных.



Имя образа	ЦП	Время ЦП	Память	Вирт.п.	Баз.пр.	Дескр.	Потоков
explorer.exe	02	0:05:32	28 120 КБ	18 160 КБ	Средний	466	9
Far.exe	00	0:00:34	656 КБ	5 764 КБ	Средний	105	1
flashget.exe	00	0:00:07	3 124 КБ	3 204 КБ	Средний	157	2
hh.exe	00	0:00:06	17 532 КБ	9 868 КБ	Средний	267	5
iTouch.exe	00	0:00:01	7 460 КБ	3 824 КБ	Средний	153	3
lsass.exe	00	0:00:08	1 460 КБ	3 744 КБ	Средний	376	20
LvAgent.exe	00	0:00:00	3 380 КБ	1 652 КБ	Средний	57	1
MDM.EXE	00	0:00:00	3 108 КБ	1 064 КБ	Средний	101	4
msimn.exe	00	0:00:38	2 820 КБ	9 908 КБ	Средний	342	9
mysql.exe	00	0:00:00	60 КБ	1 784 КБ	Средний	36	1
mysql.exe	00	0:00:00	2 444 КБ	1 784 КБ	Средний	36	1
mysql.exe	00	0:00:00	60 КБ	1 784 КБ	Средний	36	1
mysqld-nt.exe	00	0:00:00	14 668 КБ	85 948 КБ	Средний	4 421	17
nvsvc32.exe	00	0:00:00	3 492 КБ	2 320 КБ	Средний	117	3
Opera.exe	00	0:08:22	30 328 КБ	110 680 КБ	Средний	250	9
outpost.exe	00	0:02:32	38 220 КБ	31 424 КБ	Средний	501	17
PCLEScheduler.exe	00	0:00:00	5 432 КБ	2 204 КБ	Средний	128	4
rundll32.exe	00	0:00:03	5 524 КБ	3 368 КБ	Средний	107	3
server.exe	00	0:00:00	2 692 КБ	944 КБ	Средний	85	4
services.exe	00	0:00:28	3 216 КБ	1 716 КБ	Средний	293	16

Рис. 1.1. Клиенты (mysql.exe) и сервер (mysqld-nt.exe) СУБД MySQL

На рис. 1.1 представлен список процесса Windows-утилиты Диспетчер задач. Как правило, сервер всегда один (на рис. 1.1 это процесс mysqld-nt.exe), а клиентов может быть несколько (так на рис. 1.1 видно три запущенных консольных клиента mysql.exe). Процесс сервера MySQL всегда имеет в своем составе слово "mysqld", это может быть mysqld-nt.exe, mysqld.exe или mysqld-

max.exe. Символ "d" обозначает принадлежность к серверам в операционной системе UNIX, где резидентные программы, выполняющиеся в фоновом режиме и ожидающие обращения клиента, традиционно называют *демонами* (daemon, слово изначально образовано от греческого "дух").

Для того чтобы проверить, запущен MySQL-сервер в UNIX-среде или нет, используют команду

```
ps -Af | grep mysql
```

Утилита `ps` выводит список всех (`-A`) процессов в полной форме (`-f`), при этом результат передается утилите `grep`, которая ищет в нем строки, в состав которых входит последовательность `mysql`. В UNIX-подобных операционных системах MySQL-сервер запускается в виде процесса `mysqld` (причем запускается сразу несколько процессов одновременно).

При создании независимых приложений, которые не должны зависеть от внешнего MySQL-сервера, зачастую прибегают к встроенному (embedded) серверу MySQL, который входит в состав основной программы, а не запускается в виде независимого процесса (процессов).

1.1.2. Загрузка дистрибутивов

На официальном сайте MySQL (**http://www.mysql.com**) дистрибутивы для большинства операционных систем доступны в предкомпилированном виде (свыше 20 вариантов). Разумеется, как для всякого продукта, который распространяется по свободной GPL-лицензии, для MySQL возможна загрузка исходных кодов. Однако самостоятельная компиляция исходных кодов не рекомендуется. Предкомпилированные версии оптимизированы для каждой из операционных систем для достижения наибольшей производительности, поэтому самостоятельная компиляция в лучшем случае приведет к сравнимым с бинарными дистрибутивами результатам.

Дистрибутив MySQL для разных операционных систем различается по своей структуре. Windows-машины традиционно используются в качестве клиентов, поэтому для операционной системы Windows дистрибутив представлен в виде единого файла (либо zip-архива, либо exe или msi установочных файлов). Linux, FreeBSD и вообще UNIX-подобные операционные системы традиционно используются в качестве серверной платформы, поэтому серверная, клиентская части, а также библиотеки совместного доступа распространяются для них в виде отдельных файлов: администратор имеет возможность установить только серверную часть и не устанавливать клиентскую, которой на сервере никто пользоваться не будет, и наоборот, клиентскую машину можно снабдить только клиентскими утилитами, не устанавливая полноценный сервер.

На официальном сайте дистрибутив MySQL на момент написания книги доступен в трех вариантах:

- устаревшая версия (на момент написания книги MySQL 4.1);
- рекомендуемая версия (на момент написания книги MySQL 5.0);
- разрабатываемая версия (на момент написания книги MySQL 5.1).

Принцип обновления версий следующий: в разрабатываемую версию добавляются новые функциональные возможности, которые разработчики отлаживают, регулярно выкладывая релизы дистрибутива. После того, как принято решение прекратить добавление нововведений в разрабатываемую версию, она получает статус "бета-версии". С этого момента разработчики лишь исправляют ошибки, после того, как достигается стабильность работы СУБД MySQL на всех тестах, она получает статус рекомендуемой версии, которая работает стабильно и официально рекомендуется пользователям для загрузки. При этом текущая рекомендуемая версия получает статус устаревшей. Лишь для этих трех версий компания MySQL AB выпускает обновления. Поэтому нужно следить, чтобы используемая версия не была ниже, чем версия "устаревшего" дистрибутива.

Замечание

Загрузить последние версии дистрибутива MySQL для операционной системы Windows можно также со страницы **http://www.softtime.ru/public/**.

Номера версий доступных в настоящий момент дистрибутивов можно найти на странице **http://dev.mysql.com/downloads/mysql/**, на момент написания книги URL для каждой из версий выглядели следующим образом:

- MySQL 4.1 — **http://dev.mysql.com/downloads/mysql/4.1.html**;
- MySQL 5.0 — **http://dev.mysql.com/downloads/mysql/5.0.html**;
- MySQL 5.1 — **http://dev.mysql.com/downloads/mysql/5.1.html**.

На каждой странице доступны ссылки для загрузки дистрибутива. В разделе **Windows downloads** доступны три версии Windows-дистрибутива:

- Windows Essentials (x86) — урезанная версия дистрибутива, из которой удалены все вспомогательные утилиты, по сути, это "голый" сервер MySQL;
- Windows (x86) — полная версия дистрибутива MySQL, включающая автоматический установщик;
- Without installer (unzip in C:\) — полная версия MySQL без автоматического установщика.

Рекомендуется выбрать вариант Windows (x86), который позволяет установить и настроить СУБД MySQL в диалоговом режиме, а также автоматически

устанавливает сервис для автоматического запуска и остановки MySQL при старте и завершении работы операционной системы. В случае версии без автоматического инсталлятора, придется либо каждый раз запускать систему автоматически, либо установить сервис самостоятельно при помощи команды `mysqld-nt --install` *Имя_сервиса*.

Замечание

Каждый из файлов снабжается хэш-кодом алгоритма MD5, который является уникальным для любой последовательности, даже если она отличается лишь одним байтом. MD5-хэш позволяет проконтролировать корректность загрузки дистрибутива.

UNIX-дистрибутивы разбиты на несколько частей, каждую из которых можно загружать отдельно:

- Server — сервер MySQL;
- Client — клиентские утилиты MySQL;
- Shared libraries — библиотеки совместного доступа, необходимые, главным образом, для работы разнообразных клиентов MySQL;
- Shared compatibility libraries — библиотеки совместного доступа старых версий. Установка данного пакета необходима в том случае, если текущие программы были скомпилированы со старыми библиотеками MySQL и их пакеты требуют удовлетворения зависимостей;
- Headers and libraries — заголовочные файлы и библиотеки, необходимые для компиляции программ, обращающихся к СУБД MySQL;
- Test suite — тесты производительности на языке Perl;
- Debug information — отладочная информация.

Замечание

Дистрибутивы достаточно объемны, поэтому они размещены на компакт-диске, поставляемом вместе с книгой. Загрузить последние версии СУБД MySQL, помимо официального сайта, можно также по адресу **http://www.softtime.ru/public/**.

1.1.3. Структура дистрибутива

Структура дистрибутивов под Windows и UNIX отличаются, т. к. в данных операционных системах существует значительное различие в организации файловой структуры. В Windows, где традиционно каждое приложение занимает свой каталог, расположение каталогов СУБД может выглядеть следующим образом:

- bin — каталог бинарных файлов, здесь хранятся файлы сервера и вспомогательных утилит MySQL;

- data — каталог данных, здесь хранятся базы, разнообразные журнальные файлы и прочая информация, которая подвергается изменениям;
- Docs — каталог содержит документацию по MySQL в текстовом формате и формате HTML;
- Embedded — библиотечные файлы для построения приложений, содержащих встроенный сервер MySQL;
- examples — примеры на языках C и Perl обращения к MySQL-серверу;
- include — заголовочные файлы, необходимые для разработки C-программ, обращающихся к MySQL-серверу;
- lib — библиотечные файлы, необходимые для создания клиентских программ, обращающихся к MySQL-серверу;
- scripts — вспомогательные скрипты для обслуживания MySQL-сервера (применяются в основном в UNIX-среде);
- share — общие файлы, необходимые как для клиентских программ, так и для MySQL-сервера, каталог содержит в основном файлы локализации для разных национальных языков;
- sql-bench — Perl-скрипты для измерения производительности MySQL-сервера.

В UNIX-подобных операционных системах разные части программ располагаются в разных каталогах, а зачастую и в разных разделах диска. Так, например, в RedHat-производных дистрибутивах изменяющиеся данные (журнальные файлы, каталог данных) хранятся в каталоге /var, утилиты — в каталоге /usr/sbin, а библиотеки — в каталоге /urs/lib, общие файлы — в каталоге /usr/share/mysql, а конфигурационные файлы — в каталоге /etc.

Замечание

Конечно, можно собрать все подкаталоги в один каталог, но, как правило, так не поступают.

1.1.4. Официальная документация

Программное обеспечение с открытым кодом характеризуется высоким уровнем динамизма. Дистрибутивы постоянно снабжаются дополнительной функциональностью, неудачные решения и утилиты объявляются устаревшими, а затем исключаются из новых версий. Изменения могут касаться не только функциональности, но и порядка установки, обновления и повседневной работы с программным обеспечением. Это в полной мере касается и СУБД MySQL. Долгое время MySQL была "облегченной" СУБД, в которой не были реализованы многие положения стандарта SQL. В последние годы

эти пробелы по большей части устранены, и каждая новая версия предлагает пользователям новую функциональность, приближая MySQL к коммерческим флагманам отрасли (Oracle, MS SQL).

Следить за последними изменениями в СУБД MySQL удобно по официальному справочному руководству, которое можно загрузить в различных форматах с сайта компании MySQL AB (**http://dev.mysql.com/doc/**).

До версии MySQL 5.0 для документации был предусмотрен единый вариант, в котором помечалось, какие изменения, начиная с какой версии, вступили в силу. Начиная с версии MySQL 5.0, поддерживаются три варианта документации: для устаревшей, рекомендуемой и разрабатываемой версий MySQL *(см. разд. 1.1.2)*. При этом изменения указываются лишь в рамках текущей версии. Помимо архивов документации, на официальном сайте MySQL доступны online-версии. На момент написания книги были доступны следующие справочные руководства:

- MySQL 3.23, 4.0, 4.1 — **http://dev.mysql.com/doc/refman/4.1/en/index.html**;
- MySQL 5.0 — **http://dev.mysql.com/doc/refman/5.0/en/index.html**;
- MySQL 5.1 — **http://dev.mysql.com/doc/refman/5.1/en/index.html**.

Вся документация ведется на английском языке. Русский перевод документации доступен только для версии 4.0, загрузить ее можно со страницы **http://www.softtime.ru/info/mysql40_ru.php**.

Замечание

Для пользователей операционной системы Windows удобно воспользоваться CHM-вариантом справочного руководства, который позволяет осуществлять поиск по ключевым словам.

1.2. Установка MySQL под Windows

1.2.1. Процесс установки

При работе в Windows NT/2000/XP необходимо зарегистрироваться в системе с привилегиями администратора, разархивировать дистрибутив во временный каталог, после чего запустить файл setup.exe. В результате появится окно мастера установки, показанное на рис. 1.2.

Замечание

Перед установкой сервера MySQL рекомендуется отключить FireWall, если он имеется в системе, и включить его только после того, как сервер MySQL успешно установлен. Если после этого сервер MySQL прекратит принимать запросы, в FireWall следует открыть порт 3306 (или другой порт, если значение порта будет изменено во время установки).

Замечание

Установка дистрибутива MySQL описывается на примере полного дистрибутива Windows (x86) *(см. разд. 1.1.2)*.

Рис. 1.2. Стартовое окно автоматического установщика

Для продолжения установки следует нажать кнопку **Next**, после чего откроется окно, показанное на рис. 1.3, в котором предлагается выбрать различные способы установки:

- **Typical** (Стандартный);
- **Complete** (Полный);
- **Custom** (Выборочный).

В первом случае устанавливаются стандартные компоненты, во втором — все возможные компоненты, в третьем — по выбору пользователя.

В режиме **Custom** нажмите кнопку **Next** и в появившемся окне (рис. 1.4) выберите необходимые компоненты. Компоненты, которые по умолчанию отключены, перечеркнуты красным крестиком. Для того чтобы установить предлагаемые компоненты, необходимо выделить нужный компонент и выбрать в выпадающем меню пункт **This feature will be installed on local hard drive**. Если у вас отсутствует опыт установки MySQL, при первой установке лучше ничего не менять.

Рис. 1.3. Выбор режима установки MySQL

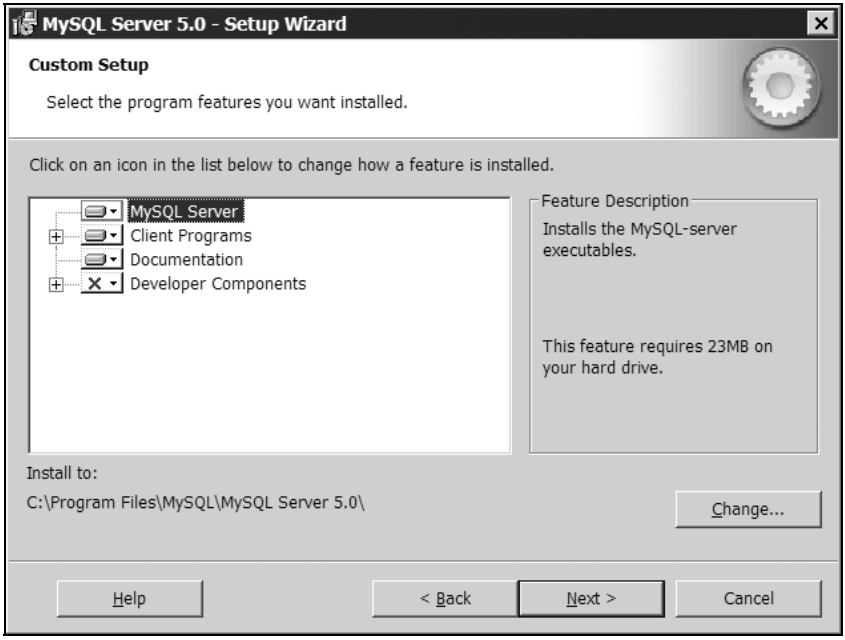


Рис. 1.4. Выбор компонентов для установки

После того как все компоненты выбраны, в этом же диалоговом окне (см. рис. 1.4) можно изменить каталог установки, нажав кнопку **Change**.

Замечание

По умолчанию каталогом установки является C:\Program Files\MySQL\MySQL Server 5.0\.

Рекомендуется изменить путь по умолчанию на более короткий, например на C:\mysql5\. Это необходимо для более комфортной работы с утилитами MySQL в командной строке. Нажав кнопку **Change**, введите путь C:\mysql5\ (рис. 1.5).

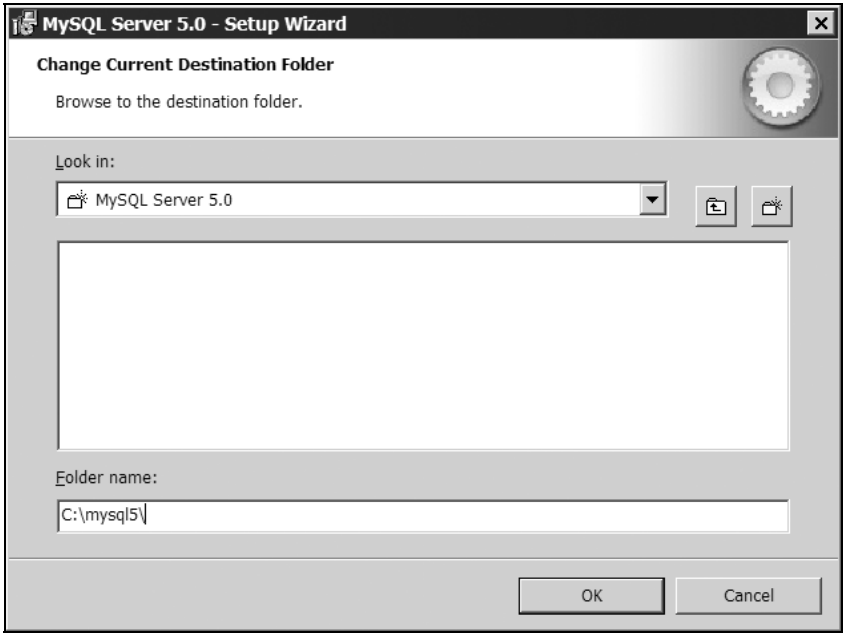


Рис. 1.5. Выбор каталога, в который будет установлен MySQL-сервер

Если вы устанавливаете MySQL поверх старой копии, например, MySQL 4.1, 4.0 или 3.23 — будьте осторожны. Такой способ вполне допустим и часто применяется для того, чтобы базы данных старой версии перешли в новую инсталляцию. Но следует помнить, что системная база данных `mysql` в настоящий момент несовместима с предыдущими версиями. Поэтому если вы производите обновление MySQL, помимо остановки сервера, следует удалить каталог C:\mysql\data\mysql системной базы данных `mysql`. При первой инсталляции проблем, связанных с несовместимостью, возникать не должно.

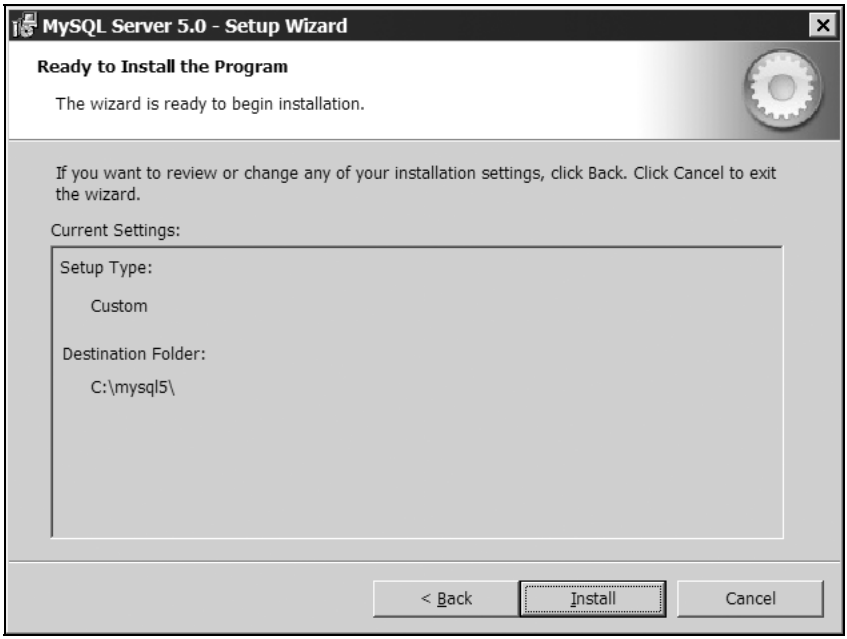


Рис. 1.6. Завершающее окно перед процессом инсталляции

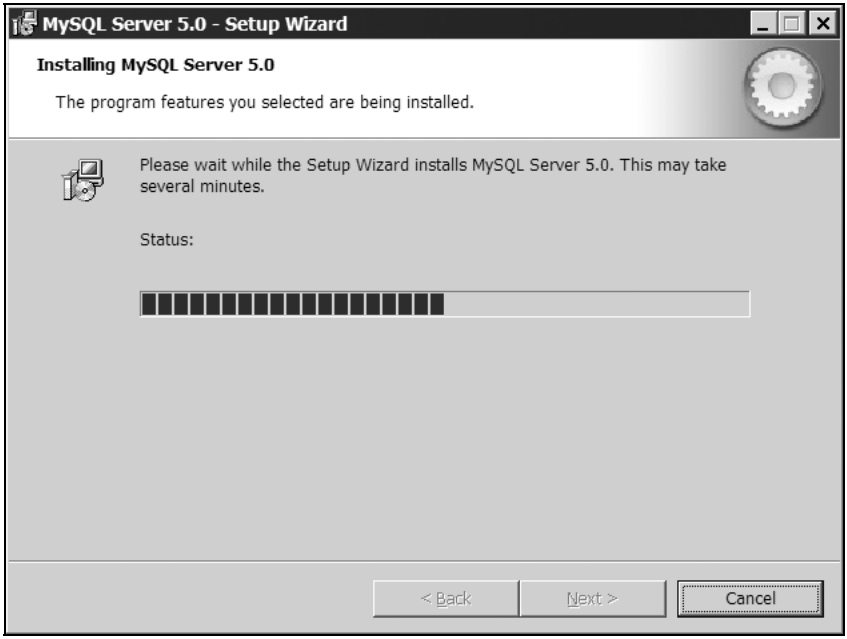


Рис. 1.7. Копирование файлов

Рис. 1.8. Регистрация на сайте **http://www.mysql.com**

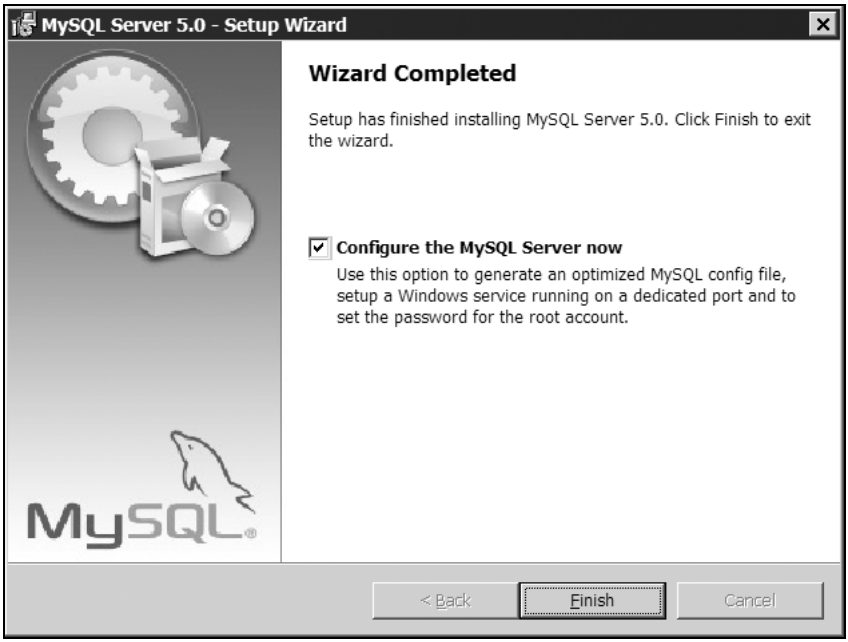


Рис. 1.9. Завершение основного этапа установки MySQL

После завершения прединсталляционной настройки выводится окно, представленное на рис. 1.6. Нажатие кнопки **Install** начинает процесс копирования файлов, представленный на рис. 1.7.

После копирования файлов мастер установки предлагает зарегистрироваться на сайте **http://www.mysql.com**. Для того чтобы пропустить эту процедуру, выберите пункт **Skip Sign-Up** (рис. 1.8). Следующее окно (рис. 1.9) сообщает о завершении основного этапа установки MySQL.

Если в данном окне не отключать возможность немедленной настройки сервера, то после нажатия кнопки **Finish** запустится утилита MySQL Server Instance Config Wizard, которая подробно рассматривается в следующем разделе.

1.2.2. Постинсталляционная настройка

Сразу после установки MySQL запускается утилита постинсталляционной настройки. Данный этап можно пропустить, учитывая, что к настройке всегда можно вернуться, выбрав пункт системного меню **Пуск | Программы | MySQL | MySQL Server 5.0 | MySQL Server Instance Config Wizard**. Тем не менее рекомендуется сразу произвести настройку системы.

Настройка начинается со стартового окна, изображенного на рис. 1.10. После нажатия кнопки **Next** открывается окно, представленное на рис. 1.11, в котором предлагается выбрать режим настройки:

- **Detailed Configuration** (Подробный);
- **Standard Configuration** (Стандартный).

Для более гибкой настройки системы следует выбрать первый пункт (**Detailed Configuration**). После нажатия кнопки **Next** открывается окно настройки производительности MySQL (рис. 1.12) со следующими опциями:

- **Developer Machine** (Машина разработчика);
- **Server Machine** (Сервер);
- **Dedicated MySQL Server Machine** (Выделенный сервер).

Все три опции различаются по интенсивности использования процессора, объему оперативной памяти и жесткого диска. Следует выбрать первый пункт, т. к. в этом случае MySQL занимает наименьший объем оперативной памяти, не мешая работе других приложений. Второй пункт предназначен для сервера, на котором, помимо MySQL, будут работать другие серверы, например, Web-сервер или транспортный почтовый агент. Третий пункт предназначен для выделенного сервера MySQL, на котором не будут выполняться никакие другие приложения.

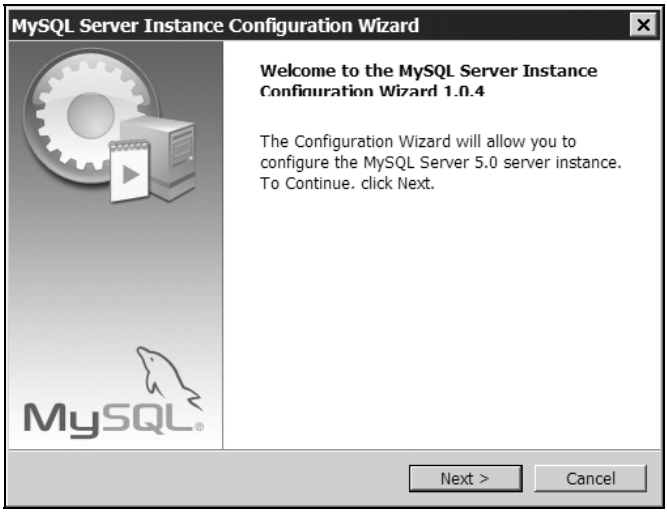


Рис. 1.10. Настройка MySQL при помощи мастера MySQL Server Instance Configuration Wizard

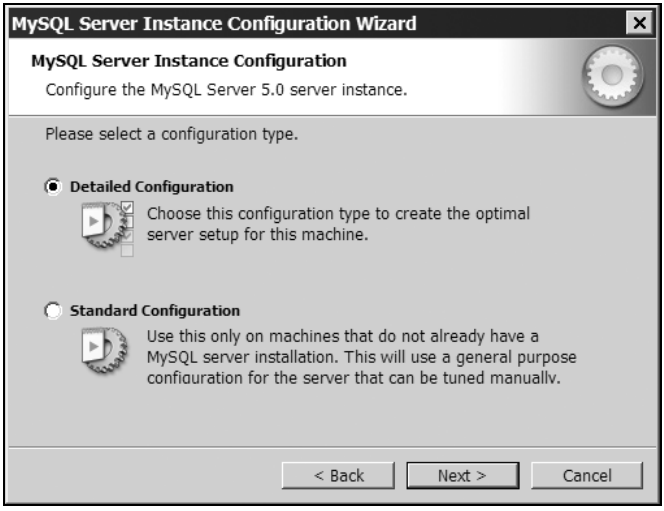


Рис. 1.11. Выбор режима настройки

Окно, представленное на рис. 1.13, позволяет выбрать предпочтительный тип для таблиц, который назначается по умолчанию. Следует оставить первый пункт.

Замечание

Результатом работы утилиты MySQL Server Instance Configuration Wizard является конфигурационный файл C:\mysql5\my.ini, который позже можно отредактировать вручную.

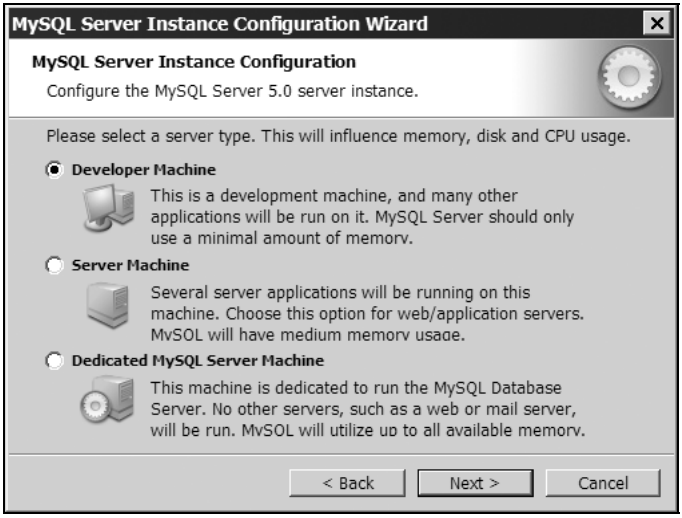


Рис. 1.12. Настройка производительности сервера MySQL

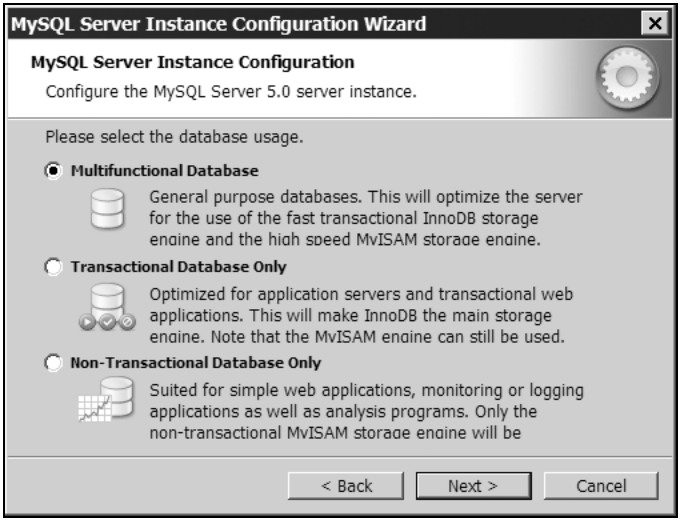


Рис. 1.13. Выбор типа таблиц по умолчанию

В окне, показанном на рис. 1.13, предлагается выбор, по сути, между двумя типами таблиц: MyISAM и InnoDB, которые более подробно обсуждаются в *главе 2*. Главное различие заключается в том, что под каждую таблицу MyISAM создается отдельный файл, который хранится в соответствующей базе данных каталога, а все таблицы InnoDB хранятся в едином табличном пространстве, под которое выделяется один файл. Кроме этого, скорость ра-

боты с MyISAM в несколько раз превышает скорость работы с таблицами типа InnoDB. Однако таблицы InnoDB поддерживают транзакции на уровне строк, более надежны и позволяют работать с таблицами большого объема. При первом знакомстве рекомендуется выбрать более простые в обслуживании и настройке таблицы типа MyISAM.

Выбор диска для хранения этого файла и пути, куда он будет установлен, осуществляется в следующем окне, представленном на рис. 1.14.

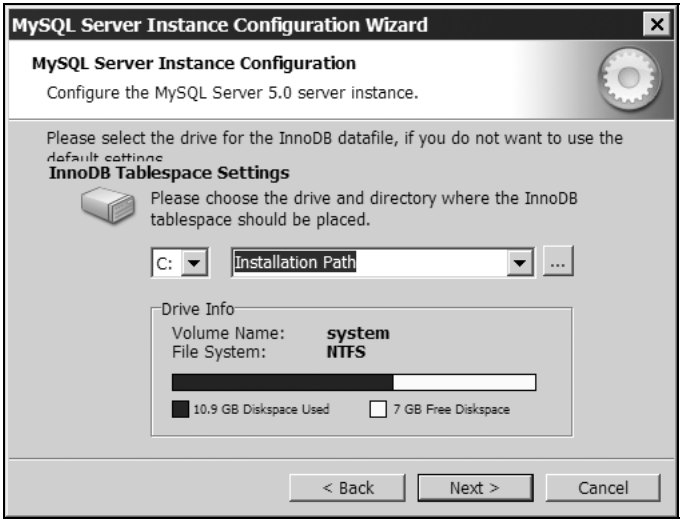


Рис. 1.14. Выбор пути для хранения файла под таблицы InnoDB

Следующее окно (рис. 1.15) предлагает указать максимальное число клиентов, которые могут одновременно подключиться к серверу. Первый пункт (рекомендуется выбрать именно его) предполагает, что число таких соединений не будет превышать 20, второй пункт допускает 500 соединений с сервером, третий пункт позволяет назначить собственный предел для числа активных соединений.

На реальных серверах не следует устанавливать это число более 200, обычно, если оперативной памяти выделено достаточно и производительность процессора (процессоров) хорошая, запросы выполняются очень быстро, не достигая этого предела. Как только один из компонентов сервера начинает потреблять лишние ресурсы, все остальные процессы ожидают выполнения и накапливаются в очереди. Так как в памяти накапливается множество невыполненных запросов, ситуация еще больше усугубляется. Ограничение количества одновременных запросов позволяет при такой критической ситуации отбрасывать все новые запросы. В результате после того, как пиковая нагрузка минует, сервер выполняет все накопившиеся в очереди запросы, и ситуа-

ция нормализуется. При отсутствии такого ограничения накапливающие запросы приведут к зависанию сервера и необходимости ручного вмешательства для его перезапуска.

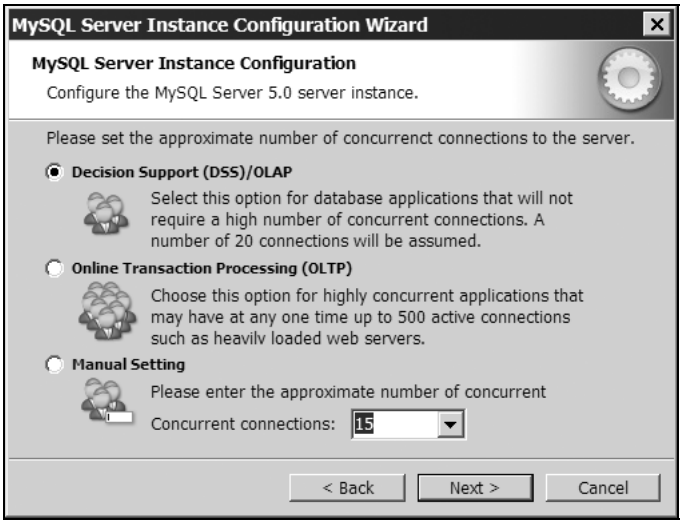


Рис. 1.15. Выбор максимального числа активных соединений с сервером

Замечание

Если ситуация с превышением максимального количества одновременных соединений воспроизводится регулярно и вызвана не утечками памяти или перегрузкой процессора, следует увеличить ограничение на количество одновременно выполняющихся запросов.

В следующем окне (рис. 1.16) устанавливается номер порта, по которому будет происходить соединение клиентов с MySQL-сервером (по умолчанию — 3306). Если на компьютере не имеется других баз данных MySQL, рекомендуется оставить это значение по умолчанию, т. к. порт 3306 является стандартным для MySQL.

Примечание

В том случае, если необходимо запускать MySQL 5 совместно с другими версиями MySQL, можно выбрать другой номер порта, отличный от тех, по которым происходит обращение к прочим MySQL-серверам.

В следующем окне предлагается указать кодировку по умолчанию (рис. 1.17). Необходимо выбрать третий пункт (ручной выбор кодировки) и в выпадающем списке выделить тип **cp1251**, соответствующий русской Windows-кодировке.

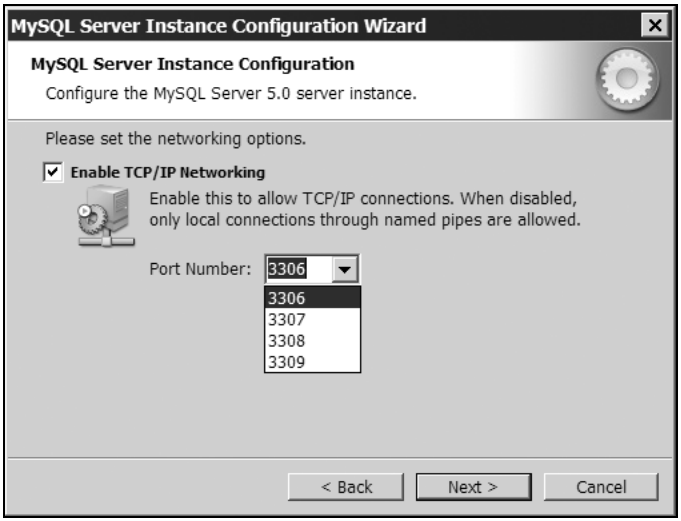


Рис. 1.16. Выбор порта, по которому сервер MySQL будет слушать клиентские запросы

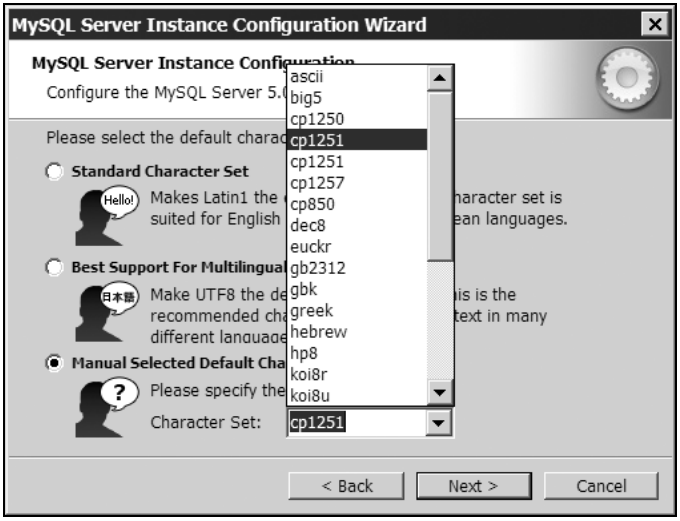


Рис. 1.17. Выбор кодировки на MySQL-сервере по умолчанию

При работе в среде Windows NT/2000/XP можно установить MySQL в качестве сервиса, что обеспечит запуск сервера mysqld.exe при старте системы и корректное завершение работы сервера при выключении компьютера.

Окно, представленное на рис. 1.18, предназначено для настройки этого сервиса. Установка флажка **Install As Windows Service** позволяет установить сервис с именем, которое следует выбрать в выпадающем списке **Service**

Name. Можно изменить имя сервиса, особенно если в системе присутствует инсталлированный MySQL-сервер более ранней версии — это позволит избежать конфликтов при запуске серверов как MySQL 5, так и более ранней версии. Отметка флажка **Launch the MySQL Server automatic** позволяет настроить сервис на автоматический режим работы, когда запуск сервера производится со стартом системы, а останов — при завершении работы с системой. В противном случае потребуются ручной запуск и останов сервера. Как будет показано далее, можно изменить режим работы сервиса в любой момент в консоли управления сервисами.

Флажок **Include Bin Directory in Windows PATH** позволяет прописать путь к каталогу C:\mysql5\bin в системной переменной `PATH`, что может быть удобным при частом использовании утилит из этого каталога.

Рис. 1.18. Настройка сервиса для автоматического запуска MySQL-сервера

В следующем окне (рис. 1.19) производится настройка учетных записей. Если вы не знакомы с системой авторизации MySQL и производите установку первый раз, рекомендуется снять флажок **Modify Security Settings** и оставить данные настройки по умолчанию. Работа с учетными записями подробно рассматривается в *главе 11*. Два текстовых поля позволяют задать пароль для суперпользователя `root` (в противном случае в качестве пароля будет выступать пустая строка). Флажок **Create An Anonymous Account** позволяет создать анонимного пользователя. Такой пользователь обладает минимальными правами, в качестве имени и пароля у него выступает пустая строка.

Последняя страница утилиты настройки MySQL-сервера MySQL Server Instance Configuration Wizard представлена на рис. 1.20.

Рис. 1.19. Настройка учетных записей

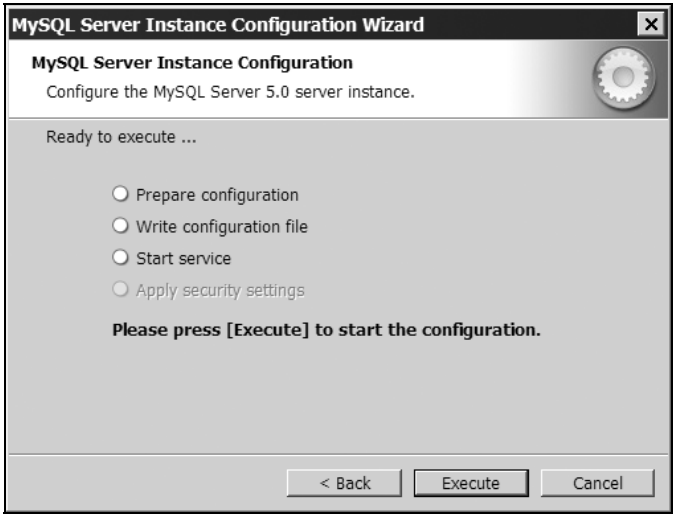


Рис. 1.20. Последняя страница утилиты MySQL Server Instance Configuration Wizard

После нажатия кнопки **Execute** будет создан конфигурационный файл C:\mysql5\my.ini и запущен сервер MySQL.

Замечание

Если при установке сервера MySQL у вас возникли затруднения, вы можете обратиться на форум по адресу **http://www.softtime.ru/forum/index.php?id_forum=3**. Авторы присутствуют на форуме каждый день и помогают читателям с установ-

кой MySQL, Web-сервера Apache и PHP уже на протяжении двух лет. На данном форуме можно также задавать любые вопросы, связанные как с администрированием, так и с программированием в среде MySQL.

1.2.3. Проверка работоспособности MySQL

После того как установка и конфигурирование MySQL завершены, необходимо убедиться в работоспособности сервера MySQL. Для этого следует запустить командную строку, выбрав в системном меню **Пуск | Программы | MySQL | MySQL Server 5.0 | MySQL Command Line Client**. Открывшееся окно может выглядеть так, как это показано на рис 1.21. На предложение ввести пароль нажмите клавишу <Enter>. После того как появилось приглашение `mysql>`, введите команду

```
SELECT VERSION();
```

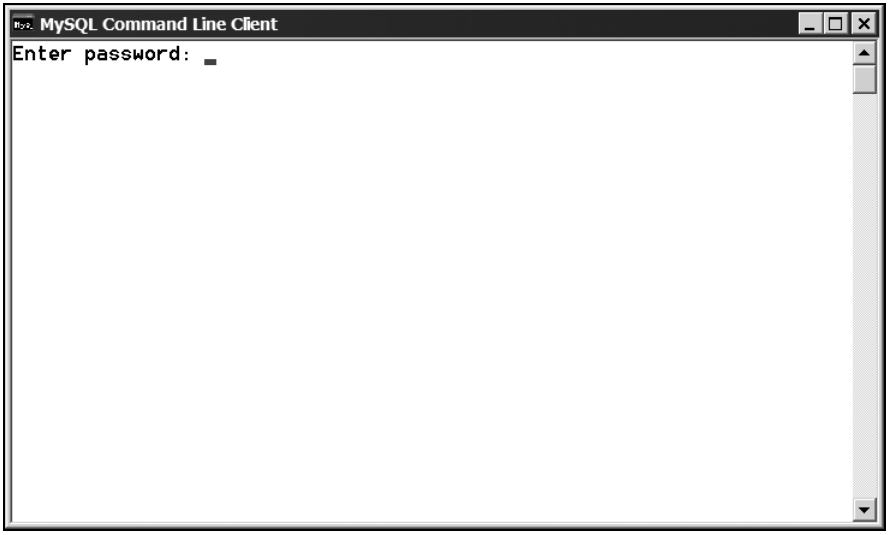


Рис. 1.21. Командная строка утилиты MySQL Command Line Client

В результате сервер должен показать текущую версию, в нашем случае это `5.0.26-community-nt` (рис. 1.22).

Замечание

Для выхода введите команду `EXIT`.

Замечание

Возможно, при обращении к утилите MySQL Command Line Client она издаст писк и закроется. В этом случае следует отредактировать файл my.ini таким об-

разом, чтобы директива `default-character-set` присутствовала только в секции `[mysqld]` и отсутствовала в секции `[mysql]` и всех других секциях.

```
MySQL Command Line Client
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.0.26-community-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT VERSION();
+---------------------+
| VERSION()           |
+---------------------+
| 5.0.26-community-nt |
+---------------------+
1 row in set (0.06 sec)

mysql>
```

Рис. 1.22. Результат выполнения команды `SELECT VERSION()`

Если сервер не запущен, то после нажатия клавиши <Enter> окно будет закрыто. Для запуска сервера следует перейти в консоль управления сервисами, выполнив команду **Пуск | Настройка | Панель управления | Администрирование | Службы**. В результате этого откроется окно, представленное на рис. 1.23.

Замечание

Проверить, запущен MySQL-сервер или нет, можно также по наличию процесса mysql-nt.exe в Диспетчере задач (см. рис. 1.1).

В консоли управления сервисами необходимо найти сервис MySQL. Если поле **Состояние** данного сервиса пусто, то он не запущен. Для его запуска следует выбрать в контекстном меню пункт **Пуск**. Для остановки сервиса необходимо выбрать пункт **Стоп**. Обратите внимание на столбец **Тип запуска**: значение **Авто** сообщает Windows о необходимости запускать сервис при старте операционной системы, значение **Вручную** означает необходимость запуска сервиса пользователем через консоль управления сервисами. Можно изменить режим запуска сервиса, выбрав в контекстном меню сервиса пункт **Свойства**.

Замечание

Для того чтобы запустить MySQL-сервер в обход сервисов, необходимо воспользоваться параметром `--standalone`.

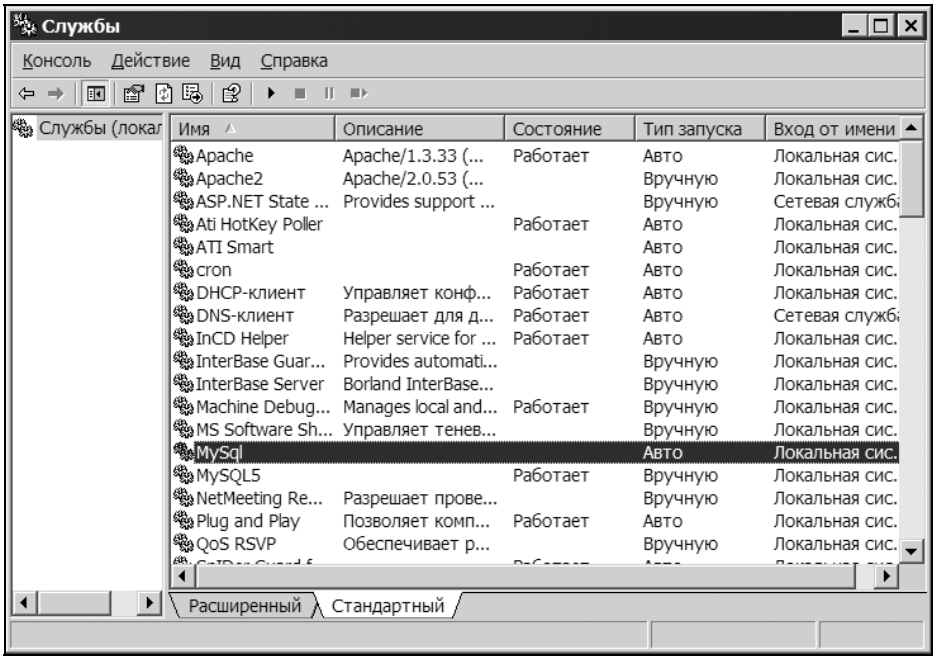


Рис. 1.23. Консоль управления сервисами

1.3. Установка MySQL под Linux

Большинство дистрибутивов Linux содержат в своем составе СУБД MySQL. В данном разделе будет рассмотрена установка сервера и клиента применительно к RedHat Linux (Fedora Core).

Несмотря на то, что в среде Linux принято устанавливать программное обеспечение из исходных кодов, и MySQL, как база данных с открытым кодом, позволяет делать это, официально рекомендуется ставить MySQL из бинарных пакетов, т. к. они оптимизированы с учетом использования конкретной архитектуры для достижения максимальной эффективности.

Замечание

Перед тем как устанавливать MySQL, необходимо остановить сервер, если он был перед этим запущен. Для этого в системах, основанных на дистрибутиве RadHat, как правило, требуется остановить демон mysql, выполнив команду из-под root: `/sbin/service mysql stop`.

Чтобы просмотреть все файлы в rpm-пакете, выполните команду:

```
rpm -qpl MySQL-VERSION.i386.rpm
```

Для того чтобы выяснить требования пакета, необходимо выполнить команду:

```
rpm -qRp MySQL-VERSION.i386.rpm
```

Для выполнения стандартной установки выполните команды, предварительно войдя в систему из-под `root`:

```
rpm -ihv MySQL-server-VERSION.i386.rpm
rpm -Uhv MySQL-client-VERSION.i386.rpm
```

Замечание

Если пакет не ставится из-за неудовлетворенных зависимостей, можно потребовать игнорировать зависимости при помощи параметра `--nodeps`. Часть зависимостей необходимо удовлетворять обязательно, часть нет: так модуль DBI для Perl требуется только для запуска тестовых скриптов, а без соответствующей версии библиотеки glibc MySQL не заработает.

Дистрибутивы RedHat и Fedora Core до версии 3 поставлялись с MySQL версии 3.23.58. Это связано с тем, что большое число приложений ориентировались именно на эту версию, и ее удаление приводило к нарушению зависимостей. Поэтому для установки более новых версий необходимо дополнительно ставить пакет `MySQL-shared-compact`, который включает в себя две динамические библиотеки для обратной совместимости: libmysqlclint.so.12 для MySQL 4.0 и libmysqlclient.so.10 для MySQL 3.23. В Fedora Core 4 этот пакет входит в состав дистрибутива, для более ранних версий необходимо загрузить пакет `MySQL-shared-compat-VERSION.i386.rpm` с официального сайта MySQL.

На загрузочной странице **http://dev.mysql.com/downloads/mysql/5.0.html** для дистрибутива, основанного на RedHat, лучше ориентироваться на раздел **Linux x86 generic RPM (statically linked against glibc 2.2.5) downloads**. В отличие от Windows-раздела, MySQL поставляется в виде отдельных компонентов, а не в виде единого rpm-пакета. Для установки сервера потребуется пакет `MySQL-server-VERSION.i386.rpm` из раздела **Server**. Для установки клиентского программного обеспечения нужен пакет `MySQL-client-VERSION.i386.rpm` из раздела **Client programs**. По желанию могут быть установлены тесты производительности **Benchmark/test suites**, библиотеки и заголовочные файлы для разработки программ **Libraries and header files**, динамические библиотеки клиентского ПО для поддержки клиентов MySQL — **Dynamic client libraries**. Раздел **Dynamic client libraries (including 3.23.x libraries)** позволяет загрузить упоминавшуюся ранее библиотеку `MySQL-shared-compat-VERSION.i386.rpm`, предназначенную для обратной совместимости.

RPM помещает данные в каталог /var/lib/mysql/ и создает соответствующие вхождения в каталоге /etc/init.d/ для автоматического запуска сервера при загрузке.

Замечание

После установки основных файлов автоматически запускается скрипт `mysql_intall_db`, который разворачивает системные базы данных. Если он не срабатывает в силу каких-либо причин (например, из-за неправильно сконфигурированного файла /etc/my.cnf), его можно запустить позже.

При обновлении уже существующей версии MySQL можно воспользоваться командами:

```
rpm -Uhv MySQL-server-VERSION.i386.rpm
rpm -Uhv MySQL-client-VERSION.i386.rpm
```

После этого можно запустить сервер при помощи команды `/usr/bin/mysqld_safe`. Для проверки работоспособности далее следует набрать команду:

```
mysql –u root
```

в ответ на которую должно открыться приглашение вида:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18 to server version: 5.0.18-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

При помощи команды `SELECT VERSION()` можно запросить версию сервера, как это описывалось в разделе, посвященном установке MySQL под Windows.

Замечание

Более подробно утилита `mysql` описывается в следующей главе. Здесь стоит лишь упомянуть, что покинуть утилиту `mysql` можно не только при помощи команды `EXIT`, но также и команды `QUIT`.

После установки новой версии инсталлятор перезаписывает скрипт автоматического запуска и остановки сервера `/etc/rc.d/init.d/mysql` сценарием `mysql.server` из дистрибутива, в результате чего он может не работать, как это, например, происходит в системе Fedora Core. Для того чтобы восстановить его работу, необходимо переписать функцию `parse_server_arguments()`, указав пути к каталогу данных и серверу, как это представлено в листинге 1.1.

Замечание

Если при установке скрипт `mysql.server` не был развернут в каталог /etc/rc.d/init.d/, его так же можно найти в каталоге /usr/share/mysql/.

Листинг 1.1. Функция `parse_server_arguments()` скрипта `/etc/rc.d/init.d/mysql`

```
parse_server_arguments() {
  for arg do
    case "$arg" in
      --basedir=*)  basedir=`echo "$arg" | sed -e 's/^[^=]*=//'`
                    bindir="/usr/bin"
                    datadir="/var/lib/mysql"
                    sbindir="/usr/sbin"
                    libexecdir="$basedir/libexec"
        ;;
      --datadir=*)  datadir=`echo "$arg" | sed -e 's/^[^=]*=//'` ;;
      --user=*)  user=`echo "$arg" | sed -e 's/^[^=]*=//'` ;;
      --pid-file=*) server_pid_file=`echo "$arg" | sed -e 's/^[^=]*=//'` ;;
      --use-mysqld_safe) use_mysqld_safe=1;;
      --use-manager)     use_mysqld_safe=0;;
    esac
  done
}
```

Скрипт `/etc/rc.d/init.d/mysql` в первую очередь предназначен для автоматического запуска и остановки сервера MySQL при старте и остановке операционной системы. Однако он вполне допускает старт и остановку сервера в ручном режиме. Для запуска `mysqld` необходимо войти в систему с правами суперпользователя и выполнить команду:

```
/etc/rc.d/init.d/mysql start
```

Следующая команда выполнит остановку сервера:

```
/etc/rc.d/init.d/mysql start
```

Помимо этого можно осуществить перезапуск сервера (это особенно полезно при изменении параметров конфигурационного файла my.ini):

```
/etc/rc.d/init.d/mysql restart
```

Для того чтобы проверить, запущен MySQL-сервер в UNIX-среде или нет, можно использовать команду:

```
ps -Af | grep mysql
```

Если сервер запущен, команда выдаст строки, соответствующие процессу `mysqld`.

1.4. Конфигурационный файл

Сервер MySQL имеет переменные состояния, которые подробно рассматриваются в *приложении 1*. Некоторые из переменных состояния могут быть изменены при помощи параметров запуска. При просмотре команды запуска сервера MySQL в свойствах сервиса *(см. разд. 1.2.3)* или при помощи команды `ps -Af` *(см. разд. 1.3)* можно заменить, что им передаются параметры запуска. *Параметром* называют ключевое слово, которое следует после имени процесса. Например, параметр `--defaults-file` в листинге 1.2 сообщает MySQL-серверу, что конфигурационный файл my.ini следует искать по адресу C:\mysql5\my.ini.

Замечание

В Windows, если не указывается параметр `--defaults-file`, конфигурационный файл ищется в корне диска C:, в UNIX-подобных операционных системах конфигурационные файлы традиционно помещаются в каталог /etc, поэтому путем по умолчанию будет /etc/my.cnf.

Листинг 1.2. Параметр запуска `--defaults-file`

```
C:\mysql5\bin\mysqld-nt.exe --defaults-file="C:\mysql5\my.ini"
```

Параметров сервера MySQL достаточно много, поэтому их может накапливаться изрядное количество. Просмотреть их полный список можно, передав серверу MySQL параметры `--verbose --help` (до версии MySQL 4.1 можно указать только параметр `--help`). Некоторые параметры имеют короткие сокращения и начинаются с одного дефиса, а не с двух. Например, параметр `--user` имеет сокращение `-u`.

Параметры запуска можно не прописывать непосредственно в строке запуска сервера, а поместить в конфигурационный файл my.ini или my.cnf, из которого сервер прочитает их при старте.

Следует отметить, что в конфигурационном файле my.ini параметры указываются без предваряющих дефисов (они не нужны, т. к. каждый параметр располагается на отдельной строке). Если строка начинается с символа диеза `#` или точки с запятой `;`, то вся строка относится к комментарию. Параметры в конфигурационном файле называют *директивами*.

В операционной системе Windows конфигурационный файл может иметь как расширение ini (такой файл может располагаться в каталогах C:\Windows\ и в C:\mysql5\), так и расширение cnf (обычно располагается в корне диска C:).

В UNIX-подобных операционных системах конфигурационный файл имеет, как правило, расширение cnf.

Конфигурационный файл влияет на работу не только сервера MySQL, но и вспомогательных утилит, таких как консольный клиент `mysql`, утилита создания SQL-дампов `mysqldump` и т. п. Более того, один конфигурационный файл может управлять работой нескольких серверов MySQL. Поэтому содержимое конфигурационного файла разделено на секции, которые имеют вид `[имя_секции]`. Имя секции определяют утилиту или сервер, к которым будут относиться перечисленные далее директивы до тех пор, пока не встретится новая секция или конец файла. В табл. 1.1 перечислены наиболее типичные секции.

Таблица 1.1. *Секции конфигурационного файла my.ini или my.cnf*

Имя секции	Описание
`[mysqld]`	Сервер MySQL
`[server]`	Сервер MySQL
`[mysqld-4.0]`	Сервер MySQL версии 4.0
`[mysqld-4.1]`	Сервер MySQL версии 4.1
`[mysqld-5.0]`	Сервер MySQL версии 5.0
`[mysqld-5.1]`	Сервер MySQL версии 5.1
`[mysqld_safe]`	Утилита запуска `mysqld_safe`
`[safe_mysqld]`	Утилита запуска `mysqld_safe`
`[mysql.server]`	Скрипт запуска `mysql.server`
`[mysqld_multi]`	Утилита запуска `mysqld_multi`
`[client]`	Любая клиентская утилита, обращающаяся к серверу
`[mysql]`	Консольный клиент `mysql`
`[mysqldump]`	Утилита создания SQL-дампов `mysqldump`
`[mysqlhotcopy]`	Утилита "горячего" копирования бинарных файлов базы данных

Наличие секции `[mysqld]` и специальных секций для разных версий обусловлены тем, что с каждой новой версией появляется все больше и больше параметров запуска, и если конфигурационный файл управляет несколькими серверами, то некоторые директивы будут одинаковыми для всех серверов, а другие уникальны для каждой из версий.

Замечание

Утилиты MySQL более подробно рассматриваются в *разд. 1.7*.

Точно такая же ситуация сложилась с секцией `[client]` и секциями для каждой из утилит. Дело в том, что каждая из утилит обладает сходными параметрами (например, параметры соединения с серверами у всех одинаковые) и в то же время имеет уникальные параметры, характерные только для ее функциональных возможностей.

В листинге 1.3 приводится пример конфигурационного файла my.ini, характерного для операционной системы Windows.

Листинг 1.3. Типичный конфигурационный файл my.ini

```
# Секция MySQL-сервера
[mysqld]
# Порт TCP/IP, который прослушивает MySQL-сервер, порт 3306 является
# стандартным, однако можно назначить любой другой не занятый порт
port=3306

# Путь установки MySQL-сервера, все остальные пути,
# если не указано другое, вычисляются относительно этого пути
basedir="C:/mysql5/"

# Путь к каталогу данных, в случае данного конфигурационного файла
# эту директиву можно не указывать, т. к. путь будет правильно
# вычислен относительно директивы basedir
datadir="C:/mysql5/Data/"

# В качестве кодировки по умолчанию для новых таблиц и баз данных
# будет выступать русская кодировка Windows-1251,
# в случае отсутствия этой директивы по умолчанию будет назначена
# кодировка latin1 (русский текст будет сортироваться неправильно)
default-character-set=cp1251

# Объем оперативной памяти, которая отводится на кэш ключей
key_buffer_size = 128M
# Максимальный размер запроса, который посылает клиент серверу
max_allowed_packet = 8M
# Объем оперативной памяти, которая отводится на кэш запросов
query_cache_size = 64M
# Максимальное количество одновременных подключений
max_connections = 200

# Секция консольного клиента mysql
[mysql]
# Пользователь по умолчанию, если клиенту mysql через параметр -u
# не будет передан текущий пользователь, будет использован root
user = root
```

В конфигурационном файле my.ini, приведенном в листинге 1.3, представлена лишь небольшая часть директив. Полный список возможных директив представлен в *приложении 1*.

1.5. Перемещение каталога данных

Важным элементом СУБД MySQL является каталог баз данных, который располагается в подкаталоге C:\mysql\data при установке MySQL под Windows и /var/lib/mysql/ в RedHat-подобных дистрибутивах Linux. Каталог данных содержит бинарные файлы баз данных, журнальные файлы, pid-файл (файл с номером идентификатора главного процесса сервера MySQL) и прочие вспомогательные файлы. Все эти файлы определяют текущее состояние, т. е., переместив их на другой сервер MySQL, можно получить текущую копию сервера.

Расположение каталога данных по умолчанию является не совсем удобным, т. к. его разумно вынести в отдельный раздел (раздел, в свою очередь, может быть монтирован на отдельный жесткий диск или нет). Расположение каталога данных в отдельном разделе позволяет безопасно переставлять операционную систему в случае сбоя или удобно проводить резервное копирование данных.

Как видно из листинга 1.3, расположение каталога данных определяется директивой `datadir`. Для того чтобы переместить каталог данных, необходимо изменить значение директивы `datadir` и переместить каталог data из каталога C:\mysql5 в новый каталог. Важно отметить, что все операции перемещения каталога необходимо выполнять, отключив предварительно сервер MySQL, в противном случае можно повредить таблицы баз данных. Отключение сервера MySQL описывается в *разд. 1.2* и *1.3*.

1.6. Обновление текущей версии

Обновлять сервер MySQL можно двумя способами: деинсталлируя предыдущую версию и устанавливая новую версию поверх старой. Какой бы способ вы не выбрали, базы данных старой версии, которые находятся в каталоге данных, не удаляются и переходят в новую версию. Проблемы, которые могут возникать, касаются лишь системной базы данных `mysql` — дело в том, что с каждой новой версией MySQL увеличивается функциональность СУБД, что требует введения новых дополнительных таблиц или изменения старых таблиц.

Замечание

Каждая база данных имеет в своем составе системную базу данных, где хранится информация о пользователях, их правах доступа, хранимые процедуры и т. п.

Если речь идет о локальной машине с версией MySQL ниже 5.0, в которой один или несколько пользователей, восстановить которых не представляет труда, перед установкой новой версии можно просто удалить системную базу данных из каталога данных (C:\mysql\data\mysql для Windows или /var/lib/mysql/mysql для Linux).

Если сервер MySQL обслуживает сотни клиентов, восстановление прав доступа которых затруднительно, или используются последние версии MySQL (5.0 и выше), включающие хранимые процедуры, лучше оставить каталог данных без изменения. Сразу после установки сервера необходимо обновить каталог данных. В UNIX-подобных операционных средах для этого используется скрипт `mysql_fix_privilege_tables` (листинг 1.4), который преобразует таблицы системной базы данных к новому формату. Скрипт пытается установить соединение с сервером базы данных от имени пользователя `root`, поэтому если пользователь `root` защищен паролем, его следует передать через параметр `--password`.

Листинг 1.4. Выполнение скрипта `mysql_fix_privilege_table`

```
mysql_fix_privilege_table --password=пароль
```

После того как скрипт выполнит свою работу, необходимо перезапустить сервер MySQL.

Для пользователей операционной системы Windows выполнение bash- и perl-скриптов обычно не доступно, поэтому Windows-дистрибутив снабжается файлом mysql_fix_privilege_tables.sql, который можно выполнить при помощи команды `SOURCE` клиента `mysql` (листинг 1.5).

Замечание

Команда `SOURCE` выполняет SQL-запросы в файле, путь к которому указывается после команды. Более подробно порядок работы с консольным клиентом `mysql` описывается в *разд. 1.8*.

Листинг 1.5. Выполнение SQL-дампа mysql_fix_privilege_tables.sql

```
SOURCE C:\mysql5\scripts\mysql_fix_privilege_tables.sql
```

1.7. Обзор утилит MySQL

Дистрибутив MySQL состоит из множества программ, каждую из которых можно отнести к одной из трех групп:

- сервер MySQL и сценарии его запуска;
- клиентские программы, устанавливающие соединение с сервером, способные работать как на локальной машине, так и удаленно, через сеть;
- программы, которые работают независимо от того, запущен MySQL-сервер или нет.

Сервер MySQL имеет имя `mysqld` в UNIX-подобных операционных системах или `mysqld-nt.exe` в Windows.

Сценарии запуска характерны для UNIX-подобных операционных систем и не поставляются в Windows-версии. К таким сценариям относят:

- `mysqld_safe` — позволяет запускать сервер `mysqld` от имени специального пользователя `mysql`, что увеличивает защиту системы в целом;
- `mysql.server` — скрипт автоматического запуска и остановки сервера MySQL;
- `mysql_multi` — сценарий, позволяющий запускать сразу несколько MySQL-серверов.

К клиентским программам, устанавливающим соединение с MySQL-сервером, относят следующие утилиты:

- `mysql` — консольный клиент для доступа к MySQL-серверу, позволяет выполнять SQL-запросы и осуществлять администрирование сервера;
- `mysqladmin` — утилита для выполнения административных функций, таких как создание или удаление баз данных, получение информации с сервера о номере версии, процессах, состоянии сервера и т. п.;
- `mysqldump` — выводит содержимое базы данных MySQL в виде файла с SQL-операторами или в виде текстовых файлов с символом табуляции в качестве разделителя. Такие файлы часто называют SQL-дампом (dump) базы данных;
- `mysqlhotcopy` — утилита для создания резервной копии таблиц без остановки сервера MySQL, т. е. создания "горячей" копии базы данных;
- `mysqlimport` — выполняет перенос информации из текстового файла в таблицы базы данных;
- `mysqlshow` — отображает информацию о существующих базах данных, таблицах, полях и индексах.

К утилитам, которые могут функционировать без подключения к серверу mysql, относятся:

- `myisampack` — сжимает таблицы типа MyISAM, уменьшая их в размере и делая доступными только для чтения;
- `mysqlcheck` — утилита, используемая для описания, проверки, оптимизации и восстановления таблиц;
- `mysqlbinlog` — данная утилита используется для чтения содержимого журнала двоичной регистрации при восстановлении данных в нештатных ситуациях;
- `perror` — утилита, которая выводит расшифровку кодов системных ошибок и ошибок MySQL;
- `replace` — утилита замены строк в файлах или стандартном потоке.

Замечание

В стандартном дистрибутиве MySQL поставляются только консольные утилиты, однако на официальном сайте MySQL представлено несколько графических клиентов, доступных для свободной загрузки. Подробнее графические клиенты описываются в *разд. 1.10*.

1.8. Утилита *mysql*

Консольный клиент `mysql` часто называют "терминальным монитором" или просто "монитором". Пользователи операционной системы Linux могут набирать имя данной программы с параметрами из любой точки системы. Владельцам Windows для запуска `mysql` необходимо выбрать следующий пункт системного меню: **Пуск | Программы | MySQL | MySQL Server 5.0 | MySQL Command Line Client**.

Замечание

Возможно, при обращении к утилите MySQL Command Line Client она издаст писк и закроется. В этом случае следует отредактировать файл my.ini таким образом, чтобы директива `default-character-set` присутствовала только в секции `[mysqld]` и отсутствовала в секции `[mysql]` и всех других секциях.

1.8.1. Командная строка

При запуске утилиты `mysql` с помощью MySQL Command Line Client нет возможности вводить какие-либо параметры. Кроме того, для работы с другими утилитами, входящими в поставку MySQL, необходим доступ к командной строке. Получить его можно, обратившись к следующему пункту: **Пуск |**

Программы | Стандартные | Командная строка. В результате будет открыто окно, представленное на рис. 1.24.

Замечание

В данной книге цвет окна и его размеры выбраны отличными от тех, что выставляются по умолчанию системой. Можно самостоятельно изменить цветовую схему командной строки и ее размеры в свойствах системного меню окна.

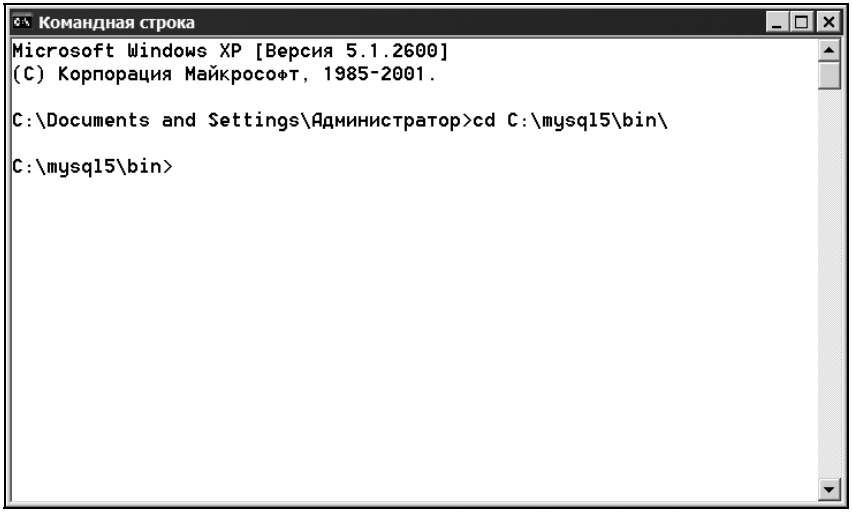


Рис. 1.24. Командная строка

После запуска командной строки необходимо перейти в подкаталог bin корневого каталога MySQL. Для этого следует набрать команду:

```
cd C:\mysql5\bin
```

В ответ будет выведено приглашение командной строки:

```
C:\mysql5\bin
```

Замечание

Если корневой каталог MySQL расположен на другом диске, например D:, необходимо перед выполнением команды `cd` сменить диск при помощи команды `D:`. Кроме этого, создав ярлык для командной строки, в свойствах ярлыка можно выставить в качестве рабочего каталога путь к каталогу bin, для того чтобы не вводить всякий раз команду `cd`.

Теперь мы находимся в каталоге bin и можем запускать расположенные в нем утилиты. Для этого достаточно набрать в командной строке имя утилиты и, если необходимо, параметры. Параметры — это символы, начинающиеся с дефиса, например, `-u`, за которыми следует их значение. Использование

различных параметров, которые будут рассмотрены в этой главе, позволяет изменять режим работы утилит.

Каждый раз набирать команды перехода в каталог bin достаточно утомительно, особенно если сервер MySQL был установлен в каталог C:\Program Files. Имеется несколько возможностей для автоматизации загрузки консольного клиента `mysql`. Первая из них заключается в создании ярлыка командной строки на рабочем столе. После чего, в контекстном меню ярлыка следует выбрать пункт **Свойства**, а в открывшемся диалоге вкладку **Ярлык**. В поле **Рабочая папка** следует ввести путь до каталога bin (на рис. 1.25 это путь C:/mysql5/bin), после чего нажать кнопку **OK** или **Применить**.

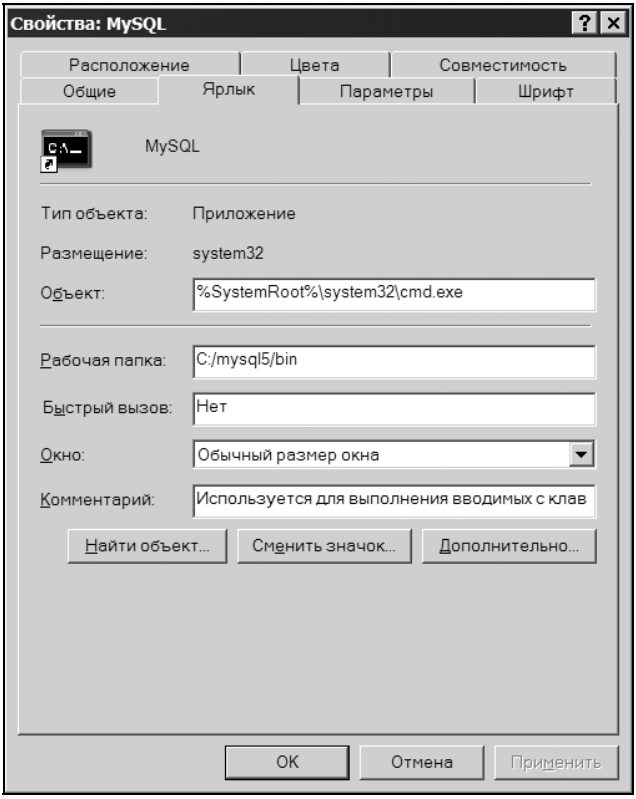


Рис. 1.25. Свойства ярлыка для командной строки

Теперь при вызове командной строки посредством ярлыка в качестве текущего каталога изначально будет выставлен каталог C:\mysql5\bin.

Второй путь автоматизации процесса запуска утилиты `mysql` заключается в том, чтобы прописать каталог C:\mysql5\bin в переменной окружения `PATH`.

Это позволит запускать утилиты из каталога bin из любого другого каталога компьютера.

Замечание

В UNIX-подобных операционных системах все утилиты помещаются либо в каталог /bin, либо в каталог /usr/sbin, которые указаны в переменной окружения `PATH` по умолчанию. Если по какой-то причине утилиты MySQL расположены в других каталогах, следует либо указывать полный путь к ним, либо прописать каталог в переменной окружения `PATH`.

Переменная окружения — это параметр, который позволяет настроить поведение операционной системы соответствующим образом: где искать в первую очередь исполняемые файлы, где хранить временные файлы, где расположен системный каталог и т. п. На рис. 1.25 в поле **Объект** используется конструкция %SystemRoot% — это один из примеров переменной окружения, которая хранит путь к каталогу Windows. Отдельные программы могут создавать собственные переменные окружения, однако некоторые переменные окружения являются стандартными. К стандартным переменным окружения относится переменная `PATH`, которая сообщает операционной системе, в каких каталогах следует искать исполняемые файлы.

В операционной системе Windows XP, для того чтобы получить доступ к переменным окружения, следует открыть диалог **Свойства системы**, выбрав в контекстном меню значка **Мой компьютер** пункт **Свойства** или выбрать следующий пункт системного меню: **Пуск | Настройка | Панель управления | Система**. В результате будет открыто диалоговое окно, изображенное на рис. 1.26.

На вкладке **Дополнительно** необходимо нажать кнопку **Переменные среды**, в результате чего будет открыто диалоговое окно, представленное на рис. 1.27.

В разделе **Системные переменные** следует отыскать переменную окружения `PATH` и дополнить ее путем к каталогу bin СУБД MySQL. Отдельные пути в значении переменной `PATH` разделяются точкой с запятой (в конце всей строки точка с запятой не требуется). Новое значение переменной окружения `PATH` вступает в силу после перезагрузки компьютера.

Еще одним важным параметром командной строки является кодировка. По умолчанию набранный к консоли русский текст будет помещен в базу данных MySQL в кодировке cp866 (DOS), что может быть очень не удобным. Для смены кодировки в консоли используется команда `chcp`, запуск которой без параметров сообщает текущую кодировку, передача команде в качестве параметра номера кодировки приводит к смене кодировки консоли. Например, для того чтобы установить кодировку Windows-1251, следует выполнить команду:

```
chcp 1251
```

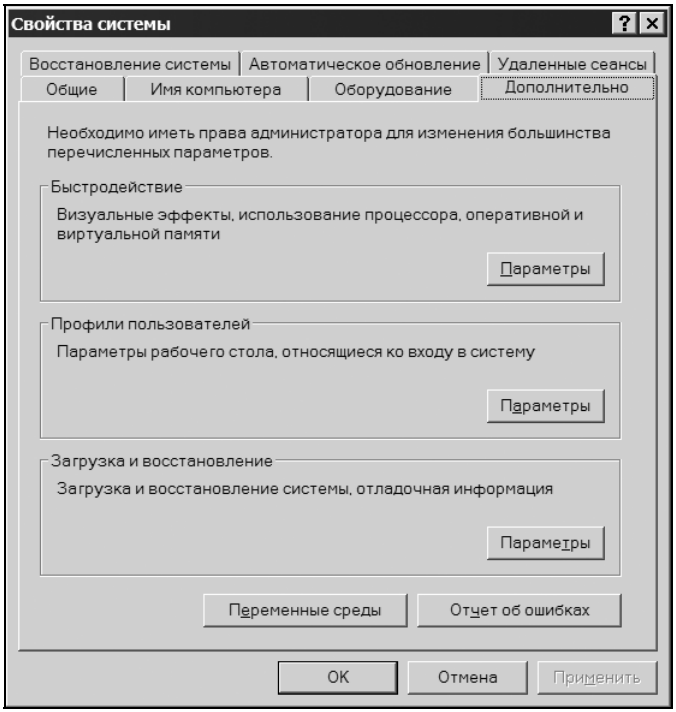


Рис. 1.26. Диалоговое окно **Свойства системы**

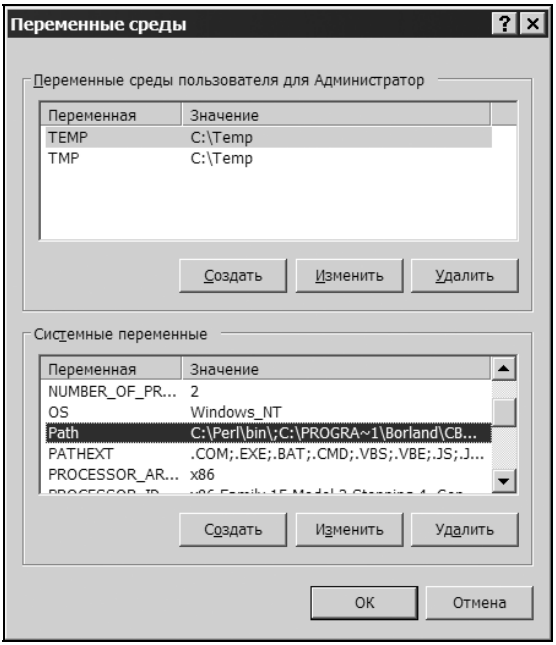


Рис. 1.27. Диалоговое окно **Переменные среды**

Замечание

Если в результате выполнения команды `chcp 1251` набираемый русский текст отображается в искаженном виде, следует в свойствах окна консоли поменять шрифт с точечного на Lucida Console.

1.8.2. Установка соединения с локальным сервером

Для соединения с сервером базы данных в параметрах `mysql` необходимо указать имя пользователя и его пароль. В только что установленной системе существует один пользователь `root`, наделенный правами администратора. Паролем по умолчанию выступает пустая строка. Поэтому для получения доступа к серверу достаточно набрать команду:

```
mysql -u root
```

Результат выполнения команды представлен на рис. 1.28. После вывода версии сервера и информации о справочных ключах выводится приглашение `mysql>`, после которого можно набирать команды.

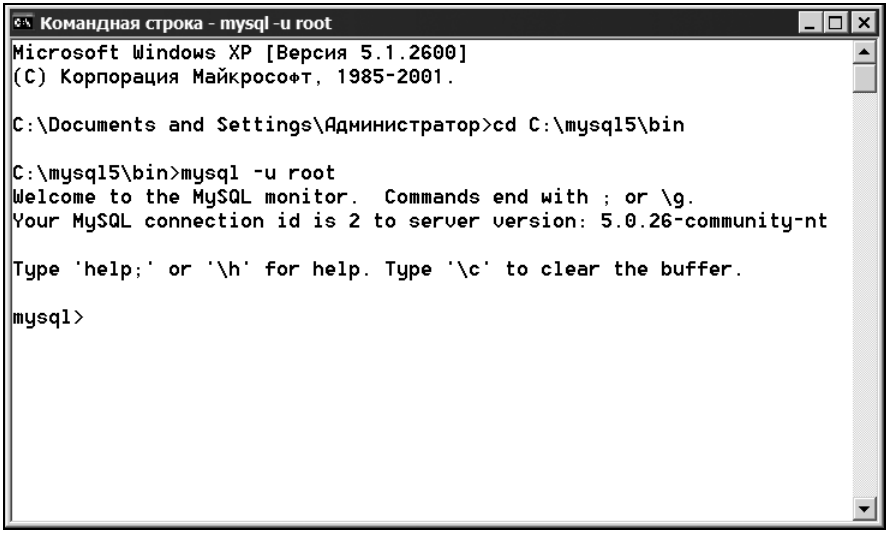


Рис. 1.28. Приглашение `mysql>` консольного клиента `mysql`

Для демонстрации полноценной авторизации создадим нового пользователя `user` с паролем `hello` при помощи инструкции `CREATE USER`, представленной в листинге 1.6. Эту инструкцию следует набрать сразу после приглашения `mysql>`, как это показано на рис. 1.29.

Листинг 1.6. Создание нового пользователя `user`

```
CREATE USER user IDENTIFIED BY 'hello';
```

После этого выйдем их оболочки `mysql` и попробуем зайти от имени нового пользователя. Выход из оболочки `mysql` производится при помощи команды `exit` или `quit` (рис. 1.29).

```
Командная строка
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.

C:\Documents and Settings\Администратор>cd C:\mysql5\bin

C:\mysql5\bin>mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 5.0.26-community-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE USER user IDENTIFIED BY 'hello';
Query OK, 0 rows affected (0.06 sec)

mysql> exit
Bye

C:\mysql5\bin>
```

Рис. 1.29. Создание пользователя `user` с паролем `hello` и выход из оболочки

Теперь команда `mysql -u user` не проходит — сервер отказывается предоставить доступ, без подтверждения полномочия паролем (рис. 1.30). Для того чтобы подсоединиться к серверу с правами пользователя `user`, необходимо передать пароль. Для этого используется параметр `-p`, сразу после которого без пробела вводится пароль. Для удобства в других параметрах допускается использование пробела между параметром и его значением. Например, следующие записи эквивалентны:

```
mysql -uuser -phello
mysql -u user -phello
```

Параметр `-p` является исключением, т. к. все символы после него воспринимаются как часть пароля.

В некоторых ситуациях требуется скрыть пароль за символами звездочки, в этом случае допускается использование параметра `-p` без значения. Утилита `mysql` запросит пароль при помощи отдельной строки `Enter password:`, в которой вводимые символы будут скрыты символом звездочки (рис. 1.31).

```
Командная строка - mysql -u user -phello

C:\mysql5\bin>mysql -u user
ERROR 1045 (28000): Access denied for user 'user'@'localhost' (using p
assword: NO)

C:\mysql5\bin>mysql -u user -phello
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12 to server version: 5.0.3-beta-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Рис. 1.30. Авторизация с использованием пароля

```
Командная строка - mysql -u user -p

C:\mysql5\bin>mysql -u user -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 5.0.26-community-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Рис. 1.31. Альтернативный способ авторизации

Команды и SQL-инструкции, за редким исключением (`exit`, `quit`, `use`), должны заканчиваться точкой с запятой. Например, на рис. 1.32 приведена SQL-инструкция, запрашивающая у сервера MySQL его версию и текущую дату.

```
Командная строка - mysql -u user -p
mysql> SELECT VERSION(), CURRENT_DATE;
+---------------------+--------------+
| VERSION()           | CURRENT_DATE |
+---------------------+--------------+
| 5.0.26-community-nt | 2006-12-13   |
+---------------------+--------------+
1 row in set (0.08 sec)

mysql> SELECT VERSION(),
    -> CURRENT_DATE
    ->
    ->
    ->
    -> ;
+---------------------+--------------+
| VERSION()           | CURRENT_DATE |
+---------------------+--------------+
| 5.0.26-community-nt | 2006-12-13   |
+---------------------+--------------+
1 row in set (0.00 sec)
```

Рис. 1.32. Выполнение команд

Замечание

Начиная с MySQL 5.0, изменить символ завершения запроса с точки запятой на новый символ, например `//`, можно либо при помощи команды `DELIMITER //`, либо указав разделитель посредством параметра `--delimiter=name`.

Когда пользователь вводит команду, она отправляется на сервер для выполнения, и, если нет ошибок в синтаксисе, на экран выводится результат в виде результирующей таблицы, а на новой строке приглашение `mysql>`, после которого можно вводить новые команды.

В первой строке таблицы с результатами содержатся заголовки столбцов, а в следующих строках — ответ сервера на запрос. Обычно заголовками столбцов становятся имена, полученные из таблиц базы. Если же извлекается не столбец таблицы, а значение выражения (как это происходит в приведенном выше примере), `mysql` дает столбцу имя запрашиваемого выражения. После этого сообщается количество возвращаемых строк (`1 row in set` — одна строка в результате) и время выполнения запроса.

Для ввода ключевых слов можно использовать любой регистр символов. Так приведенные в листинге 1.7 запросы абсолютно идентичны.

Листинг 1.7. Имена инструкций и ключевые слова не зависят от регистра символов

```
SELECT VERSION(), CURRENT_DATE;
select version(), current_date;
SeLeCt vErSiOn(), current_DATE;
```

Если команда не помещается на одной строке, возможен переход на другую строку после нажатия клавиши <Enter> — запрос отправляется серверу только после того, как консольный клиент `mysql` встретит символ точки с запятой. Приглашение командной строки после ввода первой строки этого запроса меняется с `mysql>` на `->` (см. рис. 1.32). Таким образом, программа `mysql` показывает, что завершенного выражения она пока что не получила, и ожидает его полного ввода. Точно так же утилита `mysql` ведет себя, когда ожидает завершение строки, заключенной в двойные (`"`) или одинарные (`'`) кавычки (рис. 1.33).

```
Командная строка - mysql -u user -p
mysql> SELECT "Hello world!";
+--------------+
| Hello world! |
+--------------+
| Hello world! |
+--------------+
1 row in set (0.00 sec)

mysql> SELECT "Hello
    "> world!";
+--------------+
| Hello
world! |
+--------------+
| Hello
world! |
+--------------+
1 row in set (0.00 sec)

mysql>
```

Рис. 1.33. Многострочный ввод текста, заключенного в двойные кавычки

Как видно из рис. 1.33, при переходе на следующую строку приглашение командной строки меняется с `mysql>` на `">`. Если строка заключена в одинарные кавычки, приглашение меняется на `'>`. Для того чтобы отменить текущий запрос, следует ввести последовательность `\c`.

Уже введенные ранее команды не обязательно вводить снова, для этого достаточно их вызвать клавишами <↑> и <↓> (стрелка вверх и стрелка вниз), очистить строку запроса можно при помощи клавиши <Esc>. Полный список комбинаций клавиш, которые применяются в редакторе утилиты `mysql`, приведен в табл. 1.2.

Параметры в утилитах MySQL могут иметь две формы: полную, начинающуюся с двух дефисов `--user`, и краткую, которая начинается с одного дефиса `-u`. Более подробно параметры утилиты `mysql` описываются в справочном руководстве. Далее рассматриваются наиболее интересные параметры.

Таблица 1.2. *Комбинации клавиш утилиты* `mysql`

Комбинация клавиш (UNIX)	Комбинация клавиш (Windows)	Значение
<↑>, <Ctrl>+<P>	<↑>	Вызов предыдущей команды из буфера команд
<↓>, <Ctrl>+<N>	<↓>	Вызов следующей команды из буфера команд
<←>, <Ctrl>+<B>	<←>	Перемещение курсора влево (назад)
<→>, <Ctrl>+<F>	<→>	Перемещение курсора вправо (вперед)
<Esc>+<Ctrl>+<B>	<Ctrl>+<←>	Перемещение курсора влево на одно слово
<Esc>+<Ctrl>+<F>	<Ctrl>+<→>	Перемещение курсора вправо на одно слово
<Ctrl>+<A>	<Home>	Перемещение курсора в начало строки
<Ctrl>+<E>	<End>	Перемещение курсора в конец строки
<Ctrl>+<D>	—	Удаление символа, выделенного курсором
<Delete>	<Delete>	Удаление символа слева от курсора
<Backspace>	<Backspace>	Удаление символа справа от курсора
<Esc>+<D>	—	Удаление слова
<Esc>+<Backspace>	—	Удаление слова слева от курсора
—	<Esc>	Удаление всего запроса
<Ctrl>+<K>	—	Удаление части строки от курсора до конца строки
<Ctrl>+<_>	—	Отмена последнего изменения

1.8.3. Установка соединения с удаленным сервером

Интересна возможность установки соединения с удаленным хостом. Так в домашней локальной сети одного из авторов книги, на компьютере с IP-адресом 192.168.200.1 установлен сервер MySQL версии 5.0.18. На рис. 1.34 показан процесс установки соединения с данным удаленным сервером.

На каждом компьютере по определенному порту может быть установлен только один сервер, прослушивающий данный порт. Если необходимо установить еще один сервер, его требуется установить на другой порт, например 3307, в этом случае при установке соединения в параметре `-P` следует указывать номер порта (`-P 3307`).

```
Командная строка - mysql -u root -h 192.168.200.1 -P 3306

C:\mysql5\bin>mysql -u root -h 192.168.200.1 -P 3306
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 5.0.18-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Рис. 1.34. Установка соединения с удаленным сервером 192.168.200.1

Замечание

Для успешной установки соединения на сервере должны быть прописаны IP-адреса, с которых пользователь может обращаться к MySQL-серверу. Если планируется использовать MySQL-сервер в локальной сети, ознакомьтесь с *главой 11*, в которой рассматриваются учетные записи MySQL.

1.8.4. Вертикальный режим вывода

Иногда результирующие таблицы содержат слишком большое число столбцов (рис. 1.35). В итоге структура результирующей таблицы нарушается, и воспринимать информацию практически невозможно.

Выходом из ситуации является использование так называемого вертикального режима вывода, включить который можно, задав для утилиты `mysql` параметр `--vertical`. Результат приведенного на рис. 1.35 запроса может выглядеть так, как это представлено на рис. 1.36.

В таком режиме каждая строка таблицы выводится при помощи отдельной таблицы, в первом столбце которой перечисляются имена столбцов, а во втором — их значения для данной строки.

Иногда вертикальный режим неудобно включать в начале работы с утилитой `mysql`. Для этого используются внутренние команды `mysql`. Так для включения вертикального режима используется команда `\G`, которую следует расположить после запроса (рис. 1.37).

Замечание

При использовании команды `\G` запрос не нужно завершать точкой с запятой.

```
Командная строка - mysql -u root -h 192.168.200.1
mysql> SHOW INDEX FROM db;
+-------+------------+----------+--------------+-------------+--------
---+-------------+----------+--------+------+------------+---------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collati
on | Cardinality | Sub_part | Packed | Null | Index_type | Comment |
+-------+------------+----------+--------------+-------------+--------
---+-------------+----------+--------+------+------------+---------+
| db    |          0 | PRIMARY  |            1 | Host        | A
   |        NULL |     NULL | NULL   |      | BTREE      |         |
| db    |          0 | PRIMARY  |            2 | Db          | A
   |        NULL |     NULL | NULL   |      | BTREE      |         |
| db    |          0 | PRIMARY  |            3 | User        | A
   |           2 |     NULL | NULL   |      | BTREE      |         |
| db    |          1 | User     |            1 | User        | A
   |        NULL |     NULL | NULL   |      | BTREE      |         |
+-------+------------+----------+--------------+-------------+--------
---+-------------+----------+--------+------+------------+---------+
4 rows in set (0.00 sec)

mysql>
```

Рис. 1.35. Искажение структуры результирующей таблицы из-за большого числа столбцов

```
Командная строка - mysql -u root -h 192.168.200.1 --vertical
mysql> SHOW INDEX FROM db;
*************************** 1. row ***************************
        Table: db
   Non_unique: 0
     Key_name: PRIMARY
Seq_in_index: 1
 Column_name: Host
    Collation: A
 Cardinality: NULL
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
*************************** 2. row ***************************
        Table: db
   Non_unique: 0
     Key_name: PRIMARY
Seq_in_index: 2
 Column_name: Db
```

Рис. 1.36. Вертикальный режим вывода результирующей таблицы

```
Командная строка - mysql -u root -h 192.168.200.1
mysql> SHOW INDEX FROM db\G
*************************** 1. row ***************************
        Table: db
   Non_unique: 0
     Key_name: PRIMARY
Seq_in_index: 1
  Column_name: Host
    Collation: A
  Cardinality: NULL
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
*************************** 2. row ***************************
        Table: db
   Non_unique: 0
     Key_name: PRIMARY
Seq_in_index: 2
  Column_name: Db
```

Рис. 1.37. Использование команды `\G` для вызова вертикального режима отображения результатов

1.9. Настройка кодировок

Начиная с версии 4.1, MySQL предоставляет расширенную поддержку кодировок. Существует несколько наборов символов, один набор символов может поддерживать несколько языков. Например, кодировка cp1251 может поддерживать русский, украинский и болгарские языки. Получить полный список кодировок можно при помощи оператора `SHOW CHARACTER SET`. В листинге 1.8 результирующая таблица ограничена при помощи конструкции `LIKE` таким образом, чтобы выводились только кодировки, начинающиеся с последовательности "cp".

Листинг 1.8. Использование оператора `SHOW CHARACTER SET`

```
SHOW CHARACTER SET LIKE 'cp%';
+---------+---------------------------+-------------------+--------+
| Charset | Description               | Default collation | Maxlen |
+---------+---------------------------+-------------------+--------+
| cp850   | DOS West European         | cp850_general_ci  |      1 |
| cp1250  | Windows Central European  | cp1250_general_ci |      1 |
| cp866   | DOS Russian               | cp866_general_ci  |      1 |
| cp852   | DOS Central European      | cp852_general_ci  |      1 |
| cp1251  | Windows Cyrillic          | cp1251_general_ci |      1 |
```

```
| cp1256   | Windows Arabic            | cp1256_general_ci |      1 |
| cp1257   | Windows Baltic            | cp1257_general_ci |      1 |
| cp932    | SJIS for Windows Japanese | cp932_japanese_ci |      2 |
+---------+---------------------------+-------------------+--------+
```

Первый столбец `Charset` результирующей таблицы сообщает обозначение кодировки в запросах и конфигурационных файлах MySQL, во втором столбце `Description` приводится описание кодировки. Третий столбец `Default collation` сообщает сортировку кодировки по умолчанию. Например, для кодировки cp1251 сортировкой по умолчанию является `cp1251_general_ci`, означающая, что если сортировка не уточняется, сравнение и сортировка текстовых значений будут выполняться в предположении, что текст является русским (а не болгарским или украинским), и сортировку следует проводить без учета регистра. Последний столбец `Maxlen` сообщает, сколько байтов отводится под один символ кодировки.

Получить полный список сортировок можно при помощи оператора `SHOW COLLATION`. В листинге 1.9 результирующая таблица оператора ограничена при помощи ключевого слова `LIKE` таким образом, чтобы выводились только сортировки, соответствующие кодировке cp1251.

Листинг 1.9. Сортировки, применяемые совместно с кодировкой cp1251

```
SHOW COLLATION LIKE 'cp1251%';
+---------------------+---------+----+---------+----------+---------+
| Collation           | Charset | Id | Default | Compiled | Sortlen |
+---------------------+---------+----+---------+----------+---------+
| cp1251_bulgarian_ci | cp1251  | 14 |         |          |       0 |
| cp1251_ukrainian_ci | cp1251  | 23 |         |          |       0 |
| cp1251_bin          | cp1251  | 50 |         |          |       0 |
| cp1251_general_ci   | cp1251  | 51 | Yes     |          |       0 |
| cp1251_general_cs   | cp1251  | 52 |         |          |       0 |
+---------------------+---------+----+---------+----------+---------+
```

Как видно из листинга 1.9, кодировка cp1251 имеет 5 сортировок со следующими значениями:

- `cp1251_bulgarian_ci` — болгарский язык (регистронезависимый);
- `cp1251_ukrainian_ci` — украинский язык (регистронезависимый);
- `cp1251_bin` — бинарная сортировка, сравнение по ASCII-кодам символов;
- `cp1251_general_ci` — многоязыковая сортировка (регистронезависимая);
- `cp1251_general_cs` — многоязыковая сортировка (зависимая от регистра).

Замечание

Суффикс `_ci` в любой сортировки обозначает независимость ее от регистра, в то время как суффикс `_cs` обозначает зависимость от регистра.

Последние две сортировки можно использовать для корректной работы с русским языком. В четвертом столбце результирующей таблицы оператора `SHOW COLLATION` отметкой `Yes` указывается сортировка по умолчанию (см. листинг 1.9). Так, если указывается только одна кодировка cp1251, без уточнения сортировки, то будет выбрана сортировка `cp1251_general_ci`. Это означает, что если в таблице (базе данных, столбце) планируется хранить русский текст в кодировке cp1251, то вполне можно использовать оператор `CREATE TABLE`, представленный в листинге 1.10.

Листинг 1.10. Создание таблицы, для хранения русского текста в кодировке cp1251

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL,
  name TINYTEXT NOT NULL
) ENGINE=MyISAM CHARACTER SET cp1251;
```

Однако если в таблице планируется хранить текст на украинском языке, потребуется явно указать сортировку, как показано в листинге 1.11, т. к. в противном случае будет выставлена кодировка по умолчанию и все операции сортировки текста будут осуществляться некорректно.

Листинг 1.11. Создание таблицы для хранения украинского текста в кодировке cp1251

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL,
  name TINYTEXT NOT NULL
) ENGINE=MyISAM CHARACTER SET cp1251 COLLATION cp1251_ukrainian_ci;
```

Настройка кодировок в СУБД MySQL производится на пяти уровнях:

- сервер;
- соединение;
- база данных;
- таблица;
- столбец.

От сервера до столбца увеличивается приоритет, т. е. кодировка, выставленная на сервере, может быть переопределена на уровне соединения, уровень соединения переопределяется уровнем базы данных и т. д. Это означает, что если база данных имеет кодировку latin1 и мы не указываем кодировки по умолчанию при создании таблицы, то таблица также будет иметь кодировку latin1. Однако если кодировка указывается при помощи ключевого слова `CHARACTER SET` (см. листинг 1.10), то, несмотря на то что база данных устанавливает кодировку по умолчанию latin1, столбцы в таблице будут отображаться и сортироваться при помощи сортировки, установленной посредством параметра `CHARACTER SET`.

Сразу после установки MySQL, если не проводились дополнительные настройки сервера, на уровне сервера устанавливается кодировка latin1 со шведской сортировкой `latin1_swedish_ci`. Для того чтобы переопределить кодировку на уровне сервера, необходимо при запуске передать серверу параметр `--default-character-set` с указанием кодировки. При необходимости при помощи параметра `--default-collation` можно указать сортировку. В листинге 1.12 приводится пример запуска сервера с кодировкой cp1251 с сортировкой по умолчанию (для русского текста) и с сортировкой `cp1251_ukrainian_ci` (для украинского текста).

Замечание

В операционной системе Linux, где сервер MySQL может запускаться при помощи скрипта `mysqld_safe`, передавать параметры `--default-character-set` и `--default-collation` можно и скрипту `mysqld_safe`.

Листинг 1.12. Запуск MySQL-сервера с кодировкой cp1251 по умолчанию

```
mysqld --default-character-set=cp1251
mysqld --default-character-set=cp1251
       --default-collation=cp1251_ukrainian_ci
```

Можно избежать передачи параметров всякий раз при запуске сервера. Для этого их можно поместить в конфигурационном файле my.cnf (my.ini) в секции `[mysqld]` (листинг 1.13).

Листинг 1.13. Установка кодировки и сортировки в конфигурационном файле my.ini

```
[mysqld]
default-character-set=cp1251
default-collation=cp1251_general_ci
```

При указании сортировки нужно следить, чтобы она соответствовала кодировке: если совместно с кодировкой будет указана сортировка из другого кодового набора, сервер не запустится.

Как уже упоминалось ранее, помимо серверного уровня настройки кодировок, существует уровень соединения (клиента с сервером). На этом уровне настройка осуществляется за счет изменения соответствующих системных переменных.

Замечание

Системные переменные более подробно рассматриваются в *главе 9* и *приложении 1*.

При установке соединения клиента с сервером сервер ожидает от клиента запросы в кодировке, которую определяет системная переменная `character_set_client`. После того как запрос получен, сервер транслирует его в кодировку, заданную в системной переменной `character_set_connection`. Сортировка задается при помощи системной переменной `collation_connection`.

Переменная `character_set_results` задает кодировку, в которой сервер возвращает результаты клиенту. Таким образом, если в базе данных хранится русский текст и клиент (или клиентский код) запрашивает результирующие таблицы или сам вставляет записи с русским текстом, то перед работой необходимо выставить системные переменные так, как это представлено в листинге 1.14.

Листинг 1.14. Установка системных переменных для работы с русским текстом

```
SET character_set_client='cp1251';
SET character_set_results='cp1251';
SET character_set_connection='cp1251';
```

Вместо трех операторов, представленных в листинге 1.14, можно воспользоваться специальным оператором `SET NAMES 'имя_кодировки'`. Оператор в листинге 1.15 по действию полностью эквивалентен операторам из листинга 1.14.

Листинг 1.15. Использование оператора `SET NAMES`

```
SET NAMES 'cp1251';
```

Помимо оператора `SET NAMES`, существует схожий оператор — `SET CHARACTER SET`, использование которого демонстрируется в листинге 1.16.

Листинг 1.16. Использование оператора `SET CHARACTER SET`

```
SET CHARACTER SET cp1251;
```

Оператор `SET CHARACTER SET` отличается от оператора `SET NAMES` тем, что не устанавливает переданную ему в качестве аргумента кодировку для системной переменной `character_set_connection`, вместо этого данной переменной присваивается имя кодировки для базы данных по умолчанию. Листинг 1.16 полностью эквивалентен листингу 1.17.

Листинг 1.17. Установка системных переменных для работы с русским текстом

```
SET character_set_client='cp1251';
SET character_set_results='cp1251';
SET collation_connection=@@collation_database;
```

Если кодировкой соединения не управлять, то в результирующих таблицах вместо русского текста появятся знаки вопроса (листинг 1.18).

Листинг 1.18. Искажение результатов из-за неправильно выбранной кодировки

```
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          3 | ????????????       |
|          4 | ??????? ?????      |
|          2 | ?????????? ?????   |
|          5 | ?????????? ?????? |
|          1 | ??????????         |
+------------+--------------------+
```

Если при запуске консольного клиента `mysql` возвращается ошибка:

```
Client does not support authentication protocol requested by server
```

следует удалить директиву `default-character-set` из секции `[mysql]` конфигурационного файл my.ini, данная директива должна остаться только в секции `[mysqld]`.

1.10. Графические клиенты MySQL

Пользователям Windows и X Window часто трудно приспособиться к командной строке. Решением этой проблемы может стать выбор клиента с графическим интерфейсом. На Web-странице **http://dev.mysql.com/downloads/**

для свободной загрузки доступны графические клиенты MySQL Administrator и MySQL Query Browser. Они доступны в пакете, загрузить который можно по ссылке **http://dev.mysql.com/downloads/gui-tools/5.0.html**.

Замечание

Поддержка и выпуск ранее популярной утилиты MySQL Control Center в настоящий момент прекращена.

Дистрибутивы этих графических клиентов можно также найти на компакт-диске, прилагаемом к данной книге (каталог clients).

Следует отметить, что среди Web-разработчиков пользуется большой популярностью Web-интерфейс phpMyAdmin, который стал стандартной панелью управления базами MySQL у российских хост-провайдеров. Его дистрибутив можно загрузить с сайта **http://www.phpmyadmin.net/**, однако для установки Web-интерфейса потребуется налаженная связка Web-сервера Apache и языка программирования PHP. Со статьями, посвященными настройке данной связки, можно ознакомиться на нашем сайте по ссылке **http://www.softtime.ru /article/index.php?id_page=9**. В случае возникновения сложностей можно обратиться на форум **http://www.softtime.ru/forum/?id_forum=3**, где мы всегда поможем с решением возникших проблем.

1.11. Перенос баз данных с одного сервера на другой

Одной из часто встречающихся задач является перенос баз данных с одного сервера на другой или просто создание резервной копии баз данных. СУБД MySQL предоставляет множество инструментов для решения этой задачи. Как правило, решения два:

- копирование бинарных файлов баз данных;
- создания SQL-дампа.

Замечание

Здесь не рассматривается процесс репликации, т. к. это по своей сути создание зеркального подчиненного сервера, на котором отражаются все изменения, производимые на главном сервере. Процесс репликации более подробно рассматривается в *главе 11*.

1.11.1. Копирование бинарных файлов

Перенос бинарных файлов основан на том факте, что таблицы типа MyISAM (основной тип таблиц в MySQL) являются платформонезависимыми, их мож-

но перемещать с одного сервера на другой, независимо от версии сервера и операционной системы, под управлением которой сервер работает.

Замечание

Создание в каталоге данных нового каталога аналогично созданию новой базы данных.

В каталоге данных, который описывается в *разд. 1.5*, для каждой базы данных заводится свой подкаталог, каждая таблица представлена тремя файлами, имена которых совпадают с именем таблицы, а расширения имеют следующий смысл:

- frm — данный файл определяет структуру таблицы, имена полей, их типы, параметры таблицы и т. п.;
- MYD — файл с этим расширением содержит данные таблицы (расширение образовано от сокращения MYData);
- MYI — файл с данным расширением содержит индексную информацию (расширение образовано от сокращения MYIndex);

Однако копирование данных непосредственно из каталога данных работающего сервера может привести к тому, что копии таблиц будут повреждены. Причем поврежденными таблицы могут оказаться даже в том случае, если MySQL-сервер вроде бы не производил обращений в текущий момент к копируемым таблицам. Поэтому перед копированием бинарных данных необходимо либо остановить сервер *(см. разд. 1.2.3 и 1.3)*, либо блокировать таблицы на запись. Блокировку таблиц удобно осуществить при помощи оператора `FLUSH TABLES` (листинг 1.19).

Листинг 1.19. Блокировка таблиц на запись

```
FLUSH TABLES WITH READ LOCK;
```

Для того чтобы блокировка осталась в силе на время копирования, следует оставить клиент включенным и не выходить из него до тех пор, пока копирование каталога данных не будет завершено.

Для того чтобы снять блокировку на запись, следует выполнить запрос `UNLOCK TABLES` (листинг 1.20).

Листинг 1.20. Снятие блокировки

```
UNLOCK TABLES;
```

В дистрибутив MySQL входит скрипт "горячего" копирования бинарных файлов баз данных `mysqlhotcopy`. Данный скрипт действует по той же схеме,

что описана ранее: блокирует таблицы базы данных `base` и копирует их бинарное представление по указанному пути /to/new/path (листинг 1.21).

Листинг 1.21. Использование скрипта `mysqlhotcopy`

```
mysqlhotcopy base /to/new/path
```

1.11.2. Создание SQL-дампа

Копирование бинарных файлов не всегда удобно, базу данных иногда требуется развернуть в другой СУБД или в СУБД MySQL более ранней версии, не поддерживающей нововведения поздних версий. В этом случае часто прибегают к созданию SQL-дампов. SQL-дамп — это текстовый файл с SQL-инструкциями, выполнение которых воссоздает базу данных.

Основным инструментом для создания SQL-дампов служит утилита `mysqldump`. Для того чтобы создать резервную копию базы данных, например `base`, необходимо выполнить команду, представленную в листинге 1.22.

Листинг 1.22. Создание дампа базы данных `base`

```
C:\mysql5\bin> mysqldump -u root base > base.sql
```

Как видно из листинга 1.22, утилита `mysqldump` принимает имя пользователя при помощи параметра `-u`. Если учетная запись защищена паролем, следует добавить параметр `-p`. После всех параметров указывается имя базы данных `shop`, для которой производится создание дампа. Так как вывод данных осуществляется в стандартный поток (за которым по умолчанию закреплен экран монитора), его следует перенаправить в файл (в листинге 1.22 — это `base.sql`). Перенаправление данных осуществляется при помощи оператора `>`. Если вместо оператора `>` использовать `>>`, то данные не будут перезаписывать уже существующий файл, а будут добавлены в конец файла.

При помощи параметра `--databases` (в сокращенной форме `-B`) можно создать дамп сразу нескольких баз данных, которые перечисляются через пробел (листинг 1.23).

Листинг 1.23. Создание дампа нескольких баз данных

```
C:\mysql5\bin> mysqldump -u root -B base mysql > base_mysql.sql
```

Так, команда, представленная в листинге 1.23, сохраняет дампы баз данных `base` и `mysql` в файл base_mysql.sql.

Если необходимо сохранить дамп всех баз данных MySQL-сервера, следует воспользоваться параметром `--all-databases` или в сокращенной форме `-A` (листинг 1.24).

Листинг 1.24. Создание дампа всех баз данных MySQL-сервера

```
mysqldump -u root --all-databases > all_databases.sql
```

Развернуть SQL-дамп на другом сервере можно при помощи утилиты `mysql` в пакетном режиме (листинг 1.25).

Листинг 1.25. Развертывание дампа базы данных с использованием `mysql`

```
mysql -u root test < base.sql
```

В листинге 1.25 данные из дампа `base.sql` перенаправляются на стандартный вход утилиты `mysql`, которая размещает таблицы базы данных `base` в базе данных `test`.

Развернуть SQL-дамп можно не только в пакетном режиме, но и в диалоговом. Самый простой способ — это воспользоваться командой `SOURCE`, которая выполняет несколько SQL-инструкций, перечисленных в SQL-дампе (листинг 1.26).

Замечание

Следует указывать абсолютный путь к файлу в команде `SOURCE` или помещать SQL-дамп в каталог bin. В последнем случае можно использовать лишь имя файла.

Листинг 1.26. Использование команды `SOURCE`

```
SOURCE base.sql;
```

Объемные текстовые файлы и многочисленные бинарные файлы таблиц зачастую не удобны для транспортировки, и их обычно упаковывают в архив. В Windows не представляет труда создать архив при помощи специализированных пакетов WinZip или WinRar. В UNIX-подобных операционных системах для создания архива каталога обычно используют утилиту `tar`. В листинге 1.27 первая команда создает архив t.tgz каталога /dir, вторая команда распаковывает содержимое архива t.tgz в каталог /dir.

Замечание

Параметр `--same-owner` позволяет сохранить владельцев файлов и каталогов при распаковке их из архива. Без использования параметра `--same-owner` вла-

дельцем будет назначен пользователь, от чьего имени производится разархивация.

Листинг 1.27. Упаковка каталога в архив

```
tar czvf t.tgz /dir
tar xjvf --same-owner t.tgz /dir
```

1.11.3. Использование операторов *SELECT* и *LOAD DATA*

Еще одним способом создания резервных копий является использование операторов `LOAD DATA` и `SELECT ... INTO OUTFILE`.

Оператор `LOAD DATA` загружает в таблицу данные из текстового файла и имеет следующий синтаксис:

```
LOAD DATA [LOCAL] INFILE 'filename' INTO TABLE tbl
```

Если задано необязательное ключевое слово `LOCAL`, то файл читается с клиентского хоста. Если же `LOCAL` не указывается, то файл должен находиться на сервере.

Создадим при помощи оператора `CREATE TABLE` таблицу `catalogs` (листинг 1.28).

Листинг 1.28. Создание таблицы `catalogs`

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL,
  name TINYTEXT NOT NULL
)
```

Пусть в каталоге C:\mysql5\bin имеется текстовый файл catalogs.txt следующего содержания:

```
1	Процессоры
2	Материнские платы
3	Видеоадаптеры
4	Жесткие диски
5	Оперативная память
```

Поля в файле должны быть разделены символом табуляции. Тогда выполнение запроса, представленного в листинге 1.29, приводит к заполнению таблицы `catalogs` строками из файла catalogs.txt.

Листинг 1.29. Использование оператора `LOAD DATA`

```
LOAD DATA INFILE 'C:\\mysql5\\bin\\catalogs.txt'
INTO TABLE catalogs;
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
```

Замечание

При формировании пути к файлу в операционной системе Windows обратные слэши необходимо экранировать, этого можно избежать, если использовать прямые слэши: `'C:/mysql5/bin/catalogs.txt'`.

Перед конструкцией `INTO TABLE` можно разместить одно из двух ключевых слов, которые управляют обработкой ситуации, когда данные из текстового файла дублируют значения первичного или уникальных ключей:

- `IGNORE` — пропуск строк с дублирующими значениями;
- `REPLACE` — замена уже существующих записей новыми.

Создадим текстовый файл newcatalog.txt следующего содержания:

```
1	Сетевые адаптеры
2	Программное обеспечение
3	Мониторы
4	Периферия
5	CD-RW/DVD
```

Загрузка данных при помощи оператора `LOAD DATA` без использования ключевых слов `IGNORE` и `REPLACE` приведет к сообщению об ошибке (листинг 1.30). Однако использование оператора `REPLACE` приводит к полной замене содержимого таблицы `catalogs`.

Листинг 1.30. Использование ключевого слова `REPLACE`

```
LOAD DATA INFILE 'C:\\mysql5\\bin\\newcatalogs.txt'
INTO TABLE catalogs;
062: Duplicate entry '1' for key 1
LOAD DATA INFILE 'C:\\mysql5\\bin\\newcatalogs.txt'
```

```
REPLACE INTO TABLE catalogs;
SELECT * FROM catalogs;
+------------+-------------------------+
| id_catalog | name                    |
+------------+-------------------------+
|          1 | Сетевые адаптеры        |
|          2 | Программное обеспечение |
|          3 | Мониторы                |
|          4 | Периферия               |
|          5 | CD-RW/DVD               |
+------------+-------------------------+
```

Ключевое слово `IGNORE` *`number`* `LINES` позволяет задать количество *`number`* строк, которые необходимо пропустить от начала файла. Эта конструкция используется для того, чтобы пропустить заголовки столбцов, часто располагаемые на первой строке текстового файла (листинг 1.31).

Листинг 1.31. Использование конструкции `IGNORE` *`number`* `LINES`

```
LOAD DATA INFILE 'C:\\mysql5\\bin\\catalogs.txt'
REPLACE INTO TABLE catalogs IGNORE 1 LINES;
```

Кроме базового синтаксиса, оператор `LOAD DATA` допускает дополнительные опции, позволяющие задать формат файла данных. Это осуществляется при помощи следующих ключевых слов:

- `LINES` — определяет формат строки данных, соответствующей записи;
- `FIELDS` — определяет формат поля данных.

Ключевое слово `LINES` позволяет задать символ начала и конца строки при помощи конструкций `STARTING BY` и `TERMINATED BY` соответственно. Так, по умолчанию строка в текстовом файле должна заканчиваться символом перевода строки `\n`. Однако при создании текстовых файлов в среде Windows строка заканчивается двумя символами `\r\n`, что может приводить к неправильной интерпретации файла данных и потере информации. При помощи конструкции `TERMINATED BY` можно переопределить символ конца строки. Представленный в листинге 1.32 SQL-запрос при использовании его в операционной системе Windows может выглядеть следующим образом.

Листинг 1.32. Использование конструкции `LINES TERMINATED BY`

```
LOAD DATA INFILE 'C:\\mysql5\\bin\\catalogs.txt'
REPLACE INTO TABLE catalogs LINES TERMINATED BY '\r\n';
```

По умолчанию в качестве начальной строки задается пустая строка, таким образом, запросы в листинге 1.33 идентичны.

Листинг 1.33. Варианты использования оператора `LOAD DATA INFILE`

```
LOAD DATA INFILE 'C:\\mysql5\\bin\\catalogs.txt'
REPLACE INTO TABLE catalogs;

LOAD DATA INFILE 'C:\\mysql5\\bin\\catalogs.txt'
REPLACE INTO TABLE catalogs
LINES STARTING BY '' TERMINATED BY '\n';
```

Ключевое слово `FIELDS` позволяет задать порядок обработки полей. Совместно с ним применяются следующие конструкции:

- `TERMINATED BY` — данная конструкция определяет символ-разделитель между полями в строке, по умолчанию он равен символу табуляции `\t`, но может быть задан и любой другой символ;
- `ENCLOSED BY` — данная конструкция определяет символ кавычек, которыми ограничиваются поля, по умолчанию он равен пустой строке, т. е. кавычки не применяются;
- `ESCAPED BY` — используется для назначения символа экранирования в полях, по умолчанию принимает значение обратного слэша `\`.

Запросы, представленные в листинге 1.34, идентичны.

Листинг 1.34. Варианты использования оператора `LOAD DATA INFILE`

```
LOAD DATA INFILE 'C:\\mysql5\\bin\\catalogs.txt'
REPLACE INTO TABLE catalogs;

LOAD DATA INFILE 'C:\\mysql5\\bin\\catalogs.txt'
REPLACE INTO TABLE catalogs
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\';
```

Пусть формат текстового файла данных таков, что поля разделены запятыми, значения обрамлены двойными кавычками, а строки заканчиваются символом `\r\n` (Windows-стиль).

```
"1","Сетевые адаптеры"
"2","Программное обеспечение"
"3","Мониторы"
"4","Периферия"
"5","CD-RW/DVD"
```

Тогда, запрос, загружающий данные из такого файла, может выглядеть так, как это представлено в листинге 1.35.

Листинг 1.35. Загрузка данных из CSV-файла

```
LOAD DATA INFILE 'C:\\mysql5\\bin\\catalogs.txt'
REPLACE INTO TABLE catalogs
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\r\n';
```

Помимо операции загрузки информации из текстового файла, можно осуществить обратную задачу — сохранить результирующую таблицу, полученную при помощи `SELECT`-запроса, в текстовый файл. Для решения данной задачи используется оператор `SELECT ... INTO OUTFILE '`*`file_name`*`'`. Эта разновидность оператора `SELECT` осуществляет запись выбранных строк в файл, указанный в *`file_name`*.

Выполнение SQL-запроса, показанного в листинге 1.36, приведет к созданию файла text.sql в каталоге bin (в Windows — C:\mysql5\bin). Помимо относительного пути, можно указывать также абсолютный путь.

Замечание

Файл *`file_name`* не должен существовать, иначе будет возвращена ошибка.

Листинг 1.36. Сохранение результирующей таблицы во внешнем файле text.sql

```
SELECT * INTO OUTFILE 'text.sql' FROM catalogs;
Query OK, 5 rows affected (0.03 sec)
```

Важной деталью при составлении запроса, сохраняющего результат `SELECT`-запроса во внешнем текстовом файле, является то, что предложение `INTO OUTFILE` располагается до ключевого слова `FROM`. Открытие данного файла в текстовом редакторе (Notepad) Windows приведет к тому, что перевод строк не производится, а вместо символа перевода строки присутствует квадратик, которым обозначаются нечитаемые символы. Здесь, так же как и в случае оператора `LOAD DATA INFILE`, по умолчанию принят UNIX-формат, т. е. перевод строк осуществляется только одним символом перевода строки `\n`, без добавления символа перевода каретки `\r\n`, который необходим для корректной работы многих Windows-программ.

Точно так же, как и для оператора `LOAD DATA INFILE`, формат текстового файла задается при помощи ключевых слов:

- `LINES` — определяет формат строки данных, соответствующей записи;
- `FIELDS` — определяет формат поля данных.

Ключевое слово `LINES` позволяет задать символы начала и конца строки при помощи конструкций `STARTING BY` и `TERMINATED BY` соответственно. Для того чтобы строка начиналась со слова "TABLE ", а заканчивалась переводом строки в стиле Windows — `\r\n`, необходимо выполнить запрос, представленный в листинге 1.37.

Листинг 1.37. Изменение формата внешнего файла при помощи `LINES`

```
SELECT * INTO OUTFILE 'text.sql'
LINES STARTING BY 'TABLE ' TERMINATED BY '\n'
FROM catalogs
ORDER BY id_catalog;
```

В результате будет создан файл следующего содержания:

```
TABLE 1 Процессоры
TABLE 2 Материнские платы
TABLE 3 Видеоадаптеры
TABLE 4 Жесткие диски
TABLE 5 Оперативная память
```

Конструкция `ORDER BY id_catalog` обеспечивает сортировку по первому полю таблицы `catalogs`.

Ключевое слово `FIELDS` позволяет задать порядок обработки полей, совместно с ним применяются следующие конструкции:

- `TERMINATED BY` — данная конструкция определяет символ-разделитель между полями в строке; по умолчанию это символ табуляции `\t`, но может быть задан любой другой символ;
- `ENCLOSED BY` — данная конструкция определяет символ кавычек, которыми ограничиваются поля; по умолчанию это пустая строка, т. е. кавычки не применяются;
- `ESCAPED BY` — используется для назначения символа экранирования в полях; по умолчанию принимает значение обратного слэша `\`.

Замечание

Конструкция `FIELDS` размещается перед конструкцией `LINES`.

Листинг 1.38. Изменение формата внешнего файла при помощи `FIELDS`

```
SELECT * INTO OUTFILE 'text.sql'
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES STARTING BY 'TABLE ' TERMINATED BY '\r\n'
FROM catalogs ORDER BY id_catalog;
```

Выполнение запроса из листинга 1.38 приводит к созданию файла text.sql следующего содержания:

```
TABLE "1","Процессоры"
TABLE "2","Материнские платы"
TABLE "3","Видеоадаптеры"
TABLE "4","Жесткие диски"
TABLE "5","Оперативная память"
```

Часто удобно формировать текстовый файл в виде инструкций SQL, как это делает утилита `mysqldump` *(см. разд. 1.11.2)*. В листинге 1.39 приведен запрос, создающий текстовый файл, каждая строка в котором оформлена в виде запроса `INSERT`.

Листинг 1.39. Создание дампа таблицы

```
SELECT * INTO OUTFILE 'text.sql'
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES STARTING BY 'INSERT INTO tbl VALUES(' TERMINATED BY ');\r\n'
FROM catalogs ORDER BY id_catalog;
```

В результате запроса, представленного в листинге 1.39, будет создан файл text.sql следующего содержания:

```
INSERT INTO tbl VALUES("1","Процессоры");
INSERT INTO tbl VALUES("2","Материнские платы");
INSERT INTO tbl VALUES("3","Видеоадаптеры");
INSERT INTO tbl VALUES("4","Жесткие диски");
INSERT INTO tbl VALUES("5","Оперативная память");
```

1.11.4. Операторы *BACKUP TABLE* и *RESTORE TABLE*

Оператор `BACKUP TABLE` копирует в резервный каталог минимальный набор файлов таблицы, необходимый для ее последующего восстановления (листинг 1.40). Копируются только файлы данных с расширением MYD и определения структуры таблицы с расширением frm. Индексный файл MYI может быть построен на основе первых двух.

Замечание

Операторы `BACKUP TABLE` и `RESTORE TABLE` могут применяться только к таблицам типа MyISAM.

Замечание

Наряду с формой BACKUP TABLE, может использоваться форма BACKUP TABLES, которая является синонимом и производит те же действия, что и оператор BACKUP TABLE.

Листинг 1.40. Использование оператора BACKUP TABLE

```
BACKUP TABLE products, users, orders TO 'd:/tmp';
+---------------+--------+----------+----------+
| Table         | Op     | Msg_type | Msg_text |
+---------------+--------+----------+----------+
| shop.products | backup | status   | OK       |
| shop.users    | backup | status   | OK       |
| shop.orders   | backup | status   | OK       |
+---------------+--------+----------+----------+
```

Важной деталью является то обстоятельство, что допускается использование только абсолютного пути к резервному каталогу. Резервный каталог, указанный после ключевого слова TO, должен существовать, иначе оператор BACKUP TABLE возвращает ошибку (листинг 1.41).

Листинг 1.41. Резервный каталог d:\tmp должен существовать

```
BACKUP TABLE products, users, orders TO 'd:/tmp';
+---------------+--------+----------+----------------------------------+
| Table         | Op     | Msg_type | Msg_text                         |
+---------------+--------+----------+----------------------------------+
| shop.products | backup | error    |Failed copying .frm file (errno: 2)|
| shop.products | backup | status   |Operation failed                  |
| shop.users    | backup | error    |Failed copying .frm file (errno: 2)|
| shop.users    | backup | status   |Operation failed                  |
| shop.orders   | backup | error    |Failed copying .frm file (errno: 2)|
| shop.orders   | backup | status   |Operation failed                  |
+---------------+--------+----------+----------------------------------+
```

При резервном копировании каждая таблица последовательно блокируется по чтению (READ). Если в резервной копии требуется получить согласованный набор данных, для предотвращения изменения таблиц во время резервирования требуется перед вызовом оператора BACKUP TABLE вызвать блокировку таблиц по чтению (листинг 1.42). Следует помнить, что в операторе LOCK TABLES необходимо перечислить все таблицы, которые появляются в запросах после установки блокировки, в противном случае запрос выполнить не удастся.

Листинг 1.42. Предварительная блокировка таблиц

```
LOCK TABLES products READ, users READ, orders READ;
BACKUP TABLE products, users, orders TO 'd:/tmp';
+---------------+--------+----------+----------+
| Table         | Op     | Msg_type | Msg_text |
+---------------+--------+----------+----------+
| shop.products | backup | status   | OK       |
| shop.users    | backup | status   | OK       |
| shop.orders   | backup | status   | OK       |
+---------------+--------+----------+----------+
```

Оператор `RESTORE TABLE` (листинг 1.43) восстанавливает таблицу из резервной копии, созданной при помощи оператора `BACKUP TABLE`.

Замечание

Наряду с формой `RESTORE TABLE`, может использоваться форма `RESTORE TABLES`, которая является синонимом и производит те же действия, что и оператор `RESTORE TABLE`.

Листинг 1.43. Использование оператора `RESTORE TABLE`

```
RESTORE TABLE products, users, orders FROM 'd:/tmp';
+---------------+---------+----------+----------+
| Table         | Op      | Msg_type | Msg_text |
+---------------+---------+----------+----------+
| shop.products | restore | status   | OK       |
| shop.users    | restore | status   | OK       |
| shop.orders   | restore | status   | OK       |
+---------------+---------+----------+----------+
```

Существующие таблицы не перезаписываются. Если производится попытка восстановить таблицы поверх существующих таблиц, то возвращается сообщение об ошибке '`table exists, will not overwrite on restore`' (таблица существует, невозможно осуществить перезапись при восстановлении) (листинг 1.44).

Листинг 1.44. Восстанавливаемых таблиц в базе данных не должно быть

```
RESTORE TABLE products, users, orders FROM 'd:/tmp';
+----------+---------+----------+-----------------------------------+
| Table    | Op      | Msg_type | Msg_text                          |
+----------+---------+----------+-----------------------------------+
| products | restore | error    | table exists, will not overwrite on
                                  restore                            |
```

```
| users    | restore | error    | table exists, will not overwrite on
                                  restore                            |
| orders   | restore | error    | table exists, will not overwrite on
                                  restore                            |
+----------+---------+----------+-----------------------------------+
```

Точно так же, как и в случае оператора `BACKUP TABLE`, оператор `RESTORE TABLE` требует указания абсолютного пути к резервному каталогу.

Резервная копия каждой таблицы состоит из его файлов данных MYD и файла структуры таблицы frm. При восстановлении таблицы эти файлы копируются в каталог базы данных, после чего по ним восстанавливается индексный файл с расширением MYI. Восстановление при помощи оператора `RESTORE TABLE` занимает больше времени, чем резервное копирование при помощи конструкции `BACKUP TABLE`, т. к. требуется воссоздание индексов. Чем больше у таблицы индексов, тем больше требуется времени на восстановление.

1.12. Журнальные файлы MySQL

MySQL поддерживает несколько видов журнальных файлов, в которых регистрируются различные события, происходящие на MySQL-сервере (табл. 1.3).

Таблица 1.3. *Журнальные файлы MySQL*

Журнальный файл	Регистрируемая информация
Журнал ошибок	В журнал помещаются сообщения обо всех ошибках, происходящих во время запуска, работы или остановки MySQL-сервера
Общий журнал запросов	Данный журнал позволяет регистрировать все установленные клиентом соединения и выполненные запросы
Бинарный журнал регистрации	Данный журнал регистрирует все операторы, которые приводят к изменению данных. Используется для репликации
Журнал медленных запросов	В данный журнал заносятся все запросы, выполнение которых потребовало больше времени, чем указано в системной переменной `long_query_time` (в секундах)

По умолчанию все журналы создаются в каталоге данных. Заставить MySQL-сервер открыть и закрыть журнал заново (в некоторых случаях — переключиться на новый журнал) можно с помощью оператора `FLUSH LOGS`.

1.12.1. Журнальные таблицы

До версии 5.1.6 сервер размещал журналы общих и медленных запросов только в файлах. Однако, начиная с версии 5.1.6, журналы могут быть записаны как в файл, так и в специальные таблицы `general_log` и `slow_log` системной базы данных `mysql`. Допускается также одновременное сохранение журнальной информации как в таблицы, так и в файл.

Способ хранения журнальных записей выбирается при помощи параметра `--log-output[=value,...]` сервера MySQL. В качестве значения `value` могут выступать следующие три константы:

- `TABLE` — сохранять записи журналов общих и медленных запросов в таблицах `mysql.general_log` и `mysql.slow_log` соответственно;
- `FILE` — сохранять записи журналов в текстовом файле;
- `NONE` — не сохранять записи ни в таблицах, ни в файлах. Если указана данная константа, все остальные константы игнорируются.

Если не указано ни одно из значений `--log-output`, это эквивалентно запуску сервера с параметром `--log-output=TABLE`. Если необходимо регистрировать журнальную информацию и в таблицах системной базы данных `mysql`, и в журнальных файлах, параметр принимает следующий вид: `--log-output=TABLE,FILE`.

Если при запуске MySQL-сервера используются параметры `--log[=`*`file_name`*`]` и `--log-slow-queries[=`*`file_name`*`]`, то путь к журнальному файлу *`file_name`* игнорируется до тех пор, пока в параметре `--log-output` не будет указана константа `FILE`.

1.12.2. Журнал ошибок

Журнал ошибок содержит информацию о том, когда запускался и останавливался сервер MySQL, а также фиксирует любые критические ошибки, возникшие во время его работы. В журнал заносятся записи, если сервер встречает таблицу, которая нуждается в проверке и восстановлении, если работа сервера была неожиданно остановлена и т. п.

При помощи параметра `--log-error[=`*`file_name`*`]` MySQL-сервера можно указать путь к файлу *`file_name`*, в котором будет храниться журнал ошибок. Если путь *`file_name`* не указан, сервер MySQL создает в каталоге данных файл *host*.err, где *host* — имя хоста, на котором расположен сервер. При использовании оператора `FLUSH LOGS` текущий файл журнала ошибок будет переименован и сохранен с префиксом -old, а MySQL-сервер создаст новый пустой журнальный файл с именем *host*.err.

Можно перенаправить вывод сообщений об ошибках в консоль. Для этого в UNIX используется параметр `--log-error`, а в Windows — параметр `--console`.

1.12.3. Общий журнал запросов

Общий журнал запросов позволяет регистрировать все соединения клиентов и их SQL-запросы в текстовом формате. Для активации журнала MySQL-сервер необходимо запустить с параметром `--log[=file_name]` или `-l [file_name]`, где *file_name* — путь к файлу журнала. Если путь *file_name* не указывается, то журнал создается в каталоге данных, а в качестве имени используется *host*.log, где *host* — имя хоста, на котором расположен сервер.

Замечание

Начиная с версии 5.1.6, записи общего журнала запросов можно направлять не в файл, а в таблицу `general_log` системной базы `mysql` *(см. разд. 1.12.1)*.

Сервер MySQL записывает операторы в журнал в том порядке, котором он их получает. Такой порядок может отличаться от порядка, в котором эти операторы выполняются. Этим общий журнал запросов отличается от бинарного журнала регистрации, который заполняется после того, как обработан запрос.

Замечание

Перезапуск сервера и очистка журнала не приводят к созданию нового файла для общего журнала запросов.

1.12.4. Бинарный журнал регистраций

Бинарный журнал регистраций регистрирует все запросы, которые приводят к изменению состояния базы данных. Однако в журнал заносятся только те запросы, которые действительно обновляют данные. Так, операторы `UPDATE` и `DELETE`, конструкция `WHERE` которых не соответствует ни одной строке, в журнале не регистрируются. Также пропускаются даже те операторы `UPDATE`, которые присваивают столбцу уже имеющиеся у него значения.

Бинарный журнал содержит информацию о том, сколько времени заняло выполнение каждого оператора, посредством которого были внесены изменения в базу данных. Операторы, не изменяющие никаких данных, в таком журнале не регистрируются. Если требуется регистрация абсолютно всех операторов, лучше использовать общий журнал запросов *(см. разд. 1.12.3)*.

Основное назначение бинарного журнала — это предоставить возможность обновить базу данных во время операции восстановления настолько подробно, насколько это возможно, т. к. именно в бинарном журнале будут содер-

жаться все изменения, произошедшие после того, как была сделана резервная копия всей системы.

Замечание

Бинарный журнал используется также для отправки подчиненным серверам репликации изменений, произошедших на главном сервере *(см. главу 11)*.

Замечание

Активация бинарного журнала регистрации снижает производительность MySQL-сервера в среднем на 1%. Однако предоставляемые таким журналом преимущества во время восстановления данных и настройки репликации с лихвой окупают себя.

Для активации бинарного журнала регистрации необходимо запустить сервер MySQL с параметром `--log-bin[=file_name]`, где *`file_name`* — путь к файлу журнала. Если путь *`file_name`* не указывается, то журнал создается в каталоге данных, а в качестве имени используется *host*-bin, где *host* — имя хоста, на котором расположен сервер.

Замечание

Если в параметре *`file_name`* не указан путь к каталогу, а только имя файла, файл будет помещен в каталог данных. Если в имени журнального файла присутствует расширение, оно будет удалено.

Сервер MySQL присоединяет в конец имени файла бинарного журнала расширение в виде номера. Этот номер увеличивается каждый раз при запуске сервера или выполнении команд по очистке журналов (`FLUSH LOGS`, `mysqladmin flush-logs` и `mysqladmin refresh`). Новый бинарный журнал создается автоматически, когда размер текущего журнала достигает максимума, заданного в системной переменной `max_binlog_size`.

Кроме этого, создается также индексный файл host-bin.index, в котором перечисляются имена всех используемых файлов бинарного журнала. Изменить имя индексного файла бинарного журнала можно с помощью параметра `--log-bin-index[=file_name]`.

Замечание

Индексный файл бинарного журнала нельзя редактировать во время работы сервера MySQL.

Заносимую в бинарный журнал информацию можно контролировать при помощи следующих параметров.

- `--binlog-do-db=name_database`. Параметр указывает серверу MySQL регистрировать запросы в бинарном журнале только в том случае, если текущей

базой данных (т. е. базой данных, выбранной при помощи оператора `USE`) является *name_database*. При использовании данного параметра запрос, представленный в листинге 1.45, не попадет в бинарный журнал регистраций.

Листинг 1.45. Запрос, не попадающий в бинарный журнал регистраций

```
USE shop;
UPDATE seles.january SET amount = amount + 1000;
```

- `--binlog-ignore-db=`*name_database*. Параметр указывает серверу MySQL не регистрировать запросы в бинарный журнал, если текущей базой данных (т. е. базой данных, выбранной при помощи оператора `USE`) является *name_database*. Это означает, что запрос из листинга 1.46 попадет в бинарный журнал, несмотря на то, что сервер запускается с параметром `--binlog-ignore-db=sales`.

Листинг 1.46. Запрос, попадающий в бинарный журнал регистраций

```
USE shop;
UPDATE seles.january SET amount = amount + 1000;
```

Чтобы регистрировать или игнорировать обновления сразу нескольких баз данных, следует задать соответствующий параметр необходимое число раз, по одному для каждой базы данных.

Обновления, которые регистрируются в бинарном журнале или игнорируются, определяются при помощи следующих правил:

1. Установлены ли правила при помощи `--binlog-do-db` или `--binlog-ignore-db`?
   - Если нет, в бинарный журнал заносится запись и работа алгоритма завершается.
   - Если да, переходим к следующему шагу.
2. Если одно или оба правила `--binlog-do-db` или `--binlog-ignore-db` определены, осуществляется проверка, имеется ли текущая база данных (т. е. была ли какая-нибудь база данных выбрана при помощи оператора `USE`)?
   - Если нет, запись в бинарный журнал не заносится и работа алгоритма завершается.
   - Если да, переходим к следующему шагу.
3. Если текущая база данных выбрана при помощи оператора `USE`, проверяется, установлены ли какие-нибудь правила `binlog-do-db`?

- Если да, проверяется, соответствует ли текущая база данных правилам, указанным в `binlog-do-db`?
  - Если да, запись в бинарный журнал заносится и работа алгоритма завершается.
  - Если нет, запись в бинарный журнал не заносится и работа алгоритма завершается.
- Если нет, переходим к следующему шагу.

4. Осуществляется проверка, установлено ли какое-нибудь правило `binlog-ignore-db` и соответствует ли текущая база данных любому из указанных правил?
   - Если да, запись в бинарный журнал не заносится и работа алгоритма завершается.
   - Если нет, запись в бинарный журнал заносится и работа алгоритма завершается.

Клиент с привилегией `SUPER` имеет возможность отключить регистрацию своих операторов в бинарном журнале, используя оператор `SET SQL_LOG_BIN=0`.

1.12.5. Журнал медленных запросов

При запуске сервера MySQL с параметром `--log-slow-queries[=`*`file_name`*`]` создается журнальный файл *`file_name`*, в который записываются все SQL-операторы, выполнение которых заняло больше времени, чем указано в системной переменной `long_query_time` секунд. Время на начальную блокировку таблиц при выполнении запросов не учитывается.

Если значение *`file_name`* не указано, по умолчанию в качестве имени файла присваивается имя хоста с суффиксом -slow.log. Если имя файла указано, но путь к нему не задан, файл журнала размещается в каталоге данных.

Оператор записывается в журнал медленных запросов только после того, как он был выполнен, и после того, как сняты все блокировки, поэтому порядок размещения операторов в журнале может отличаться от порядка, в котором они выполнялись.

Журнал медленных запросов может использоваться с целью выявления запросов, на выполнение которых ушло слишком много времени, а значит, требующих оптимизации. Однако разобраться в слишком объемном журнале медленных запросов не так просто, поэтому предварительно лучше прогнать журнал через команду `mysqldumpslow`, которая отобразит список всех встречающихся в журнале запросов.

Начиная с версии 5.1, запросы, которые не используют индексов, также помещаются в журнал, если сервер не запускается с параметром `--log-queries-not-using-indexes`.

Начиная с версии 5.1, параметр `--log-slow-admin-statements` сервера MySQL позволяет помещать в журнал медленных запросов также и административные команды, такие как `OPTIMIZE TABLE`, `ANALYZE TABLE` и `ALTER TABLE`.

1.13. Лицензирование MySQL

СУБД MySQL распространяется по двойной лицензии. Это означает, что для некоммерческих проектов и для личного использования можно совершенно свободно использовать GPL-дистрибутив, который можно загрузить с официального сайта компании MySQL AB. Основным требованием GPL-лицензии является необходимость распространения, помимо бинарного дистрибутива, исходных кодов приложения. Более того, любой желающий может как угодно модифицировать код программы или использовать программу или ее части в составе своего приложения. Однако приложение, использующее GPL-программу или ее части, автоматически становится GPL-программой (т. е. должно распространяться с исходными кодами).

Замечание

GPL-программа — это не обязательно бесплатная программа. Разработчик может потребовать от заказчика деньги за программу, однако вместе с бинарными кодами он должен предоставить и исходные коды, а пользователь программы (получивший ее за деньги или бесплатно) может, в свою очередь, либо продавать ее, либо распространять бесплатно (обязательно предоставляя исходные коды). Поэтому GPL-программы при широком распространении, как правило, становятся бесплатными автоматически, т. к. от разработчиков не зависит процесс распространения.

GPL-лицензия не удобна для коммерческих организаций, которые хотели бы использовать СУБД MySQL как компонент собственных разработок. Так как использование GPL-дистрибутива автоматически переводило бы их разработки в разряд GPL-продуктов. Поэтому компания MySQL AB предоставляет всем заинтересованным лицам коммерческую версию дистрибутива MySQL, которая ничем не отличается от GPL-дистрибутива, кроме лицензии, позволяющей включать компоненты MySQL в состав собственных продуктов, распространяемых на коммерческой основе.

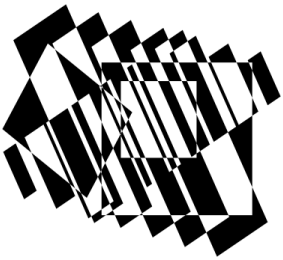
MySQL входит в состав коммерческого продукта, если:

- продукт включает в себя полноценный MySQL-сервер и предлагает его установить во время инсталляции продукта на компьютер клиента;

- продукт включает в себя встроенный MySQL-сервер;
- продукт компилируется совместно с клиентской библиотекой libmySQL, которая позволяет осуществить соединение с MySQL-сервером.

Если ни одна часть бинарного дистрибутива MySQL или его исходных кодов не используется в продукте, можно не приобретать коммерческую лицензию. В качестве примера такого приложения можно привести систему управления содержимым сайта на языке программирования PHP, использующую лишь API-интерфейс.

ГЛАВА 2



Базы данных и таблицы

2.1. Создание базы данных

В СУБД MySQL создание базы данных сводится к созданию нового подкаталога в каталоге данных. Для пользователей операционной системы Windows — это C:\mysql5\data, для пользователей RedHat-ориентированных дистрибутивов Linux — это /var/lib/mysql. Далее, упоминая путь C:\mysql5\data, мы будем иметь в виду каталог данных, который не обязательно может располагаться в каталоге C:\mysql5\data. Создание базы данных средствами SQL осуществляется при помощи оператора `CREATE DATABASE`. В листинге 2.1 приведен пример создания базы данных `wet`.

Листинг 2.1. Создание базы данных `wet`

```
CREATE DATABASE wet;
```

После выполнения запроса из листинга 4.1, заглянув в каталог C:\mysql5\data, можно обнаружить новый каталог wet. Максимальная длина имени базы данных составляет 64 знака и может включать литеры, цифры и символы _ и `$`. Имя может начинаться с цифры, но не должно полностью состоять из цифр.

Замечание

Начиная с версии 4.1.1, в каталоге базы данных создается также файл db.opt, в котором указывается кодировка, используемая в базе данных по умолчанию `default-character-set`, и порядок сортировки символов `default-collation`.

Проконтролировать создание базы данных, а также узнать имена существующих баз данных можно при помощи оператора `SHOW DATABASES` (листинг 2.2).

Листинг 2.2. Использование оператора SHOW DATABASES

```
SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| test               |
| wet                |
+--------------------+
```

Как видно из листинга 2.2, оператор SHOW DATABASES вернул имена четырех баз данных. Базы данных information_schema и mysql являются служебными и необходимы для поддержания сервера MySQL в работоспособном состоянии — в них хранится информация об учетных записях, региональных настройках и т. п.

Замечание

База данных mysql является реальной базой данных, а information_schema — виртуальной, именно поэтому в каталоге данных нет подкаталога с соответствующим именем.

Попробуйте создать в каталоге C:\mysql5\data\ новый каталог и выполнить после этого запрос SHOW DATABASES — созданная таким образом база данных будет отображена в списке баз данных. Удаление каталога приведет к исчезновению базы данных.

Удаление баз данных можно осуществить и штатными средствами при помощи оператора DROP DATABASE, за которым следует имя базы данных (листинг 2.3).

Листинг 2.3. Удаление базы данных wet

```
DROP DATABASE wet;
SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| test               |
+--------------------+
```

После выполнения оператора DROP DATABASE можно убедиться, что из каталога данных C:\mysql5\data\ был удален подкаталог wet.

Если производится попытка создания уже существующей базы данных, возвращается ошибка (листинг 2.4).

Замечание

Так как имя базы данных — это имя каталога, то чувствительность к регистру определяется конкретной операционной системой. Так в операционной системе Windows имена `wet` и `Wet` будут обозначать одну базу данных, в то время как в UNIX-подобной операционной системе это будут две разные базы данных.

Листинг 2.4. Невозможно создать базу данных `wet`: база данных существует

```
CREATE DATABASE wet;
Query OK, 1 row affected (0.00 sec)
CREATE DATABASE wet;
ERROR 1007: Can't create database 'wet'; database exists
```

Часто такое поведение нежелательно, особенно при работе с объемными SQL-дампами, которые выполняются в пакетном режиме. Для предотвращения такой ошибки оператор `CREATE DATABASE` можно снабдить конструкцией `IF NOT EXISTS`, при наличии которой база данных создается, если она еще не существует, если же существует — никакие действия не производятся (листинг 2.5).

Листинг 2.5. Использование конструкции `IF NOT EXISTS`

```
DROP DATABASE wet;
Query OK, 0 rows affected (0.00 sec)
CREATE DATABASE IF NOT EXISTS wet;
Query OK, 1 row affected (0.00 sec)
CREATE DATABASE IF NOT EXISTS wet;
Query OK, 0 rows affected (0.00 sec)
```

Как видно из листинга 2.5, после удаления базы данных `wet` первый оператор `CREATE DATABASE` возвращает ответ, в котором говорится о том, что было произведено одно действие (`1 row affected`), т. е. база данных создается. Второй оператор `CREATE DATABASE` возвращает сообщение, что запрос не произвел ни одной операции (`0 rows affected`), т. е. база данных не создается, однако и ошибка не выдается.

Точно такой же механизм разработан для оператора `DROP DATABASE`, добавление ключевого слова `IF EXISTS` удаляет базу данных, если она существует, и не производит никаких действий, если база данных отсутствует (листинг 2.6).

Листинг 2.6. Использование конструкции IF EXISTS

```
DROP DATABASE IF EXISTS wet;
Query OK, 1 rows affected (0.03 sec)
DROP DATABASE IF EXISTS wet;
Query OK, 0 rows affected (0.00 sec)
```

Для переименования базы данных предназначен оператор RENAME DATABASE, который имеет следующий синтаксис:

RENAME DATABASE *db_name* TO *new_db_name*;

Оператор переименовывает базу данных *db_name* в *new_db_name*. Следует отметить, что оператор RENAME DATABASE появился в СУБД MySQL лишь начиная с версии 5.1.7. В более ранних версиях для осуществления операции переименования необходимо было переименовывать каталог базы данных в каталоге данных C:\mysql5\data\. Пример использования оператора RENAME DATABASE приведен в листинге 2.7, где база данных wet переименовывается в complete.

Листинг 2.7. Использование оператора RENAME DATABASE

```
RENAME DATABASE wet TO complete;
Query OK, 1 rows affected (0.03 sec)
```

При создании базы данных можно указать кодировку, которая будет назначаться таблицам и столбцам по умолчанию. Для этого после имени базы данных следует указать конструкцию DEFAULT CHARACTER SET *charset_name*, где *charset_name* — имя кодировки, например, cp1251, которая обозначает русскую Windows-кодировку (листинг 2.8).

Листинг 2.8. Назначение кодировки по умолчанию

```
CREATE DATABASE wet DEFAULT CHARACTER SET cp1251;
```

Указание кодировки приводит к созданию в каталоге базы данных C:\mysql5\data\wet\ файла db.opt следующего содержания:

```
default-character-set=cp1251
default-collation=cp1251_general_ci
```

Для изменения параметров базы данных можно воспользоваться оператором ALTER DATABASE, как это продемонстрировано в листинге 2.9.

Листинг 2.9. Изменение кодировки базы данных

```
ALTER DATABASE wet CHARACTER SET cp1251
```

В листинге 2.9 для базы данных `wet` в качестве кодировки по умолчанию устанавливается cp1251.

Замечание

Оператор `ALTER DATABASE` введен в версии MySQL 4.1.1. Начиная с версии 4.1.8, допускается не указывать имя базы данных, т. е. вполне допустима запись вида `ALTER DATABASE CHARACTER SET cp1251`.

2.2. Выбор базы данных

Приступая к работе с базами данных, следует выбрать ту базу данных, к таблицам которой будет осуществляться обращение. Каждая клиентская программа решает эту задачу по-своему. Например, в консольном клиенте `mysql` выбрать новую базу данных можно при помощи `USE`. В листинге 2.10 при помощи команды `USE` в качестве текущей выбирается база данных `test`.

Листинг 2.10. Использование команды `USE`

```
USE test
```

Если текущая база данных не выбрана, то обращение по умолчанию будет заканчиваться сообщением об ошибке — "No database selected" ("База данных не выбрана") (листинг 2.11).

Листинг 2.11. База данных не выбрана

```
SHOW TABLES;
ERROR 1046 (3D000): No database selected
```

Можно не выбирать текущую базу данных, однако в этом случае во всех операторах придется явно указывать, какая база данных должна использоваться (листинг 2.12).

Листинг 2.12. Явное указание базы данных в операторе `SHOW TABLES`

```
SHOW TABLES FROM test;
```

При обращении к таблицам потребуется явное указание префикса базы данных, например, в листинге 2.13 из таблицы `User` базы данных `mysql` извлекаются поля `User` и `Host`.

Листинг 2.13. Явное указание базы данных в операторе `SELECT`

```
SELECT mysql.User.User, mysql.User.Host FROM mysql.User;
```

В результате явного обращения к базам данных SQL-запросы становятся достаточно громоздкими и менее гибкими, т. к. смена имени базы данных требует изменения SQL-запроса.

Другим способом выбрать базу данных при использовании консольного клиента `mysql` является указание имени базы данных в конце списка параметров. В листинге 2.14 при запуске утилиты `mysql` в качестве текущей будет выбрана база данных `test`.

Листинг 2.14. Выбор текущей базы данных при запуске утилиты `mysql`

```
mysql -u root test
```

В программах, обращающихся к MySQL-серверу, для выбора базы данных используется функция `mysql_select_db()` (листинг 2.15).

Замечание

Более подробно программные интерфейсы обсуждаются в *главе 18*.

Листинг 2.15. Выбор базы данных при помощи функции

```
<?php
// Адрес сервера MySQL
$dblocation = "localhost";
// Имя базы данных
$dbname = "test";
// Имя пользователя базы данных
$dbuser = "root";
// и его пароль
$dbpasswd = "";

// Устанавливаем соединение с базой данных
$dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
if (!$dbcnx) {
exit( "<P>В настоящий момент сервер базы данных не доступен,
поэтому корректное отображение страницы невозможно.</P>" );
  }
// Выбираем базу данных
if (! @mysql_select_db($dbname, $dbcnx) ) {
exit( "<P>В настоящий момент база данных не доступна,
поэтому корректное отображение страницы невозможно.</P>" );
  }
```

```
// Настраиваем кодировку соединения
@mysql_query("SET NAMES 'cp1251'");
?>
```

2.3. Создание таблиц

Информация в базе данных хранится в таблицах, создаваемых при помощи оператора `CREATE TABLE`, синтаксис которого представлен в листинге 2.16.

Замечание

Элементы в квадратных скобках являются необязательными и могут быть опущены.

Листинг 2.16. Синтаксис оператора `CREATE TABLE`

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
[(create_definition,...)]
[table_options] [select_statement]
```

Необязательная конструкция `IF NOT EXISTS` сообщает MySQL, что таблицу необходимо создавать только в том случае, если она не существует, в противном случае никаких действий предпринимать не следует (листинг 2.17).

Листинг 2.17. Создание уже существующей таблицы

```
CREATE TABLE tbl_name (k INT);
Query OK, 0 rows affected (0.11 sec)
CREATE TABLE tbl_name (k INT);
ERROR 1050: Table 'tbl_name' already exists
CREATE TABLE IF NOT EXISTS tbl_name (k INT);
Query OK, 0 rows affected (0.02 sec)
```

Как видно из листинга 2.17, повторное создание таблицы без конструкции `IF NOT EXISTS` приводит к возникновению ошибки 1050: "Таблица 'tbl_name' уже существует", однако ее применение позволяет выполнить запрос корректно. Такое поведение базы данных бывает удобным при выполнении большого числа SQL-инструкций в пакетном режиме (из файла), где ошибка вызовет остановку выполнения всего пакетного файла, несмотря на то, что дальнейшее его выполнение в такой ситуации не приведет к сбою.

Определение таблицы состоит из структуры таблицы *`create_definition`*, параметров таблицы *`table_options`* и необязательного вложенного запроса

select_statement, позволяющего создать таблицу из результата, возвращаемого оператором SELECT. Все эти конструкции будут рассмотрены далее.

2.3.1. Структура таблицы

Структура таблицы, следующая за именем таблицы *tbl_name*, представляет собой список *create_definition*, который определяет имена столбцов и индексов. В качестве элемента списка могут выступать:

- описание столбца, например, counter BIGINT;
- определение индекса.

Описание столбца включает имя столбца, следующий за ним тип и несколько необязательных ключевых слов. Имя столбца должно быть уникальным в рамках таблицы. MySQL поддерживает несколько типов данных, которые можно разбить на четыре группы:

- числовые данные — к ним относят целые числа, не содержащие дробной части (например, 124), а также вещественные числа, имеющие как целую, так и дробную части (например, 56.45);
- строковые данные — последовательность символов, заключенных в одинарные или двойные кавычки: 'Hello, world', '123', "MySQL". В качестве стандарта в SQL определяются *одинарные кавычки*, поэтому для совместимости с другими базами данных рекомендуется использовать именно их;
- календарные данные — специальный тип для обозначения даты и времени, может принимать различную форму, например, строковую "2005-04-28" или числовую 20050428. Основной характеристикой этого типа данных является их хранение в едином внутреннем формате, позволяющем осуществлять операции сложения и вычитания, независимо от внешнего представления;
- NULL — специальный тип данных, обозначающий отсутствие информации.

Числовой тип данных

В табл. 2.1 представлены характеристики и занимаемый объем числовых типов данных.

Замечание

Здесь и далее необязательные элементы синтаксиса будут заключаться в квадратные скобки, т. е. запись TINYINT[(M)] обозначает, что элемент может быть записан и как TINYINT, и как TINYINT(M).

Таблица 2.1. *Числовые типы данных*

Тип	**Объем памяти**	**Диапазон**
`TINYINT[(M)]`	1 байт	От –128 до 127 (от -2^7 до $2^7 - 1$), от 0 до 255 (от 0 до $2^8 - 1$)
`SMALLINT[(M)]`	2 байта	От –32 768 до 32 767 (от -2^{15} до $2^{15} - 1$), от 0 до 65 535 (от 0 до $2^{16} - 1$)
`MEDIUMINT[(M)]`	3 байта	От –8 388 608 до 8 388 608 (от -2^{23} до $2^{23} - 1$), от 0 до 16 777 215 (от 0 до $2^{24} - 1$)
`INT[(M)]`, `INTEGER[(M)]`	4 байта	От –2 147 683 648 до 2 147 683 648 (от -2^{31} до $2^{31} - 1$), от 0 до 4 294 967 295 (от 0 до $2^{32} - 1$)
`BIGINT[(M)]`	8 байтов	От -2^{63} до $2^{63} - 1$, от 0 до 2^{64}
`BIT[(M)]`	(`M` + 7)/8 байтов	От 1 до 64 битов, в зависимости от значения `M`
`BOOL`, `BOOLEAN`	1 байт	Либо 0, либо 1
`DECIMAL[(M[,D])]`, `DEC[(M[,D)]]`, `NUMERUC[(M[,D)]]`	`M` + 2 байта	Повышенная точность; зависит от параметров `M` и `D`
`FLOAT[(M,D)]`	4 байта	Минимальное значение $\pm 1{,}175494351 \times 10^{-39}$, максимальное значение $\pm 3{,}402823466 \times 10^{38}$
`DOUBLE[(M,D)]`, `REAL[(M,D)]`, `DOUBLE PRECISION[(M,D)]`	8 байтов	Минимальное значение $\pm 2{,}2250738585072014 \times 10^{-308}$, максимальное значение $\pm 1{,}797693134862315 \times 10^{308}$

Как видно из табл. 2.1, СУБД MySQL имеет пять целых типов: `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, `BIGINT`. Различие между ними заключается в диапазоне величин, которые можно хранить в столбцах такого типа. Чем больше диапазон значений у типа данных, тем больше памяти для него требуется.

Целые типы данных могут быть объявлены положительными. Для этого после объявления типа следует использовать ключевое слово `UNSIGNED`. В этом случае элементам данного столбца нельзя будет присвоить отрицательные значения, а допустимый диапазон, который может принимать тип, удваивается. Так тип `TINYINT` может принимать значения от –128 до 127, а `TINYINT UNSIGNED` — от 0 до 255.

При объявлении целого типа задается количество отводимых под число символов `M` (от 1 до 255). Это необязательное указание количества выводимых символов используется для дополнения пробелами слева от выводимых значений символов, меньших, чем заданная ширина столбца. Однако ограничений ни на диапазон величин, ни на количество разрядов не налагается. Если число символов, необходимых для вывода числа, превышает `M`, под столбец будет выделено больше символов. Если дополнительно указан необязательный атрибут `ZEROFILL`, свободные позиции по умолчанию заполняются нулями слева. Например, для столбца, объявленного как `INT(5) ZEROFILL`, величина 4 отображается как `00004`.

Тип `BIT[(M)]` предназначен для хранения битовых полей. Параметр `M` указывает число битовых значений, которое может принимать поле (от 1 до 64). Если параметр `M` не указан, то по умолчанию принимает значение 1.

Замечание

Тип `BIT` добавлен в MySQL, начиная с версии 5.0.3.

Тип `BOOLEAN` является синонимом для `TINYINT(1)`. Значение 1 рассматривается как истина (`true`), а 0 — как ложь (`false`).

Тип `DECIMAL`, а также его синонимы `NUMERIC` и `DEC` предназначены для величин повышенной точности, например, для денежных данных. Требуемая точность задается при объявлении столбца данных одного из этих типов, например:

```
salary DECIMAL(5,2)
```

В этом примере цифра 5 определяет общее количество символов, отводимых под число, а цифра 2 задает количество знаков после запятой. Следовательно, в этом случае интервал величин, которые могут храниться в столбце `salary`, составляет от –99,99 до 99,99 (в действительности для данного столбца MySQL обеспечивает возможность хранения чисел вплоть до 999,99, поскольку допускается не хранить знак для положительных чисел).

Замечание

Первый параметр `M` может принимать максимальное значение, равное 64, второй параметр `D` — 30.

Величины типов `DECIMAL`, `DEC` и `NUMERIC` хранятся как строки, а не как двоичные числа с плавающей точкой, чтобы сохранить точность представления этих величин в десятичном виде. Если второй параметр равен 0, то величины `DECIMAL` и `NUMERIC` не содержат десятичного знака или дробной части.

Диапазон вещественных чисел, помимо максимального значения, имеет также минимальное значение, которое характеризует точность данного типа. Параметр `M` в табл. 2.1 задает количество символов для отображения всего

числа, включая целую часть, точку и дробную часть, а D — лишь для его дробной части.

Числовые типы данных с плавающей точкой также могут иметь параметр UNSIGNED. Как и в целочисленных типах, этот атрибут предотвращает хранение в отмеченном столбце отрицательных величин, но, в отличие от целочисленных типов, максимальный интервал для величин столбца остается прежним.

При выборе столбцов для формирования структуры таблицы необходимо обращать внимание на размер, занимаемый тем или иным типом данных: если значения, размещаемые в базе данных, никогда не будут выходить за пределы 100, не следует выбирать тип больше TINYINT. Если же в полях столбца предполагается хранить только целочисленные данные, то применение атрибута UNSIGNED позволит увеличить диапазон в два раза.

В листинге 2.18 представлен оператор CREATE TABLE, создающий таблицу tbl с двумя полями:

- num1 — целочисленное поле типа INT;
- num2 — поле для вещественных чисел, тип FLOAT.

Листинг 2.18. Использование оператора CREATE TABLE

```
CREATE TABLE tbl (
num1 INT DEFAULT 0,
num2 FLOAT DEFAULT 1.0
)
```

Строковый тип данных

Для строковых типов данных максимальный размер и требования к памяти приведены в табл. 2.2. Здесь L — это длина хранимой в ячейке строки, а байты, приплюсованные к L, — накладные расходы для хранения длины строки.

Таблица 2.2. *Строковые типы данных*

Тип	Объем памяти	Максимальный размер
CHAR(M)	M символов	M символов
VARCHAR(M)	L+1 символов	M символов
TINYBLOB, TINYTEXT	L+1 символов	2^8 – 1 символов
BLOB, TEXT	L+2 символов	2^{16} – 1 символов
MEDIUMBLOB, MEDIUMTEXT	L+3 символов	2^{24} – 1 символов
LONGBLOB, LONGTEXT	L+4 символов	2^{32} – 1 символов

Таблица 2.2 *(окончание)*

Тип	Объем памяти	Максимальный размер
ENUM('value1','value2',...)	1 или 2 байта	65 535 элементов
SET('value1','value2',...)	1, 2, 3, 4 или 8 байтов	64 элемента

Тип CHAR позволяет хранить строку фиксированной длины M, его дополняет тип VARCHAR, позволяющий хранить строки переменной длины L. Величина M может принимать значения от 0 до 65 535.

Замечание

До версии MySQL 5.0.3 параметр M для типов CHAR и VARCHAR мог принимать значения только от 0 до 255.

При создании таблицы нельзя смешивать столбцы типа CHAR и VARCHAR. Если такое происходит, СУБД MySQL изменит тип столбцов согласно правилу: в случае, когда в таблице присутствует хоть один столбец переменной длины, все столбцы типа CHAR приводятся к типу VARCHAR.

Типы BLOB и TEXT предназначены для хранения строк переменной длины и во всем аналогичны друг другу, отличаясь только в деталях. Например, при выполнении операций над столбцами типа TEXT учитывается кодировка, а типа BLOB — нет.

Тип TEXT обычно используется для хранения больших объемов текста, в то время как BLOB — для больших двоичных объектов, таких как электронные документы, изображения, музыкальные файлы и т. п.

К особым типам данных относятся ENUM и SET. Строки этих типов принимают значения из заранее заданного списка допустимых значений. Основное различие между ними заключается в том, что значение типа ENUM должно содержать точно одно значение из указанного множества, тогда как столбцы SET могут содержать любой (или все) элементы заранее заданного множества одновременно. Так, значения для столбца, объявленного как ENUM('y','n'), могут принимать только два значения: либо 'y', либо 'n'.

Для типа SET, так же как и для типа ENUM, при объявлении задается список возможных значений, но ячейка может принимать любое значение из списка, а пустая строка означает, что ни один из элементов списка не выбран. Например, столбец SET('y','n') может принимать значения ('y','n'), ('y'), ('n') и пустое множество ().

Типы ENUM и SET можно назвать строковыми лишь отчасти, т. к. при объявлении они задаются списком строк, но во внутреннем представлении базы данных элементы множеств сохраняются в виде чисел. Элементы типа ENUM

нумеруются последовательно, начиная с 1. В зависимости от числа элементов в списке под столбец может отводиться 1 байт (до 256 элементов в списке) или 2 байта (от 257 до 65 536 элементов в списке).

Элементы множества SET обрабатываются как биты, размер типа при этом также определяется числом элементов в списке: 1 байт (от 1 до 8 элементов), 2 байта (от 9 до 16 элементов), 3 байта (от 17 до 24 элементов), 4 байта (от 25 до 32 элементов) и 8 байтов (от 33 до 64 элементов).

В листинге 2.19 представлен оператор CREATE TABLE, создающий таблицу tbl2 с тремя полями:

- text_field — текстовое поле типа TEXT;
- enum_field — поле типа ENUM, которое может принимать одно из трех значений: 'first', 'second' или 'third';
- set_field — поле типа SET, которое может принимать одну из комбинаций множества ('a','b','c','d').

Листинг 2.19. Создание таблицы tbl2 с тремя полями

```
CREATE TABLE tbl2
(
text_field TEXT DEFAULT '',
enum_field ENUM('first','second','third') DEFAULT 'first',
set_field SET('a','b','c','d') DEFAULT 'a,b,c'
)
```

Как видно из листинга 2.19, в качестве значения по умолчанию для поля text_field используется пустая строка, для поля enum_field — значение 'first', а для поля set_field — список 'a,b,c'.

Календарный тип данных

В табл. 2.3 представлены календарные типы данных, поддерживаемые СУБД MySQL. Тип DATE предназначен для хранения даты, TIME — для времени суток, а DATETIME — для представления и даты, и времени суток. Тип TIMESTAMP предназначен для представления даты и времени суток в виде числа секунд, прошедших с полуночи 1 января 1970 года. Тип данных YEAR позволяет хранить только год.

Для значений, имеющих тип DATE или DATETIME, в качестве первой цифры ожидается год либо в формате YYYY, например '2005-10-15', либо в формате YY, например '05-10-15'. После года через дефис указывается месяц в формате MM (например, 10), а затем день в формате DD (например, 15).

Таблица 2.3. *Календарные типы данных*

Тип	Объем памяти, байтов	Диапазон
DATE	3	От '1000-01-01' до '9999-12-31'
TIME	3	От '–828:59:59' до '828:59:59'
DATATIME	8	От '1000-01-01 00:00:00' до '9999-12-31 00:00:00'
TIMESTAMP[(M)]	4	От '1970-01-01 00:00:00' до '2038-12-31 59:59:59'
YEAR[(M)]	1	От 1901 до 2155 для YEAR(4), от 1970 до 2069 для YEAR(2)

Замечание

На протяжении всей книги мы будем обозначать формат даты как 'YYYY-MM-DD' для дат вида 2005-10-15 и формат 'YY-MM-DD' для дат вида 05-10-15. Для указания времени будет использоваться формат hh:mm:ss, где hh — часы, mm — минуты, а ss — секунды, например, 10:48:56.

В типах TIME и DATETIME время приводится в привычном формате 'hh:mm:ss', где hh — часы, mm — минуты, а ss — секунды. Дни, месяцы, часы, минуты и секунды можно записывать как с ведущим нулем: 01, так и без него: 1. Например, все следующие записи идентичны:

```
'2005-04-06 02:04:08'
'2005-4-06 02:04:08'
'2005-4-6 02:04:08'
'2005-4-6 2:04:08'
'2005-4-6 2:4:08'
'2005-4-6 2:4:8'
```

В качестве разделителя между годами, месяцами, днями, часами, минутами, секундами может выступать любой символ, отличный от цифры. Так, следующие значения идентичны:

```
'05-12-31 11:30:45'
'05.12.31 11+30+45'
'05/12/31 11*30*45'
'05@12@31 11^30^45'
```

Дата и время суток также могут быть представлены в форматах 'YYYYMMDDhhmmss' и 'YYMMDDhhmmss'. Например, строки '2005091528' и '050523091528' аналогичны '2005-05-23 09:15:28', однако строка '051122129015' уже не может рассматриваться как дата и время суток, т. к. значение для минут равно 90 и выходит за допустимый интервал. Вместо строк допустимы и

целочисленные значения, например, 2005091528 и 0523091528 рассматриваются как '2005-05-23 09:15:28'.

Начиная с версии MySQL 4.1.1, при указании времени суток после секунд через точку можно также указать микросекунды, т. е. использовать расширенный формат вида `'hh:mm:ss.ffffff'`, например '10:25:14.000001'. Кроме расширенного формата, можно использовать краткие форматы `'HH:MM'` и `'HH'` — вместо пропущенных величин будут подставлены нулевые значения.

Столбцы типа `YEAR` предназначены для хранения только года, указание параметра `M` позволяет задать формат года. Так, для `YEAR(2)` год представляется в виде двух цифр `YY`, например, `05` или `97`, с диапазоном данных от 1970 года до 2069, а тип `YEAR(4)` позволяет задать тип в виде четырех цифр `YYYY`, например, `2005` или `1997`, с диапазоном от 1901 до 2155 года. Если параметр `M` не указан, по умолчанию считается, что он равен 4.

Тип данных *NULL*

Реляционная база данных позволяет объединить многочисленные данные в одну таблицу и при помощи SQL-запросов проводить над ней различные манипуляции, получая результат в виде чисел и строк, а также новых таблиц. При создании таблицы неизбежны случаи, когда информации недостаточно, и для части данных нельзя определить, какое значение они примут. Такие данные обозначаются специальным типом — `NULL`.

Например, в форме регистрации пользователь обязательно должен указать фамилию, имя, отчество. Кроме того, по желанию он может указать свой e-mail и URL домашней страницы. Для регистрации посетителей создается таблица `users`, оператор `CREATE TABLE` которой представлен в листинге 2.20.

Листинг 2.20. Создание таблицы `users`

```
CREATE TABLE users
(
  id_user INT,                          # уникальный номер
  surname VARCHAR(255),                 # фамилия
  name VARCHAR(255),                    # имя
  patronymic VARCHAR(255),              # отчество
  email VARCHAR(255),                   # e-mail
  url VARCHAR(255)                      # адрес домашней страницы
)
```

Замечание

Комментарий в SQL начинается с двух дефисов `--`, все, что расположено правее, считается комментарием. Непосредственно после `--` должен следовать

пробел. Помимо стандартного комментария, MySQL содержит ряд собственных комментариев. Shell-комментарий # действует аналогично --, все что расположено правее его, является текстом комментария. С-комментарий / * / является многострочным — комментарий начинается с / и заканчивается только тогда, когда встретится завершение */.

Посетитель обязан указать свои фамилию, имя и отчество, но может не указывать e-mail и адрес домашней страницы, даже несмотря на то, что они у него имеются в наличии. Таким образом, если для полей email и url нет информации, это не значит, что ее нет в природе, просто на данный момент она неизвестна. Такие поля принимают значение NULL — отсутствие информации, т. е. неопределенное значение. Выполнение арифметических операций с данными типа NULL всегда дает NULL, передача NULL в качестве аргумента функции также всегда приводит к значению типа NULL. Любые действия, производимые над неопределенным значением, приводят опять к неопределенному значению.

Для указания того факта, что поле может принимать значение NULL, в определении столбца после типа данных следует указать ключевое слово NULL. Если поле ни при каких обстоятельствах не должно принимать значение NULL (регистрация невозможна, если фамилия пользователя неизвестна), следует указать ключевое слово NOT NULL (листинг 2.21).

Листинг 2.21. Альтернативное определение таблицы users

```
CREATE TABLE users
(
  id_user INT NOT NULL,                    # уникальный номер
  surname VARCHAR(255) NOT NULL,           # фамилия
  name VARCHAR(255) NOT NULL,              # имя
  patronymic VARCHAR(255) NOT NULL,        # отчество
  email VARCHAR(255) NULL,                 # e-mail
  url VARCHAR(255) NULL                    # адрес домашней страницы
)
```

Передача значения NULL первым четырем столбцам таблицы users приведет к возникновению ошибки. Если при создании новой записи поля не инициируются и им будут присвоены значения по умолчанию, то id_user получит значение 0, surname, name и patronymic будет присвоена пустая строка, а email и url — значение NULL.

Замечание

Атрибут NOT NULL можно не указывать, т. к. он присваивается столбцу по умолчанию, если никакой из атрибутов не указан.

Совместно с атрибутами NOT NULL и NULL можно использовать DEFAULT, который имеет больший приоритет (листинг 2.22).

Листинг 2.22. Совместное использование типов NULL и NOT NULL с DEFAULT

```
CREATE TABLE users
(
  id_user INT NOT NULL DEFAULT 1,                # уникальный номер
  surname VARCHAR(255) NOT NULL DEFAULT 'Ф',     # фамилия
  name VARCHAR(255) NOT NULL DEFAULT 'И',        # имя
  patronymic VARCHAR(255) NOT NULL DEFAULT 'О', # отчество
  email VARCHAR(255) NULL DEFAULT 'None',        # e-mail
  url VARCHAR(255) NULL                          # адрес домашней страницы
)
```

Если при создании новой записи поля не инициируются и им будут присвоены значения по умолчанию, то id_user получит значение 1, surname — 'Ф', name — 'И', patronymic — 'О', email — 'None', а url — NULL. При этом по-прежнему поля id_user, surname, name и patronymic не могут принимать значение NULL, а для email и url это допустимо.

2.3.2. Просмотр структуры таблицы

Для просмотра структуры таблицы можно использовать оператор DESCRIBE. В листинге 2.23 демонстрируется его применение для просмотра структуры таблицы users (см. листинг 2.22).

Замечание

Оператор DESCRIBE имеет синоним DESC.

Листинг 2.23. Структура таблицы users

```
DESCRIBE users;
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| id_user    | int(11)      | NO   |     | 1       |       |
| surname    | varchar(255) | NO   |     | Ф       |       |
| name       | varchar(255) | NO   |     | И       |       |
| patronymic | varchar(255) | NO   |     | О       |       |
| email      | varchar(255) | YES  |     | None    |       |
| url        | varchar(255) | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
```

Третье поле результирующей таблицы (`Null`) показывает, какие поля могут принимать значение `NULL` (`YES`), а какие нет (`NO`). В пятом поле (`Default`) приводятся значения полей по умолчанию. В шестом столбце `Extra` перечисляются дополнительные атрибуты столбца.

Оператор `DESCRIBE` позволяет выводить параметры лишь одного указанного столбца. В листинге 2.24 выводятся параметры столбца `name` таблицы `users`.

Листинг 2.24. Параметры столбца `name` таблицы `users`

```
DESCRIBE users name;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| name  | varchar(255) | NO   |     | И       |       |
+-------+--------------+------+-----+---------+-------+
```

Вместо имени столбца можно использовать `LIKE`-шаблон с применением специальных символов: _ (заменяет собой один любой символ) и `%` (заменяет собой любое количество символов). В листинге 2.25 выводится структура столбцов, имена которых заканчиваются на символ "е". Следует обратить внимание, что для использования шаблона строку необходимо заключать в одиночные кавычки, в противном случае сервер MySQL сообщит об ошибочном SQL-запросе.

Листинг 2.25. Использование специальных символов

```
DESCRIBE users '%e';
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| surname | varchar(255) | NO   |     | Ф       |       |
| name    | varchar(255) | NO   |     | И       |       |
+---------+--------------+------+-----+---------+-------+
```

Альтернативным способом просмотра структуры таблицы является оператор `SHOW COLUMNS` (листинг 2.26), который имеет следующий синтаксис:

```
SHOW [FULL] COLUMNS FROM tbl [FROM dbase] [LIKE 'pattern']
```

Оператор выводит список столбцов и их атрибутов для таблицы *`tbl`*. При помощи дополнительного ключевого слова `FROM` можно указать базу данных, в которой находится таблица. Предложения, заключенные в квадратные скобки, не являются обязательными и могут быть опущены.

Замечание

Оператор `SHOW COLUMNS` имеет синоним `SHOW FIELDS`.

Листинг 2.26. Использование оператора `SHOW COLUMNS`

```
SHOW COLUMNS FROM users;
```

Результирующая таблица оператора `SHOW COLUMNS` в листинге 2.26 полностью эквивалентна результирующей таблице оператора `DESCRIBE` (см. листинг 2.23).

Вместо дополнительного ключевого слова `FROM` для указания базы данных допускается использование расширенного имени таблицы. В листинге 2.27 представлены запросы, выводящие структуру таблицы `users` из базы данных `test`. Оба запроса являются эквивалентными.

Листинг 2.27. Использование ключевого слова `FROM`

```
SHOW COLUMNS FROM users FROM test;
SHOW COLUMNS FROM test.users;
```

Ключевое слово `FULL` позволяет вывести дополнительную информацию, связанную с привилегиями и комментариями к столбцам (листинг 2.28). Помимо представленных выше столбцов, при использовании ключевого слова `FULL` в результирующей таблице появляется еще несколько столбцов:

- `Collation` — для текстовых столбцов выводится используемая в данном столбце сортировка, для столбцов нетекстового типа — `NULL`;
- `Privileges` — привилегии, выделенные для столбца;
- `Comment` — комментарий для столбца, если он был задан при определении таблицы оператором `CREATE TABLE`.

Замечание

Очень широкие таблицы, содержащие большое число столбцов, удобнее выводить в вертикальном представлении *(см. разд. 1.8.4)*.

Листинг 2.28. Использование ключевого слова `FULL`

```
SHOW FULL COLUMNS FROM users LIKE 'id_user'\G
*************************** 1. row ***************************
Field: id_user
Type: int(11)
Collation: NULL
```

```
Null: NO
Key:
Default: 1
Extra:
Privileges: select,insert,update,references
Comment:
```

Необязательное ключевое слово LIKE позволяет вывести информацию только для столбцов, удовлетворяющих шаблону.

2.3.3. Параметры столбцов

Каждый столбец может быть снабжен атрибутами, которые размещают через пробел за типом данных. Допускается использование следующих атрибутов.

- NULL — означает, что столбец может содержать значение NULL (по умолчанию).
- NOT NULL — столбец не может содержать значение NULL.
- DEFAULT *default_value* — определяет значение столбца, которое он принимает по умолчанию, если ему не передано значение при вставке записи оператором INSERT. Это ключевое слово нельзя использовать совместно с типами данных BLOB и TEXT. Значения по умолчанию должны быть константами, т. е. использование встроенных функций, таких как NOW() или CURRENT_DATE(), не допустимо. Если ключевое слово DEFAULT опущено, то MySQL автоматически присваивает полю значение. В качестве значения по умолчанию в этом случае выступает значение NULL, если для столбца допустимо использование данного типа. Если использование NULL запрещено, то значение по умолчанию присваивается согласно следующим правилам:
  - для числовых типов, за исключением объявленных с атрибутом AUTO_INCREMENT, значение по умолчанию равно 0. Для столбца AUTO_INCREMENT значением по умолчанию является следующее значение в последовательности для этого столбца;
  - для типов даты и времени, отличных от TIMESTAMP, значение по умолчанию равно соответствующей нулевой величине для данного типа, например, для datetime в качестве значения по умолчанию будет назначено '0000-00-00 00:00:00'. Для столбца TIMESTAMP значение по умолчанию представляет собой текущее значение даты и времени;
  - для строковых типов, кроме ENUM, значением по умолчанию является пустая строка. Для ENUM значение по умолчанию равно первой перечисляемой величине.

- `AUTO_INCREMENT` — при вставке в таблицу новой записи передача в поле, снабженное данным атрибутом, величины 0 или `NULL` приводит к назначению ему величины, равной максимальному значению столбца плюс единица. Таблица может содержать только один столбец, снабженный атрибутом `AUTO_INCREMENT`, а значения первичного ключа должны быть уникальными, т. е. повторения в пределах одной таблицы не допускаются. Также для этого столбца не допускается использование ключевого слова `DEFAULT`.
- `KEY` — или `INDEX`, помечает столбец как обычный индекс.
- `PRIMARY KEY` — наличие данного индекса позволяет пометить столбец как первичный ключ.
- `UNIQUE [KEY]` — помечает столбец как уникальный индекс, в котором недопустимы повторяющиеся значения.
- `FOREIGN KEY` — внешний ключ применим только для таблиц типа InnoDB, для всех других таблиц игнорируется.
- `COMMENT` — комментарий к столбцу, который отображается операторами `SHOW CREATE TABLE` и `SHOW FULL COLUMNS`.
- `CHARACTER SET` — определяет кодировку текстового столбца.
- `COLLATE` — определяет правила сопоставления текстовых столбцов (сортировку).

В листинге 2.29 представлен оператор `CREATE TABLE` для таблицы `test`, который демонстрирует применение конструкций `COMMENT` и `CHARACTER SET`.

Листинг 2.29. Оператор `CREATE TABLE` для создания таблицы `test`

```
CREATE TABLE test (
dos_name TEXT CHARACTER SET cp866 COMMENT 'Кодировка DOS',
win_name TEXT CHARACTER SET cp1251 COMMENT 'Кодировка Windows',
koi8r_name TEXT CHARACTER SET koi8r COMMENT 'Кодировка KOI8-R',
utf8_name TEXT CHARACTER SET utf8 COMMENT 'Unicode'
);
```

Как видно из листинга 2.29, таблица `test` состоит из четырех полей типа `TEXT`:

- `win_name` — текст в кодировке cp866 (русская кодировка DOS);
- `win_name` — текст в кодировке cp1251 (русская кодировка Windows);
- `koi8r_name` — текст в кодировке KOI8-R (кодировка, распространенная в Интернете);
- `utf8_name` — текст в кодировке UTF-8 (Unicode).

В полях таблицы `test` теперь можно хранить текст в соответствующих кодировках, и он будет сортироваться правильно. Если же в поле с кодировкой cp866 поместить текст в кодировке cp1251, то о правильной сортировке записей при помощи конструкции `ORDER BY` не может быть и речи.

Замечание

При указании кодировки столбец не должен иметь атрибут `NOT NULL`, иначе это вызовет синтаксическую ошибку.

При помощи оператора `SHOW CHARACTER SET` можно посмотреть, какие кодировки поддерживает сервер MySQL (листинг 2.30). Для различных версий их число может быть различным, например, в тестовые версии MySQL обычно не включают национальные кодировки и поддерживаются только кодировки latin.

Листинг 2.30. Результат работы оператора `SHOW CHARACTER SET`

```
SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | ISO 8859-1 West European    | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |      1 |
| ascii    | US ASCII                    | ascii_general_ci    |      1 |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |      3 |
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |      2 |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |      1 |
| tis620   | TIS620 Thai                 | tis620_thai_ci      |      1 |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |      2 |
| koi8u    | KOI8-U Ukrainian            | koi8u_general_ci    |      1 |
| gb2312   | GB2312 Simplified Chinese   | gb2312_chinese_ci   |      2 |
| greek    | ISO 8859-7 Greek            | greek_general_ci    |      1 |
| cp1250   | Windows Central European    | cp1250_general_ci   |      1 |
| gbk      | GBK Simplified Chinese      | gbk_chinese_ci      |      2 |
| latin5   | ISO 8859-9 Turkish          | latin5_turkish_ci   |      1 |
| armscii8 | ARMSCII-8 Armenian          | armscii8_general_ci |      1 |
| utf8     | UTF-8 Unicode               | utf8_general_ci     |      3 |
| ucs2     | UCS-2 Unicode               | ucs2_general_ci     |      2 |
```

```
| cp866    | DOS Russian                  | cp866_general_ci    |      1 |
| keybcs2  | DOS Kamenicky Czech-Slovak   | keybcs2_general_ci  |      1 |
| macce    | Mac Central European         | macce_general_ci    |      1 |
| macroman | Mac West European            | macroman_general_ci |      1 |
| cp852    | DOS Central European         | cp852_general_ci    |      1 |
| latin7   | ISO 8859-13 Baltic           | latin7_general_ci   |      1 |
| cp1251   | Windows Cyrillic             | cp1251_general_ci   |      1 |
| cp1256   | Windows Arabic               | cp1256_general_ci   |      1 |
| cp1257   | Windows Baltic               | cp1257_general_ci   |      1 |
| binary   | Binary pseudo charset        | binary              |      1 |
| geostd8  | GEOSTD8 Georgian             | geostd8_general_ci  |      1 |
+----------+------------------------------+---------------------+--------+
```

Как видно из листинга 2.30, первый столбец возвращаемой таблицы содержит название кодировки, используемое в SQL-запросе `CREATE TABLE`. Второй столбец содержит официальное название кодировки, третий столбец — правила сравнения, которые назначаются для данной кодировки по умолчанию. Дело в том, что сортировка столбцов требует задать порядок сравнения букв, "А = а < Б = б < В = в" и т. д. Так как код буквы в каждой кодировке различается, необходимо составить для каждой кодировки свои правила сравнения, которые, вообще говоря, можно переопределить, указав после кодировки ключевое слово `COLLATE` (листинг 2.31). Последний столбец содержит максимальное число байтов, отводимых под одну букву в кодировке.

Замечание

Сортировки обычно могут принимать два, реже три суффикса, суффикс `_bin` характерен для языков, использующих латинский алфавит, и предписывает сравнение строк как обычных бинарных данных, т. е. по коду символа. Суффикс `_ci` предписывает осуществлять сравнение без учета регистра, а суффикс `_cs` — с учетом.

Листинг 2.31. Применение ключевого слова `COLLATE`

```
CREATE TABLE test (
dos_name TEXT CHARACTER SET cp866 COLLATE cp866_general_ci,
win_name TEXT CHARACTER SET cp1251 COLLATE cp1251_general_ci,
koi8r_name TEXT CHARACTER SET koi8r COLLATE koi8r_general_ci,
utf8_name TEXT CHARACTER SET utf8 COLLATE utf8_general_ci
);
```

Однако в повседневной практике, если нет необходимости производить поиск и сравнение строк с учетом регистра, переопределение правил сравнения текстовых строк практически не требуется.

Комментарий к столбцу, который задается после ключевого слова `COMMENT`, позволяет закрепить за столбцом текст длиной не более 60 символов. Просмотреть комментарий к каждому из столбцов можно при помощи оператора `SHOW FULL COLUMNS FROM tbl` (листинг 2.32).

Листинг 2.32. Просмотр комментария таблицы

```
SHOW FULL COLUMNS FROM test;
+------------+------+------+---------+--------+------------------+
| Field      | Type | Null | Default | Privileges | Comment       |
+------------+------+------+---------+--------+------------------+
| dos_name   | text | YES  | NULL    | select | Кодировка DOS     |
| win_name   | text | YES  | NULL    | select | Кодировка Windows |
| koi8r_name | text | YES  | NULL    | select | Кодировка KOI8-R  |
| utf8_name  | text | YES  | NULL    | select | Unicode           |
+------------+------+------+---------+--------+------------------+
```

Для удобочитаемости из конечного вывода оператора `SHOW FULL COLUMNS` исключены столбцы `Collation`, `Key` и `Extra`.

2.3.4. Допустимые имена баз данных, таблиц и столбцов

Имена баз данных, таблиц и столбцов могут иметь до 64 символов, включая любые символы, кроме `.`, `/` и `\`. Если имя совпадает с ключевым словом или содержит символы, отличные от цифр и букв (одни цифры не допускаются), его необходимо заключать в обратные кавычки: `` `time` `` (листинг 2.33).

Замечание

Использовать обратные кавычки можно не только совместно с идентификаторами, которые совпадают с ключевыми словами, не будет ошибки, если каждое имя базы данных, таблицы или столбца будет обрамлено обратными кавычками.

Листинг 2.33. Использование обратных кавычек

```
CREATE TABLE tbl
id INT,
`time` DATETIME
);
```

В противном случае, если не использовать обратные кавычки, анализатор СУБД не сможет отделить имя базы данных, таблицы или столбца от ключевого слова.

В табл. 2.4 представлен список зарезервированных слов — любое имя, совпадающее с одним из таблицы, должно заключаться в обратные кавычки.

Таблица 2.4. *Зарезервированные слова СУБД MySQL*

ACCESSIBLE	CREATE	ELSEIF
ADD	CROSS	ENCLOSED
ALL	CURRENT_DATE	ESCAPED
ALTER	CURRENT_TIME	EXISTS
ANALYZE	CURRENT_TIMESTAMP	EXIT
AND	CURRENT_USER	EXPLAIN
AS	CURSOR	FALSE
ASC	DATABASE	FETCH
ASENSITIVE	DATABASES	FLOAT
BEFORE	DAY_HOUR	FLOAT4
BETWEEN	DAY_MICROSECOND	FLOAT8
BIGINT	DAY_MINUTE	FOR
BINARY	DAY_SECOND	FORCE
BLOB	DEC	FOREIGN
BOTH	DECIMAL	FROM
BY	DECLARE	FULLTEXT
CALL	DEFAULT	GRANT
CASCADE	DELAYED	GROUP
CASE	DELETE	HAVING
CHANGE	DESC	HIGH_PRIORITY
CHAR	DESCRIBE	HOUR_MICROSECOND
CHARACTER	DETERMINISTIC	HOUR_MINUTE
CHECK	DISTINCT	HOUR_SECOND
COLLATE	DISTINCTROW	IF
COLUMN	DIV	IGNORE
CONDITION	DOUBLE	IN
CONNECTION	DROP	INDEX
CONSTRAINT	DUAL	INFILE
CONTINUE	EACH	INNER
CONVERT	ELSE	INOUT

Таблица 2.4 *(продолжение)*

INSENSITIVE	LOW_PRIORITY	READ_WRITE
INSERT	MATCH	REAL
INT	MEDIUMBLOB	REFERENCES
INT1	MEDIUMINT	REGEXP
INT2	MEDIUMTEXT	RELEASE
INT3	MIDDLEINT	RENAME
INT4	MINUTE_MICROSECOND	REPEAT
INT8	MINUTE_SECOND	REPLACE
INTEGER	MOD	REQUIRE
INTERVAL	MODIFIES	RESTRICT
INTO	NATURAL	RETURN
IS	NOT	REVOKE
ITERATE	NO_WRITE_TO_BINLOG	RIGHT
JOIN	NULL	RLIKE
KEY	NUMERIC	SCHEMA
KEYS	ON	SCHEMAS
KILL	OPTIMIZE	SECOND_MICROSECOND
LEADING	OPTION	SELECT
LEAVE	OPTIONALLY	SENSITIVE
LEFT	OR	SEPARATOR
LIKE	ORDER	SET
LIMIT	OUT	SHOW
LINEAR	OUTER	SMALLINT
LINES	OUTFILE	SPATIAL
LOAD	PRECISION	SPECIFIC
LOCALTIME	PRIMARY	SQL
LOCALTIMESTAMP	PROCEDURE	SQLEXCEPTION
LOCK	PURGE	SQLSTATE
LONG	RANGE	SQLWARNING
LONGBLOB	READ	SQL_BIG_RESULT
LONGTEXT	READS	SQL_CALC_FOUND_ROWS
LOOP	READ_ONLY	SQL_SMALL_RESULT

Таблица 2.4 *(окончание)*

SSL	UNION	VARCHAR
STARTING	UNIQUE	VARCHARACTER
STRAIGHT_JOIN	UNLOCK	VARYING
TABLE	UNSIGNED	WHEN
TERMINATED	UPDATE	WHERE
THEN	UPGRADE	WHILE
TINYBLOB	USAGE	WITH
TINYINT	USE	WRITE
TINYTEXT	USING	X509
TO	UTC_DATE	XOR
TRAILING	UTC_TIME	YEAR_MONTH
TRIGGER	UTC_TIMESTAMP	ZEROFILL
TRUE	VALUES	
UNDO	VARBINARY	

2.3.5. Параметры таблицы

Параметры таблицы, обозначенные в схеме оператора `CREATE TABLE` как *`table_options`*, были введены в MySQL, начиная с версии 3.23. Далее приводятся наиболее распространенные параметры, которые могут быть использованы при создании таблицы.

ENGINE (TYPE)

Данное ключевое слово определяет тип таблицы. Использование ключевого слова `ENGINE` является более предпочтительным, но в отличие от `TYPE` не может быть использовано в версиях MySQL младше, чем 4.0.18. Использование ключевого слова `ENGINE` демонстрируется в листинге 2.34.

Листинг 2.34. Использование ключевого слова `ENGINE`

```
CREATE TABLE tbl_myisam (i INT) ENGINE = MyISAM;
CREATE TABLE tbl_innodb (i INT) TYPE = InnoDB;
```

Параметр `ENGINE` может принимать значения, приведенные в табл. 2.5.

Таблица 2.5. *Типы таблиц*

Имя таблицы	Описание
ARCHIVE	Хранения большого объема данных в сжатом формате
BDB	Транзакционные таблицы с блокировкой на уровне страниц
BerkeleyDB	Синоним для BDB
CSV	Формат CSV, записи в котором хранятся в строках, а поля разделены точкой с запятой
EXAMPLE	Данный тип таблиц является заглушкой, демонстрирующей, каким образом создаются новые типы таблиц
FEDERATED	Позволяет хранить данные в удаленных таблицах, расположенных на другой машине сети
HEAP	Синоним для MEMORY
InnoDB	Транзакционные таблицы с блокировкой на уровне записей и поддержкой внешних ключей и каскадного удаления
MEMORY	Высокоскоростная таблица, данные в которой располагаются в оперативной памяти
MERGE	Объединение нескольких MyISAM-таблиц, выступающих как одна таблица
MRG_MyISAM	Синоним для MERGE
MyISAM	Переносимая кросс-платформенная таблица, поддерживающая полнотекстовый поиск
NDB	Таблица для кластеров
NDBCLUSTER	Синоним NDB

AUTO_INCREMENT

Параметр AUTO_INCREMENT позволяет задать число, начиная с которого будет происходить увеличение счетчика для столбца, снабженного этим атрибутом (листинг 2.35).

Листинг 2.35. Использование параметра AUTO_INCREMENT

```
CREATE TABLE name (
id_name INT(6) NOT NULL AUTO_INCREMENT,
name TINYTEXT NOT NULL,
) ENGINE = MyISAM AUTO_INCREMENT = 144;
```

При вставке новой записи поле id_name получит значение 145, а не 1. Такой прием удобно использовать для пропуска некоторого количества значений, а также для воссоздания базы данных из дампа.

[DEFAULT] CHARACTER SET

Данный параметр позволяет задать кодировку для всех текстовых полей таблицы и может принимать две формы: краткую `CHARACTER SET` и полную `DEFAULT CHARACTER SET`. Пример использования данного параметра приведен в листинге 2.36.

Листинг 2.36. Использование параметра `CHARACTER SET`

```
CREATE TABLE collection (
coll_koi8r TINYTEXT CHARACTER SET koi8r,
name TINYTEXT,
description TINYTEXT
) ENGINE = MyISAM CHARACTER SET cp1251;
```

В таблице `collection` из листинга 2.36 столбец `coll_koi8r` будет сортировать текст в предположении, что он создан при помощи кодировки KOI8-R, значения остальных столбцов (`name` и `description`) будут сортироваться в предположении, что они созданы в кодировке Windows-1251.

Параметр `CHARACTER SET` может быть задан в нескольких формах, которые эквивалентны между собой:

```
CHARACTER SET charset
CHARSET = charset
CHARSET charset
```

Здесь *`charset`* является названием кодировки, например, cp1251, koi8r и т. п. Все три формы можно также применять для определения кодировки столбца.

CHECKSUM

Параметр `CHECKSUM` является флагом, который принимает значение 1 или 0. Если устанавливается значение 1, MySQL рассчитывает контрольную сумму для каждой записи таблицы (листинг 2.37). Это приводит к небольшому снижению скорости обновления таблицы, но значительно повышает вероятность ее восстановления в случае сбоя.

Листинг 2.37. Использование параметра `CHECKSUM`

```
CREATE TABLE test (
name TINYTEXT
) ENGINE = MyISAM CHECKSUM = 1;
```

COMMENT

Данный параметр позволяет задать комментарий к таблице, максимальная длина которого не должна превышать 60 символов (листинг 2.38).

Листинг 2.38. Использование параметра COMMENT

```
CREATE TABLE test (
name TINYTEXT
) ENGINE = MyISAM COMMENT = 'Тестовая таблица';
```

Этот комментарий отображается операторами SHOW CREATE TABLE и SHOW TABLE STATUS.

DATA DIRECTORY

Параметр DATA DIRECTORY применяется только для таблиц типа MyISAM и определяет каталог, куда помещается файл с данными (myd). Путь должен быть абсолютным (листинг 2.39).

Листинг 2.39. Использование параметра DATA DIRECTORY

```
CREATE TABLE test (
name TINYTEXT
) ENGINE = MyISAM DATA DIRECTORY = 'C:/mysql/data/test';
```

Данный параметр позволяет разнести структуру таблицы (frm), индексы (myi) и данные (myd) по разным разделам и дискам, что позволяет увеличить надежность и производительность базы данных.

INDEX DIRECTORY

Параметр INDEX DIRECTORY используется только для таблиц типа MyISAM и определяет каталог, куда помещается индексный файл (myi). Путь должен быть абсолютным (листинг 2.40).

Листинг 2.40. Использование параметра INDEX DIRECTORY

```
CREATE TABLE test (
name TINYTEXT
) ENGINE = MyISAM INDEX DIRECTORY = 'C:/mysql/data/test';
```

Данный параметр позволяет разнести структуру таблицы (frm), индексы (myi) и данные (myd) по разным разделам и дискам, что увеличивает надежность и производительность базы данных.

PACK_KEYS

Параметр PACK_KEYS принимает значение либо 1, либо 0. Установка значения 1 для данного параметра включает механизм сжатия текстовых (CHAR и VARCHAR) и числовых индексов. Это приводит к замедлению операции вставки, но увеличивает скорость извлечения данных.

ROW_FORMAT

Параметр ROW_FORMAT применим только к таблицам типа MyISAM и определяет способ хранения записей (листинг 2.41). Параметр может принимать следующие значения:

- FIXED — в таблицах, не использующих типы данных BLOB и TEXT, все столбцы, имеющие тип VARCHAR, рассматриваются как CHAR;
- DYNAMIC — в таблицах, не использующих типы BLOB и TEXT, все столбцы, имеющие тип CHAR, рассматриваются как VARCHAR;
- DEFAULT — таблица хранится в неупакованном виде;
- COMPRESSED — таблица хранится в упакованном виде.

Листинг 2.41. Использование параметра ROW_FORMAT

```
CREATE TABLE test (
name VARCHAR(100)
) ENGINE = MyISAM ROW_FORMAT = FIXED;
```

2.3.6. Временные таблицы

Создание временных таблиц осуществляется при помощи необязательного ключевого слова TEMPORARY в операторе CREATE TEMPORARY TABLE (листинг 2.42).

Временная таблица автоматически удаляется по завершении соединения с сервером, а ее имя действительно только в течение данного соединения. Это означает, что два разных клиента могут использовать временные таблицы с одинаковыми именами без конфликта друг с другом или с существующей таблицей с тем же именем (существующая таблица с тем же именем остается скрытой, пока не будет удалена временная таблица).

Листинг 2.42. Создание временной таблицы temp

```
CREATE TEMPORARY TABLE temp (id_catalog INT, name TINYINT);
SHOW TABLES;
+----------------+
| Tables_in_shop |
```

```
+----------------+
| users          |
+----------------+
DESCRIBE temp;
+------------+------------+------+-----+---------+-------+
| Field      | Type       | Null | Key | Default | Extra |
+------------+------------+------+-----+---------+-------+
| id_catalog | int(11)    | YES  |     | NULL    |       |
| name       | tinyint(4) | YES  |     | NULL    |       |
+------------+------------+------+-----+---------+-------+
```

Системная переменная BIG_TABLES позволяет сообщить СУБД MySQL, где должны храниться временные таблицы: в оперативной памяти или на диске. Если переменная BIG_TABLES устанавливается в 1, все временные таблицы сохраняются на диске, если в 0, то в оперативной памяти. Установить новое значение переменной можно при помощи оператора SET (листинг 2.43).

Листинг 2.43. Управление системной переменной BIG_TABLES

```
SELECT @@BIG_TABLES;
+--------------+
| @@BIG_TABLES |
+--------------+
|            0 |
+--------------+
SET BIG_TABLES = 1;
```

Для того чтобы извлечь текущую системную переменную, ее имя следует предварить двумя символами @. Более подробно системные переменные и синтаксис оператора SET обсуждается в *главе 9*.

Замечание

Начиная с версии MySQL 4.0.0, ручное управление этой переменной, скорее всего, не потребуется, т. к. СУБД MySQL сама принимает решение о том, где размещать временную таблицу.

2.3.7. Создание копии таблицы

Часто требуется создать копию таблицы, не прибегая к созданию SQL-дампа всей базы данных. Для решения этой задачи можно воспользоваться оператором SHOW CREATE TABLE (листинг 2.44), который выводит оператор CREATE TABLE для таблицы *temp* и имеет следующий синтаксис:

```
SHOW CREATE TABLE temp
```

Листинг 2.44. Использование оператора SHOW CREATE TABLE

```
SHOW CREATE TABLE temp\G
*************************** 1. row ***************************
Table: temp
Create Table: CREATE TABLE `temp` (
`id_catalog` int(11) default NULL,
`name` tinyint(4) default NULL
) ENGINE=MyISAM DEFAULT CHARSET=cp1251
```

Здесь удобно воспользоваться последовательностью \G, заставляющей консольного клиента выводить результирующую таблицу в вертикальном формате, иначе структура таблицы нарушается. Теперь остается лишь скопировать содержимое столбца Create Table для того, чтобы получить оператор, воспроизводящий структуру копируемой таблицы.

Оператор SHOW CREATE TABLE заключает в обратные кавычки имена таблицы и столбцов. Отключить такое поведение можно, установив значение системной переменной SQL_QUOTE_SHOW_CREATE равным 0.

Получить полную копию таблицы вместе с записями удобнее всего при помощи специальной формы оператора CREATE TABLE, которая создает новую таблицу из результирующей таблицы оператора SELECT. В листинге 2.45 создается таблица tbl1 и заполняется тремя записями.

Листинг 2.45. Создание временной таблицы tbl1 при помощи вложенного запроса

```
CREATE TABLE tbl (id INT, name TINYTEXT);
INSERT INTO tbl VALUES (1, 'first'), (2, 'second'), (3, 'third');
CREATE TABLE tbl1
SELECT name
FROM tbl;
SELECT * FROM tbl1;
+--------+
| name   |
+--------+
| first  |
| second |
| third  |
+--------+
```

Следует обратить внимание на тот факт, что при таком способе создания таблицы не требуется определять структуру таблицы tbl1 — она автоматически принимает структуру результирующей таблицы оператора SELECT.

2.3.8. Размещение таблиц в оперативной памяти

Для размещения таблицы в оперативной памяти используется особый тип таблиц — MEMORY. Размещение таблицы в оперативной памяти позволяет исключить из цикла размещения/извлечения данных медленный жесткий диск, поэтому все операции протекают гораздо быстрее.

Недостатком такого способа размещения данных является полная потеря данных в случае сбоя работы сервера, поэтому в таблице данного типа хранят только временную информацию, которую можно легко восстановить заново.

Замечание

Тип HEAP является синонимом MEMORY. Последнее ключевое слово было введено в MySQL 4.1.0 и его использование является более предпочтительным по отношению к HEAP.

При создании таблицы типа MEMORY она ассоциируется с одним-единственным файлом, имеющим расширение frm, в котором определяется структура таблицы. При остановке или перезапуске сервера данный файл остается в текущей базе данных, но содержимое таблицы, которое хранится в оперативной памяти машины, теряется. То есть при каждом рестарте MySQL-сервера создавать новую таблицу MEMORY не нужно, однако потребуется ее заполнение.

Так как размещение данных происходит в оперативной памяти, MEMORY-таблицы имеют несколько ограничений.

- Индексы используются только в операциях сравнения совместно с операторами = и <=>, с другими операторами, такими как > или <, индексирование столбцов не имеет смысла.
- Так же как и в случае MERGE-таблиц, возможно использование только неуникальных индексов.
- MEMORY-таблицы используют записи фиксированной длины, поэтому в них не допустимы столбцы типов TEXT и BLOB.
- В версиях, предшествующих MySQL 4.1, не поддерживается атрибут AUTO_INCREMENT.
- В версиях, предшествующих MySQL 4.0.2, не поддерживается индексирование столбцов, содержащих NULL-значения.

Для создания таблиц данного типа параметру ENGINE в операторе CREATE TABLE необходимо присвоить значение MEMORY или HEAP (листинг 2.46).

Листинг 2.46. Создание MEMORY-таблицы

```
CREATE TABLE test (i INT) ENGINE=MEMORY;
CREATE TABLE test (i INT) ENGINE=HEAP;
```

2.3.9. Объединение нескольких таблиц в одну

Для объединения нескольких таблиц типа MyISAM используется специальный тип таблиц — MERGE. Данный прием применяется для снятия ограничения на объем для таблиц MyISAM, например, если операционная или файловая система не позволяет создавать таблицы объемом больше 4 Гбайт. Таблицы MyISAM, которые подвергаются объединению в одну таблицу MERGE, должны иметь одинаковую структуру, т. е. одинаковые столбцы и индексы, а также порядок их следования.

При создании таблицы типа MERGE будут образованы файлы структуры таблицы с расширением frm и файлы с расширением mrg. Файл mrg содержит список индексных файлов (файлы myi), работа с которыми должна осуществляться как с единым файлом. До версии MySQL 4.1.1 все используемые таблицы должны были размещаться в той же базе данных, что и таблица MERGE.

К полученной объединенной таблице можно применять команды `SELECT`, `DELETE` и `UPDATE`. Если же попытаться применить к таблице MERGE команду `DROP TABLE`, она уничтожит MERGE-таблицу, не затронув при этом MyISAM-таблицы, входящие в ее состав.

В листинге 2.47 приведен пример создания MERGE-таблицы `total` из двух MyISAM-таблиц: `tbl1` и `tbl2`. Каждая из этих таблиц заполняется при помощи многострочного оператора `INSERT` тремя значениями. Так как первичный ключ таблиц снабжен атрибутом `AUTO_INCREMENT`, он автоматически получает уникальное значение в пределах таблицы.

Листинг 2.47. Создание MERGE-таблицы

```
CREATE TABLE tbl1 (
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
name CHAR(20));
CREATE TABLE tbl2 (
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
name CHAR(20));
INSERT INTO tbl1 (name) VALUES ('12'), ('34'), ('56');
INSERT INTO tbl2 (name) VALUES ('78'), ('9a'), ('bc');
CREATE TABLE total (
id INT NOT NULL AUTO_INCREMENT,
name CHAR(20), INDEX(id))
ENGINE=MERGE UNION=(tbl1,tbl2) INSERT_METHOD=LAST;
```

Результат выборки из таблицы `total` может выглядеть так, как это представлено в листинге 2.48.

Листинг 2.48. Выборка из таблицы `total`

```
SELECT * FROM total;
+----+------+
| id | name |
+----+------+
|  1 | 12   |
|  2 | 34   |
|  3 | 56   |
|  1 | 78   |
|  2 | 9a   |
|  3 | bc   |
+----+------+
6 rows in set (0.00 sec)
```

Важно отметить, что в объединенной таблице `total` поле `id` уже не является уникальным.

Замечание

Таблицы MERGE не могут содержать уникальных ключей.

Замечание

В параметрах `ENGINE` и `TYPE` в качестве синонима `MERGE` можно применять `MRG_MYISAM`.

Как видно из листинга 2.47, помимо указания типа таблицы при помощи ключевого слова `TYPE`, в конце оператора `CREATE TABLE` ключевое слово `UNION` задает имена таблиц, которые объединяются в конечную MERGE-таблицу. Кроме этого, ключевое слово `INSERT_METHOD` позволяет указывать метод добавления новых записей в MyISAM-таблицы. В качестве значения данного поля передается одна из трех констант:

- `FIRST` — при вставке новой записи в MERGE-таблицу запись размещается в первой таблице из списка, приведенного после ключевого слова `UNION`;
- `LAST` — при вставке новой записи в MERGE-таблицу запись размещается в последней таблице из списка, приведенного после ключевого слова `UNION`;
- `NO` — данное значение аналогично ситуации, когда ключевое слово `INSERT_METHOD` опускается из определения таблицы (`CREATE TABLE`). Использование оператора `INSERT` в такой таблице приведет к ошибке.

Так, в листинге 2.49 демонстрируется вставка нового значения в таблицу `total`, в определении которой опущен параметр `INSERT_METHOD`.

Листинг 2.49. Ошибочное добавление новой записи в таблицу MERGE

```
INSERT INTO total VALUES (NULL, 'xx');
ERROR 1031: Table handler for 'total' doesn't have this option
```

2.3.10. Размещение таблиц на удаленной машине

Для того чтобы обращаться к таблицам, расположенным на удаленном хосте, прибегают к типу таблиц FEDERATED. Во время создания таблицы в локальном каталоге данных mysql\data создается только файл определения структуры таблицы с расширением frm, никакие другие файлы не создаются, т. к. все данные хранятся на удаленной машине.

Замечание

Тип таблиц FEDERATED был введен в MySQL, начиная с версии 5.0.3.

Замечание

MySQL в бинарных кодах распространяется без поддержки данного типа таблиц, для его включения необходима перекомпиляция MySQL с включенной опцией `--with-federated-storage-engine`.

При работе с обычными локальными таблицами MySQL читает структуру таблицы из файла с расширением frm и обращается к файлам данных MYD для разбора данных и выполнения операций, которые необходимы для реализации SQL-запроса. При работе с FEDERATED-таблицами синтаксис запроса проверяется на локальной машине, после чего он отправляется на удаленную машину, где и выполняется. Далее результаты отправляются обратно на локальную машину — сервер MySQL, — который возвращает их клиенту.

Для создания таблицы типа FEDERATED необходимо сначала создать таблицу на удаленной машине, например, в базе данных `federated` (листинг 2.50).

Листинг 2.50. Создание MyISAM-таблицы на удаленной машине

```
CREATE TABLE test_table (
    id     INT(20) NOT NULL auto_increment,
    name   VARCHAR(32) NOT NULL default '',
other  INT(20) NOT NULL default '0',
PRIMARY KEY (id),
KEY name (name),
KEY other_key (other)
) ENGINE = MyISAM;
```

В качестве типа таблицы (ENGINE) может выступать любой тип, не обязательно MyISAM. После создания таблицы на удаленном сервере можно приступать к развертыванию FEDERATED-таблицы на локальной машине (листинг 2.51).

Листинг 2.51. Создание FEDERATED-таблицы

```
CREATE TABLE federated_table (
    id     INT(20) NOT NULL auto_increment,
    name   VARCHAR(32) NOT NULL default '',
other  INT(20) NOT NULL default '0',
PRIMARY KEY (id),
KEY name (name),
KEY other_key (other)
) ENGINE = FEDERATED
CONNECTION = 'mysql://user_name@remote_host:3306/federated/test_table';
```

Замечание

До версии 5.0.13 для указания параметров соединения с удаленным сервером вместо ключевого слова CONNECTION использовалось ключевое слово COMMENT.

Как видно из листинга 2.51, параметр ENGINE принимает значение FEDERATED, а в параметре таблицы CONNECTION указываются параметры соединения с удаленным сервером. В результате выполнения SQL-запроса из листинга 2.51 в каталоге federated локальной машины будет создан один-единственный файл federated_table.frm.

Замечание

Структура таблиц на обоих серверах, как локальном, так и удаленном, должна быть идентична.

Общий вид строки соединения при создании FEDERATED-таблиц выглядит следующим образом:

scheme://*user_name*[:*password*]@*host_name*[:*port_num*]:/*db_name*/*tbl_name*

Протокол соединения *scheme* может принимать только одно значение — mysql; параметр *user_name* определяет пользователя, от имени которого на удаленном сервере запускается запрос. Через необязательный параметр *password* передается пароль пользователя. Параметр *host_name* принимает имя или IP-адрес удаленного хоста, а необязательный параметр *port_num* — порт, который прослушивается сервером MySQL на *host_name*. Параметр *db_name* определяет имя базы данных на удаленном хосте, а *tbl_name*, соответственно, — имя таблицы.

Далее приведено несколько примеров того, как может выглядеть строка соединения в FEDERATED-таблице:

```
CONNECTION='mysql://username:password@hostname:port/database/tablename'
CONNECTION='mysql://username@hostname/database/tablename'
CONNECTION='mysql://username:password@hostname/database/tablename'
```

Таблицы типа FEDERATED обладают следующими ограничениями:

- удаленным сервером может быть только MySQL-сервер. Поддержку других СУБД создатели MySQL обещают ввести в следующих версиях;
- таблица на удаленном сервере обязательно должна существовать, иначе не удастся создать FEDERATED-таблицу на локальном сервере;
- создание FEDERATED-таблицы, указывающей на другую FEDERATED-таблицу, возможно, но следует внимательно следить за тем, чтобы не образовывались циклические ссылки, последствия от которых непредсказуемы;
- в таблицах данного типа не поддерживаются транзакции.

FEDERATED-таблицы позволяют работать с SQL-операторами `SELECT`, `INSERT`, `UPDATE`, `DELETE`, а также с индексированными столбцами, но операторы `ALTER TABLE` и `DROP TABLE` не поддерживаются.

2.4. Удаление таблиц

Для удаления таблиц предназначен оператор `DROP TABLE`, который имеет следующий синтаксис:

```
DROP [TEMPORARY] TABLE [IF EXISTS]
     tbl_name [, tbl_name] ...
```

Замечание

Элементы в квадратных скобках являются необязательными и могут быть опущены.

Оператор позволяет удалить одну или более таблиц *`tbl_name`*. Так, в листинге 2.52 первый оператор `DROP TABLE` удаляет таблицу `temp`, а второй оператор — сразу три таблицы: `tbl1`, `tbl2` и `tbl3`.

Замечание

Для того чтобы уничтожить базу данных, необходимо воспользоваться оператором `DROP DATABASE` или просто уничтожить подкаталог базы данных в каталоге данных (C:\mysql5\data\).

Листинг 2.52. Использование оператора `DROP TABLE`

```
DROP TABLE temp;
DROP TABLE tbl1, tbl2, tbl3;
```

При использовании ключевого слова `IF EXISTS` таблица (или таблицы) удаляется только в том случае, если она существует. Если указанной таблицы нет в базе данных, оператор завершит свою работу в штатном режиме. Без использования ключевого слова `IF EXISTS` попытка удаления несуществующей таблицы приводит к ошибке (листинг 2.53).

Листинг 2.53. Использование ключевого слова `IF EXISTS` совместно с `DROP TABLE`

```
DROP TABLE catalogs;
ERROR 1051 (42S02): Unknown table 'catalogs'
DROP TABLE IF EXISTS catalogs;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

Необязательное ключевое слово `TEMPORARY` позволяет сообщить, что удалению подвергаются только временные таблицы. Даже если в списке таблиц будет случайно указана постоянная таблица, оператор `DROP TABLE` ее не затронет.

Замечание

Временные таблицы более подробно рассматриваются в *разд. 2.3.6*.

2.5. Редактирование таблицы

После того, как таблицы базы данных созданы, заполнены и введены в эксплуатацию, может встать вопрос об изменении их структуры с целью оптимизации работы базы данных и использующих ее приложений. Наиболее распространенные задачи изменения структуры таблицы включают изменение порядка следования столбцов, их названий, типов, добавление новых индексов и т. д. Все эти операции производятся при помощи SQL-оператора `ALTER TABLE`, который имеет следующий синтаксис:

```
ALTER TABLE tbl alter_specification [,alter_specification] ...
```

Сразу после оператора `ALTER TABLE` следует имя таблицы *`tbl`*, которая подвергается изменению, и спецификация *`alter_specification`*, которая определяет производимое изменение. В одном операторе `ALTER TABLE` допускается указывать сразу несколько спецификаций через запятую.

Замечание

Оператор `ALTER TABLE` во время работы создает временную копию исходной таблицы. Требуемое изменение выполняется на копии, затем исходная таблица удаляется, а новая таблица переименовывается. Это делается для того, чтобы в новую таблицу автоматически попадали все обновления, кроме неудавшихся. Во время выполнения оператора `ALTER TABLE` исходная таблица доступна для чтения другим клиентам.

2.5.1. Добавление столбца

Для добавления столбца указывается спецификатор оператора `ALTER TABLE` — `ADD [COLUMN]`. В листинге 2.54 демонстрируется добавление столбца `rebate` в конец таблицы.

Замечание

Ключевое слово `COLUMN` является необязательным и может быть опущено при добавлении нового столбца в таблицу, т. е. вместо конструкции `ADD COLUMN` допускается использование лишь одного ключевого слова `ADD`. Здесь и далее элементы, заключенные в квадратные скобки, являются необязательными.

Листинг 2.54. Добавление нового столбца в таблицу

```
CREATE TABLE tbl (
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
name CHAR(20));
ALTER TABLE tbl ADD COLUMN rebate INT;
DESCRIBE tbl;
+--------+----------+------+-----+---------+----------------+
| Field  | Type     | Null | Key | Default | Extra          |
+--------+----------+------+-----+---------+----------------+
| id     | int(11)  | NO   | PRI | NULL    | auto_increment |
| name   | char(20) | YES  |     | NULL    |                |
| rebate | int(11)  | YES  |     | NULL    |                |
+--------+----------+------+-----+---------+----------------+
```

Как видно из листинга 2.54, новый столбец `rebate` был добавлен в конец таблицы. Для того чтобы изменить позицию, в которую будет помещен столбец, совместно со спецификатором `ADD [COLUMN]` используются ключевые слова `FIRST` и `AFTER`. Ключевое слово `FIRST` требует, чтобы новый столбец был размещен первым, а `AFTER` позволяет указать, после какого столбца следует поместить новый. В листинге 2.55 демонстрируется добавление столбца `id_first` в начало таблицы и столбца `rebate` в конец таблицы.

Листинг 5.55. Добавление сразу двух столбцов

```
CREATE TABLE tbl (
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
name CHAR(20));
ALTER TABLE tbl ADD id_first INT FIRST,
ADD rebate INT AFTER name;
DESCRIBE tbl;
```

```
+----------+----------+------+-----+---------+----------------+
| Field    | Type     | Null | Key | Default | Extra          |
+----------+----------+------+-----+---------+----------------+
| id_first | int(11)  | YES  |     | NULL    |                |
| id       | int(11)  | NO   | PRI | NULL    | auto_increment |
| name     | char(20) | YES  |     | NULL    |                |
| rebate   | int(11)  | YES  |     | NULL    |                |
+----------+----------+------+-----+---------+----------------+
```

2.5.2. Удаление столбца

Удаление столбца производится при помощи спецификатора `DROP [COLUMN]` оператора `ALTER TABLE`. В листинге 2.56 демонстрируется удаление столбца.

Листинг 2.56. Удаление столбца из таблицы

```
CREATE TABLE tbl (
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
name CHAR(20));
ALTER TABLE tbl DROP COLUMN name;
DESCRIBE tbl;
+-------+---------+------+-----+---------+----------------+
| Field | Type    | Null | Key | Default | Extra          |
+-------+---------+------+-----+---------+----------------+
| id    | int(11) | NO   | PRI | NULL    | auto_increment |
+-------+---------+------+-----+---------+----------------+
```

Замечание

Если таблица содержит только один столбец, то этот столбец не может быть удален. Вместо этого нужно удалить таблицу, используя команду `DROP TABLE`.

2.5.3. Изменение уже существующих столбцов

Помимо ввода в таблицу новых столбцов, часто встает задача изменения типа или названия уже существующего столбца. Для изменения типа уже существующего столбца используется спецификатор `MODIFY`, после которого указывается имя модифицируемого столбца и его новый тип. В листинге 2.57 целочисленный тип столбца `id` меняется на текстовый.

Листинг 2.57. Изменение типа столбца

```
CREATE TABLE tbl (
id INT NOT NULL,
```

```
name CHAR(20));
ALTER TABLE tbl MODIFY id TEXT;
DESCRIBE tbl;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| id    | text     | YES  |     | NULL    |       |
| name  | char(20) | YES  |     | NULL    |       |
+-------+----------+------+-----+---------+-------+
```

Точно так же, как и при создании столбца, использование ключевых слов `FIRST` и `AFTER` позволяет изменить порядок следования столбцов. В листинге 2.58 столбцы `id` и `name` меняются местами.

Листинг 2.58. Изменение порядка следования столбцов

```
ALTER TABLE tbl MODIFY id TEXT AFTER name;
DESCRIBE tbl;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| name  | char(20) | YES  |     | NULL    |       |
| id    | text     | YES  |     | NULL    |       |
+-------+----------+------+-----+---------+-------+
```

Спецификатор `MODIFY` в операторе `ALTER TABLE` не позволяет изменить имя столбца, для этого предназначена спецификация `CHANGE`. После данного ключевого слова указывается имя столбца, затем следует новое имя и новый тип. Даже если тип остается прежним, его нужно воспроизвести. В листинге 2.59 столбец `id` переименовывается в `id_tbl`.

Листинг 2.59. Переименование столбца

```
ALTER TABLE tbl CHANGE id id_tbl TEXT;
DESCRIBE tbl;
+--------+----------+------+-----+---------+-------+
| Field  | Type     | Null | Key | Default | Extra |
+--------+----------+------+-----+---------+-------+
| name   | char(20) | YES  |     | NULL    |       |
| id_tbl | text     | YES  |     | NULL    |       |
+--------+----------+------+-----+---------+-------+
```

Как и в случае `MODIFY`, совместно со спецификатором `CHANGE` допускается использование ключевых слов `FIRST` и `AFTER`, позволяющих изменить позицию

столбца. Если при помощи CHANGE изменяется только тип столбца, но не его имя, название столбца записывается в операторе ALTER TABLE два раза.

2.5.4. Переименование таблицы

Помимо изменения структуры таблицы, оператор ALTER TABLE позволяет изменять параметры таблицы, например, ее название. Для этого используется спецификатор RENAME [TO] *new_table*, в результате применения которого таблица получает новое имя *new_table*. В листинге 2.60 таблица tbl базы данных test переименовывается в table.

Замечание

Необязательное ключевое слово TO в спецификации RENAME [TO] может быть опущено.

Листинг 2.60. Переименование таблицы

```
SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| tbl            |
+----------------+
ALTER TABLE tbl RENAME TO `table`;
SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| table          |
+----------------+
```

Для переименования таблиц существует также отдельный оператор RENAME TABLE, который имеет следующий синтаксис:

RENAME TABLE *tbl_name* TO *new_tbl_name*[, *tbl_name2* TO *new_tbl_name2*,...]

Результатом работы оператора является переименование таблицы *tbl_name* в *new_tbl_name*. В одном операторе можно переименовать сразу несколько таблиц. В листинге 2.61 таблицы tbl и tmp меняются именами.

Листинг 2.61. Обмен именами таблиц

```
RENAME TABLE tbl TO backup_table,
tmp TO tbl,
backup_table TO tmp;
```

Так как в одной базе данных не может быть двух таблиц с одинаковыми именами, в листинге 2.61 таблице `tbl` потребовалось назначить временное имя `backup_table`, которое потом заменяется на `tmp`.

2.5.5. Преобразование параметров таблицы

Спецификатор `ORDER BY` *`field_name`* позволяет отсортировать записи таблицы по столбцу *`field_name`*. Следует учитывать, что созданная таблица не будет сохранять этот порядок строк после операций вставки и удаления. Так, в листинге 2.62 записи таблицы `catalogs` сортируются по полю `name`.

Листинг 2.62. Сортировка записей таблицы `catalogs` по полю `name`

```
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
ALTER TABLE catalogs ORDER BY name;
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры         |
+------------+--------------------+
```

В некоторых случаях эта возможность может облегчить операцию сортировки в MySQL, если таблица имеет такое расположение столбцов, которое требуется в дальнейшем. Спецификатор `ORDER BY` в основном полезен, если заранее известен порядок, в котором преимущественно будут запрашиваться строки.

Замечание

В большинстве случаев предпочтительнее использовать ключевое слово `ORDER BY` в операторе `SELECT`.

Помимо этого, оператор ALTER TABLE позволяет изменять значения параметров таблицы, если они не были выставлены при ее создании. Например, ключевое слово AUTO_INCREMENT позволяет назначить новое значение для параметра таблицы AUTO_INCREMENT. В листинге 2.63 для таблицы catalogs выставляется значение AUTO_INCREMENT равным 1000, в результате чего добавление нового значения приведет к тому, что первичный ключ таблицы id_catalog получит значение 1000.

Листинг 2.63. Изменение значений параметра AUTO_INCREMENT

```
ALTER TABLE catalogs AUTO_INCREMENT = 1000;
INSERT INTO catalogs () VALUES ();
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
|       1000 |                    |
+------------+--------------------+
```

Для установки кодировки по умолчанию служит другой параметр — [DEFAULT] CHARACTER SET, использование которого демонстрируется в листинге 2.64.

Листинг 2.64. Изменение значения параметра [DEFAULT] CHARACTER SET

```
ALTER TABLE catalogs DEFAULT CHARACTER SET cp1251;
ALTER TABLE products CHARACTER SET cp1251;
ALTER TABLE users CHARACTER SET cp1251
                  COLLATE cp1251_general_cs;
```

Как видно из листинга 2.64, помимо кодировки, можно указать тип сортировки (сопоставления), который задается ключевым словом COLLATE. Так в последнем SQL-запросе для таблицы users указывается сортировка для кодировки cp1251 с учетом регистра.

Однако параметр [DEFAULT] CHARACTER SET изменяет кодировку и сортировку только для самой таблицы, если нужно изменить также кодировку и сортировку для всех текстовых столбцов таблицы (CHAR, VARCHAR, TEXT), необходимо воспользоваться альтернативной спецификацией оператора ALTER TABLE, которая принимает следующую форму:

CONVERT TO CHARACTER SET *charset_name* [COLLATE *collation_name*]

Здесь *charset_name* — имя кодировки, а *collation_name* — имя сортировки. Если потребуется изменить кодировку и сортировку не только для таблиц, но и для столбцов таблицы, запросы из листинга 2.64 следует переписать так, как это представлено в листинге 2.65.

Листинг 2.65. Изменение кодировки и сортировки для всей таблицы, включая столбцы

```
ALTER TABLE catalogs CONVERT TO CHARACTER SET cp1251;
ALTER TABLE products CONVERT TO CHARACTER SET cp1251;
ALTER TABLE users CONVERT TO CHARACTER SET cp1251
                  COLLATE cp1251_general_cs;
```

Еще два спецификатора — `DISABLE KEYS` и `ENABLE KEYS` — оператора `ALTER TABLE` позволяют управлять актуальностью неуникальных индексов таблицы. Спецификатор `DISABLE KEYS` отключает обновление индексов при добавлении и удалении записей из таблицы, а спецификатор `ENABLE KEYS` восстанавливает режим обновления. Использование данных спецификаторов необходимо в том случае, когда требуется ускорить процесс вставки или удаления записей таблицы при помощи операторов `INSERT` и `DELETE` (листинг 2.66). Если отключен режим автоматического обновления индексов, СУБД MySQL не будет тратить время на сортировку индексов.

Листинг 2.66. Использование спецификаторов `DISABLE KEYS` и `ENABLE KEYS`

```
ALTER TABLE products DISABLE KEYS;
-- многочисленные операции вставки новых значений
ALTER TABLE products ENABLE KEYS;
```

2.6. Восстановление таблиц

Таблицы представляют собой файлы, доступ к которым контролирует сервер MySQL. Если файлами манипулируют в обход сервера (например, копируют из каталога данных) или сервер аварийно завершает работу (например, если он обесточивается) и не может корректно завершить операции с файлами, в этом случае таблицы могут быть повреждены.

2.6.1. Оператор *REPAIR TABLE*

Оператор `REPAIR TABLE` предназначен для восстановления поврежденных таблиц. В листинге 2.67 демонстрируется использование оператора `REPAIR TABLE` применительно к базе данных `test`, содержащей четыре таблицы: `catalogs`,

products, users и orders. Таблицы products, users и orders имеют тип MyISAM, в то время как таблица catalogs имеет тип InnoDB.

Замечание

Данный оператор может применяться только к таблицам типа MyISAM. На время выполнения оператора REPAIR TABLE доступ к таблице блокируется.

Замечание

Наряду с формой REPAIR TABLE, может использоваться форма REPAIR TABLES, которая является синонимом и производит те же действия, что и оператор REPAIR TABLE.

Листинг 2.67. Использование оператора REPAIR TABLE

```
REPAIR TABLE catalogs, products, users, orders;
+---------------+--------+----------+---------------------------------+
| Table         | Op     | Msg_type | Msg_text                        |
+---------------+--------+----------+---------------------------------+
| test.catalogs | repair | note     | The storage engine for the table
                                      doesn't support repair          |
| test.products | repair | status   | OK                              |
| test.users    | repair | status   | OK                              |
| test.orders   | repair | status   | OK                              |
+---------------+--------+----------+---------------------------------+
```

Столбец Table содержит расширенное имя таблицы, над которой была произведена операция. Расширенное имя включает в себя имя базы данных и таблицы.

Столбец Op отражает тип операции над таблицей и может принимать одно из шести значений, который в случае оператора REPAIR TABLE всегда принимает значение repair.

Третье поле Msg_type содержит информацию о типе сообщения, а четвертое Msg_text — само сообщение. Поле Msg_type может принимать одно из следующих значений:

- status — статус операции, который, как правило, равен OK. Если статус возвращает значение "Table already up to data" ("Таблица уже обновлена"), это означает, что с момента последней проверки изменения в таблице не проводились, т. е. в ее проверке не было надобности;
- error — сообщает об ошибке;
- info — информационное сообщение;

- `warning` — предупреждение;
- `note` — замечание.

Если восстановление прошло успешно или таблица не являлась поврежденной, в четвертый столбец результирующей таблицы `Msg_text` помещается значение `OK`. Сообщение "The storage engine for the table doesn't support repair" появляется в том случае, если таблица имеет тип, отличный от MyISAM.

Оператор `REPAIR TABLE` имеет следующие дополнительные параметры:

- `QUICK` — быстрое восстановление, при котором восстанавливается только дерево индекса;
- `EXTENDED` — полное восстановление, в котором оператор просматривает в таблице строку за строкой;
- `USE_FRM` — используется (листинг 2.68), если отсутствует файл индексов с расширением myi или заголовок данного файла поврежден. В этом случае СУБД MySQL создает myi-файл заново, используя информацию из файла с расширением frm.

Листинг 2.68. Использование опции `USE_FRM`

```
REPAIR TABLE catalogs, products, users, orders USE_FRM;
+---------------+--------+----------+----------------------------------+
| Table         | Op     | Msg_type | Msg_text                         |
+---------------+--------+----------+----------------------------------+
| shop.catalogs | repair | note     |The storage engine for the table
                                       doesn't support repair           |
| shop.products | repair | warning  |Number of rows changed from 0 to 31|
| shop.products | repair | status   | OK                               |
| shop.users    | repair | warning  | Number of rows changed from 0 to 6|
| shop.users    | repair | status   | OK                               |
| shop.orders   | repair | warning  | Number of rows changed from 0 to 2|
| shop.orders   | repair | status   | OK                               |
+---------------+--------+----------+----------------------------------+
```

Если сервер аварийно завершает работу во время выполнения оператора `REPAIR TABLE`, важно после его перезапуска немедленно снова запустить `REPAIR TABLE`, прежде чем выполнять какие-то манипуляции с таблицей.

2.6.2. Контрольная сумма таблицы

Если таблицы повреждены, то при запуске сервер MySQL пытается их восстановить. Поэтому один из способов восстановления таблиц заключается в обычной перезагрузке сервера MySQL. Вероятность восстановления таблиц

будет выше, если об этом позаботиться заранее. Одним из действий, увеличивающих вероятность восстановления таблиц после сбоя, является включение автоматического подсчета контрольной суммы таблицы. Для этого необходимо при помощи оператора ALTER TABLE назначить параметру CHECKSUM таблицы значение 1 (листинг 2.69).

Замечание

По умолчанию параметр CHECKSUM принимает значение 0 и подсчет контрольной суммы не производится. Кроме того, параметр поддерживается лишь одним типом таблиц — MyISAM.

Листинг 2.69. Изменение параметра CHECKSUM

```
ALTER TABLE tbl CHECKSUM = 1;
```

Включение автоматического подсчета контрольной суммы таблицы приводит к замедлению операций вставок, обновления и удаления записей, т. к. каждая из этих операций способствует пересчету контрольной суммы.

Подсчитать контрольную сумму таблицы можно при помощи оператора CHECKSUM TABLE, который был добавлен в MySQL, начиная с версии 4.1.1 (листинг 2.70).

Листинг 2.70. Подсчет контрольной суммы таблиц

```
CHECKSUM TABLE catalogs, products, users;
+---------------+------------+
| Table         | Checksum   |
+---------------+------------+
| test.catalogs | 1895537962 |
| test.products | 1706424754 |
| test.users    | 1587572300 |
+---------------+------------+
```

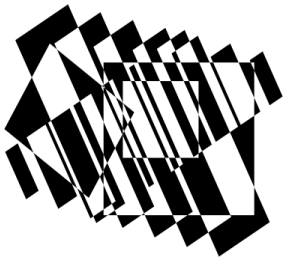
Оператор CHECKSUM TABLE имеет два взаимоисключающих параметра — QUICK и EXTENDED, которые располагаются сразу после списка таблиц. Параметр QUICK оператора CHECKSUM TABLE (листинг 2.71) позволяет извлекать контрольную сумму из счетчика, который ведется для таблицы при включенном параметре CHECKSUM (см. листинг 2.69), если параметр таблицы CHECKSUM принимает значение 0, то оператор CHECKSUM TABLE вернет NULL. Использование параметра EXNTENDED позволяет игнорировать счетчик контрольной суммы и требует от оператора CHECKSUM TABLE провести полное сканирование таблицы.

Листинг 2.71. Использование опции `QUICK`

```
CHECKSUM TABLE catalogs, products, users QUICK;
+---------------+----------+
| Table         | Checksum |
+---------------+----------+
| shop.catalogs |     NULL |
| shop.products |     NULL |
| shop.users    |     NULL |
+---------------+----------+
ALTER TABLE catalogs CHECKSUM = 1;
ALTER TABLE products CHECKSUM = 1;
ALTER TABLE users CHECKSUM = 1;
CHECKSUM TABLE catalogs, products, users QUICK;
+---------------+------------+
| Table         | Checksum   |
+---------------+------------+
| shop.catalogs | 1895537962 |
| shop.products | 1706424754 |
| shop.users    | 2410629935 |
+---------------+------------+
```

Как видно из листинга 2.71, если параметр `CHECKSUM` не установлен в единицу, вызов оператора `CHECKSUM TABLE` с опцией `QUICK` возвращает `NULL`. Если ни одна опция не указана, то MySQL возвращает контрольную сумму внутреннего счетчика для таблиц с включенной опцией `CHECKSUM` или выполняет полное сканирование таблицы в противном случае.

ГЛАВА 3



Индексы

Индексы играют большую роль в базах данных, т. к. это основной способ ускорения их работы. Обычно записи в таблице располагаются в хаотическом порядке. Для того чтобы найти нужную запись, необходимо сканировать всю таблицу, на что уходит большое количество времени. Идея индексов состоит в том, чтобы создать для столбца копию, которая постоянно будет поддерживаться в отсортированном состоянии. Это позволяет очень быстро осуществлять поиск по такому столбцу, т. к. заранее известно, где необходимо искать значение.

Обратной стороной медали является то, что добавление или удаление записи требует дополнительного времени на сортировку столбца, кроме того, создание копии увеличивает объем памяти, необходимый для размещения таблицы на жестком диске.

3.1. Индексация столбцов при создании таблицы

Существует несколько видов индексов:

- *первичный ключ* — главный индекс таблицы. В таблице может быть только один первичный ключ, и все значения такого индекса должны отличаться друг от друга, т. е. являться уникальными;
- *обычный индекс* — таких индексов может быть несколько;
- *уникальный индекс* — уникальных индексов также может быть несколько, но значения индекса не должны повторяться;
- *полнотекстовый индекс* — специальный вид индекса для столбцов типа `TEXT`, позволяющий производить полнотекстовый поиск.

Индексы могут иметь как собственные имена, так и имена индексируемых столбцов. Один индекс может охватывать один или несколько столбцов, причем тип столбца может быть любым, за исключением `ENUM` и `SET`.

3.1.1. Первичный ключ

Так как записи в реляционной таблице не упорядочены, нельзя выбрать строку по ее номеру (как в массиве), поэтому для идентификации записи в таблицу вводится так называемый *первичный ключ*. Первичный ключ является главным индексом таблицы и объявляется при помощи ключевого слова `PRIMARY KEY`, у таблицы может быть только один первичный ключ. Значение первичного ключа должно быть уникально и не повторяться в пределах таблицы. Кроме того, столбцы, помеченные атрибутом `PRIMARY KEY`, не могут принимать значение `NULL`. Для пометки поля таблицы в качестве первичного ключа достаточно поместить это ключевое слово `PRIMARY KEY` в определение столбца. В листинге 3.1 приведено определение таблицы `tbl`, где в качестве первичного ключа назначен столбец `id`.

Замечание

Первичный ключ не является обязательной конструкцией, допускается создание таблиц, не содержащих первичных ключей и индексов.

Листинг 3.1. Создание первичного ключа

```
CREATE TABLE tbl (
  id INT(11) NOT NULL PRIMARY KEY,
  name TINYTEXT NOT NULL
)
```

Просмотр структуры таблицы `tbl` при помощи оператора `DESCRIBE` показывает, что в четвертой колонке результирующей таблицы напротив поля `id` появился флаг `PRI`, который отмечает поле первичного ключа (листинг 3.2).

Листинг 3.2. Структура таблицы `tbl`

```
DESCRIBE tbl;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| id    | int(11)  | NO   | PRI |         |       |
| name  | tinytext | NO   |     |         |       |
+-------+----------+------+-----+---------+-------+
```

Существует альтернативный способ объявления первичного ключа, представленный в листинге 3.3.

Листинг 3.3. Альтернативный способ объявления первичного ключа

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  name TINYTEXT NOT NULL,
  PRIMARY KEY(id)
)
```

Как видно из листинга 3.3, объявление первичного ключа производится после объявления основных столбцов.

Замечание

Ключевое слово `PRIMARY KEY` может встречаться в таблице только один раз, т. к. в таблице разрешен только один первичный ключ, в противном случае сервер MySQL вернет ошибку 1068 — "Multiple primary key defined".

В качестве первичного ключа может выступать не только целочисленный столбец, но и текстовые данные. В этом случае необходимо обязательно добавить в круглых скобках число символов, входящих в индекс (листинг 3.4).

Листинг 3.4. Создание первичного ключа на текстовом столбце

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  name TINYTEXT NOT NULL,
  PRIMARY KEY(name(10))
)
```

Как видно из листинга 3.4, первичный ключ создается по первым 10 символам столбца `name`. Можно индексировать от 1 до 1000 символов текстового столбца. Следует помнить, что индексирование по одному символу приведет к тому, что уникальные символы быстро исчерпаются, и будет невозможно добавить новую запись в таблицу. Индексирование по большому числу символов приведет к резкому увеличению объема памяти, необходимого для хранения индексов таблицы.

Замечание

До версии 4.1.2 индекс по текстовому столбцу не мог содержать более 255 символов.

Индекс необязательно должен быть объявлен по одному столбцу, вполне допустимо объявление индекса сразу по двум или более (до 16) столбцам (листинг 3.5).

Листинг 3.5. Создание многостолбцового первичного ключа

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  name TINYTEXT NOT NULL,
  PRIMARY KEY(id, name(10))
);
DESCRIBE tbl;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| id    | int(11)  | NO   | PRI |         |       |
| name  | tinytext | NO   | PRI |         |       |
+-------+----------+------+-----+---------+-------+
```

Как видно из листинга 3.5, просмотр структуры таблицы `tbl` при помощи оператора `DESCRIBE` показывает, что в четвертой колонке результирующей таблицы оба поля — и `id`, и `name` — имеют флаг `PRI`, который отмечает первичный ключ.

Замечание

Значение полей `id` и `name` могут быть любыми, но их совокупность должна быть уникальна в пределах таблицы.

Когда же требуется использовать многостолбцовые индексы? К индексам зачастую прибегают в том случае, когда требуется увеличить скорость `SELECT`-запросов. Индексации подвергаются столбцы, которые помещаются в `WHERE`-условие `SELECT`-запроса. Если `WHERE`-условие использует лишь одно поле `fld`, для увеличения эффективности данного запроса индексируют одно поле `fld`. Если `WHERE`-условие содержит два или более поля, то создается многостолбцовый индекс. В листинге 3.6 демонстрируется запрос, содержащий три условия.

Листинг 3.6. Три условия в `SELECT`-запросе

```
SELECT * FROM tbl WHERE id1 = 1 AND id2 = 1002 AND id3 = 67003
```

Для запроса из листинга 3.6 наиболее рационально создать индекс по трем столбцам: `id1`, `id2` и `id3`.

3.1.2. Создание уникального индекса при помощи атрибута *AUTO_INCREMENT*

В качестве первичного ключа часто выступает столбец типа INT или BIGINT. Такой выбор связан с тем, что целочисленные типы данных обрабатываются быстрее всех и занимают небольшой объем. Другой причиной выбора целочисленного столбца является тот факт, что только данный тип столбца может быть снабжен атрибутом AUTO_INCREMENT, который обеспечивает автоматическое создание уникального индекса. Передача столбцу, снабженному этим атрибутом, значения NULL или 0 приводит к автоматическому присвоению ему максимального значения столбца плюс 1. Данный механизм является достаточно удобным и позволяет не заботиться о генерации уникального значения средствами прикладной программы, работающей с СУБД MySQL. В листинге 3.7 приводится пример использования ключевого слова AUTO_INCREMENT.

Листинг 3.7. Использование ключевого слова AUTO_INCREMENT

```
CREATE TABLE tbl (
  id INT(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,
  name TINYTEXT NOT NULL
);
DESCRIBE tbl;
+-------+----------+------+-----+---------+----------------+
| Field | Type     | Null | Key | Default | Extra          |
+-------+----------+------+-----+---------+----------------+
| id    | int(11)  | NO   | PRI | NULL    | auto_increment |
| name  | tinytext | NO   |     |         |                |
+-------+----------+------+-----+---------+----------------+
```

Просматривая структуру вновь созданной таблицы tbl при помощи оператора DESCRIBE, можно заметить, что для столбца id в шестой колонке Extra результирующей таблицы появилась запись auto_increment, сообщающая, что столбец снабжен этим атрибутом.

Как видно из листинга 3.7, для столбца id в качестве значения по умолчанию в результирующей таблице выставлено NULL, однако само поле не может принимать значение NULL. Передача NULL приведет к генерации уникального числа, которое и будет занесено в id. Если в таблице нет ни одного значения, то первое уникальное значение будет равным 1. Точно так же, как и в таблице, может быть только один первичный ключ, столбцов, снабженных атрибутом AUTO_INCREMENT, не должно быть больше одного.

В листинге 3.8 в таблицу tbl добавляется несколько записей, при этом поле id, снабженное атрибутом AUTO_INCREMENT, автоматически получает уникальный индекс.

Листинг 3.8. Автоматическая генерация уникального индекса

```
INSERT INTO tbl VALUES (NULL, 'Первая запись');
INSERT INTO tbl VALUES (NULL, 'Вторая запись');
INSERT INTO tbl VALUES (NULL, 'Третья запись');
SELECT * FROM tbl;
+----+---------------+
| id | name          |
+----+---------------+
|  1 | Первая запись |
|  2 | Вторая запись |
|  3 | Третья запись |
+----+---------------+
```

Замечание

СУБД MySQL имеет несколько переменных, управляющих поведением механизма `AUTO_INCREMENT`. Переменная `auto_increment_increment` позволяет изменить значения приращения (по умолчанию 1), а переменная `auto_increment_offset` установить начальное значение для индекса. Более подробно системные переменные описываются в *главе 9*.

3.1.3. Обычный и уникальный индексы

В отличие от первичного ключа, таблица может содержать несколько обычных и уникальных индексов. Для того чтобы их различать, индексы могут иметь собственные имена. Часто имена индексов совпадают с именами столбцов, которые они индексируют, но для индекса можно назначить и совершенно другое имя. Объявление индекса производится при помощи ключевого слова `INDEX` или `KEY`. Для уникальных индексов, которые не позволяют добавлять в таблицу записи с уже существующими значениями, вводится дополнительное ключевое слово `UNIQUE`, т. е. пишется `UNIQUE INDEX` и `UNIQUE KEY`.

Замечание

Ключевое слово `KEY` введено в MySQL, начиная с версии 4.1.0.

Замечание

Создание обычных индексов, в отличие от первичного ключа и уникального индекса, допустимо для столбцов, снабженных атрибутом `NULL`.

В листинге 3.9 приводится оператор `CREATE TABLE`, создающий таблицу `tbl`, снабженную первичным ключом `id` и двумя индексами: `id_tbl` и `id_cat`. При этом в результирующей таблице оператора `DESCRIBE` индексы помечаются флагом `MUL`.

Листинг 3.9. Создание таблицы, снабженной несколькими индексами

```
CREATE TABLE tbl (
  id INT(11) NOT NULL AUTO_INCREMENT,
  id_tbl INT(11) NOT NULL DEFAULT '0',
  id_cat INT(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (id),
  KEY id_tbl (id_tbl),
  KEY id_cat (id_cat)
);
DESCRIBE tbl;
+--------+---------+------+-----+---------+----------------+
| Field  | Type    | Null | Key | Default | Extra          |
+--------+---------+------+-----+---------+----------------+
| id     | int(11) | NO   | PRI | NULL    | auto_increment |
| id_tbl | int(11) | NO   | MUL | 0       |                |
| id_cat | int(11) | NO   | MUL | 0       |                |
+--------+---------+------+-----+---------+----------------+
```

Имена индексов совпадают с именами индексируемых столбцов, но вполне допустимо присвоить им другие имена, например:

```
KEY first (id),
KEY second (id_cat)
```

Замечание

В отличие от первичного ключа, для обычных индексов недопустимо их определение вместе со столбцом. Индекс объявляется отдельно в конце таблицы при помощи ключевого слова `INDEX` или `KEY`.

Объявление уникальных индексов в таблице `tbl` приводит к тому, что проиндексированные столбцы помечаются флагом `UNI` (листинг 3.10).

Листинг 3.10. Создание уникальных индексов

```
CREATE TABLE tbl (
  id INT(11) NOT NULL AUTO_INCREMENT,
  id_tbl INT(11) NOT NULL DEFAULT '0',
  id_cat INT(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (id),
  UNIQUE KEY id_tbl (id_tbl),
  UNIQUE KEY id_cat (id_cat)
);
```

```
DESCRIBE tbl;
+--------+---------+------+-----+---------+----------------+
| Field  | Type    | Null | Key | Default | Extra          |
+--------+---------+------+-----+---------+----------------+
| id     | int(11) | NO   | PRI | NULL    | auto_increment |
| id_tbl | int(11) | NO   | UNI | 0       |                |
| id_cat | int(11) | NO   | UNI | 0       |                |
+--------+---------+------+-----+---------+----------------+
```

Точно так же, как и первичный ключ, обычные индексы можно объявить на нескольких столбцах (до 16), как это продемонстрировано в листинге 3.11.

Листинг 3.11. Объявление индекса на двух столбцах

```
CREATE TABLE tbl (
  id INT(11) NOT NULL AUTO_INCREMENT,
  id_tbl INT(11) NOT NULL DEFAULT '0',
  id_cat INT(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (id),
  KEY id_tbl (id_tbl),
  KEY id_cat (id_cat)
  KEY multi_key (id, id_tbl, id_cat)
)
```

3.2. Добавление и удаление индексов в существующую таблицу

Часто решение об индексировании столбцов принимается после того, как таблица создана и заполнена данными. Для изменения структуры таблицы в этом случае прибегают либо к специальным операторам `CREATE INDEX`, `DROP INDEX`, либо к оператору `ALTER TABLE`.

3.2.1. Добавление индекса

Создать индекс в уже существующей таблице можно при помощи оператора `CREATE INDEX`.

Замечание

Для создания уникального индекса используется оператор `CREATE UNIQUE KEY`.

В листинге 3.12 демонстрируется создание индекса с именем `index` для столбца `id` таблицы `tbl`.

Листинг 3.12. Создание индекса `index` в таблице `tbl`

```
CREATE INDEX index ON tbl (id)
```

Если необходимо создать индекс по нескольким столбцам, они перечисляются в круглых скобках через запятую.

Точно так же, как и в случае первичного ключа, допускается индексирование текстовых столбцов, если указано число индексируемых символов (листинг 3.13).

Листинг 3.13. Создание текстового индекса `name` на 10 символах столбца `name`

```
CREATE INDEX name ON catalogs (name(10))
```

Допускается создание индекса на текстовых столбцах с использованием от 1 до 1000 символов, которые отсчитываются от начала строки.

Начиная с версии 4.1, в операторе `CREATE INDEX` можно при помощи ключевого слова `USING` указать тип индекса. В настоящее время допустимо создание индексов типа `BTREE` и `HASH`. Тип `BTREE` можно создавать в таблицах MyISAM, InnoDB и MEMORY, тип `HASH` доступен только в таблицах типа MEMORY (листинг 3.14).

Листинг 3.14. Создание индексов типа `BTREE` и `HASH`

```
CREATE INDEX id_btree USING BTREE ON tbl (id)
CREATE INDEX id_hash USING HASH ON tbl (id)
```

Оператор `ALTER TABLE`, помимо добавления новых столбцов, позволяет создавать в таблице индексы. Для добавления индексов предназначены четыре спецификации:

- `ADD INDEX [`*`index_name`*`] (`*`index_columns`*`)` — создание обычного индекса;
- `ADD UNIQUE [`*`index_name`*`] (`*`index_columns`*`)` — создание уникального индекса;
- `ADD PRIMARY KEY (`*`index_columns`*`)` — создание первичного ключа;
- `ADD FULLTEXT [`*`index_name`*`] (`*`index_columns`*`)` — создание полнотекстового поиска.

Замечание

Полнотекстовый поиск подробно обсуждается в *главе 7*.

Все индексы за исключением `PRIMARY KRY` могут быть снабжены необязательным именем *`index_name`*, в скобках указывается столбец (или столбцы, если индекс многостолбцовый) *`index_columns`*.

В листинге 3.15 создается первичный ключ для таблицы `tbl` и индексируется столбец `id_cat`.

Листинг 3.15. Создание индексов при помощи оператора `ALTER TABLE`

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  id_cat INT(11) NOT NULL DEFAULT '0'
);

ALTER TABLE tbl ADD PRIMARY KEY (id),
ADD INDEX id_cat (id_cat);

DESCRIBE tbl;
+--------+---------+------+-----+---------+-------+
| Field  | Type    | Null | Key | Default | Extra |
+--------+---------+------+-----+---------+-------+
| id     | int(11) | NO   | PRI |         |       |
| id_cat | int(11) | NO   | MUL | 0       |       |
+--------+---------+------+-----+---------+-------+
```

3.2.2. Удаление индекса

Оператор `DROP INDEX` удаляет индекс из таблицы. Пример его использования представлен в листинге 3.16.

Замечание

Создание и удаление первичного ключа при помощи операторов `CREATE INDEX` и `DROP INDEX` невозможно и осуществляется только при помощи оператора `ALTER TABLE`.

Листинг 3.16. Удаление индекса `index` из таблицы `tbl`

```
DROP INDEX index ON tbl
```

Удаление индексов осуществляется при помощи двух спецификаций:

- `DROP INDEX` *`index_name`* — удаление индекса с именем *`index_name`*;
- `DROP PRIMARY KEY` — удаление первичного ключа таблицы.

Замечание

Как уникальные (`UNIQUE`), так и обычные индексы удаляются при помощи единой конструкции `DROP INDEX`. Для удаления первичного ключа существует от-

дельная конструкция DROP PRIMARY KEY, для которой не предусмотрена передача имени индекса *index_name*, т. к. первичный ключ в таблице всегда один и не имеет имени.

Замечание

В версиях, предшествующих MySQL 5.1, использование спецификации DROP PRIMARY KEY в таблице, где отсутствовал первичный ключ, приводило к удалению первого уникального индекса. Начиная с MySQL 5.1, такое поведение изменилось, теперь при отсутствии первичного ключа использование спецификации DROP PRIMARY KEY приводит к ошибке.

В листинге 3.17 происходит удаление индексов, созданных SQL-запросом из листинга 3.15.

Листинг 3.17. Удаление индексов при помощи оператора ALTER TABLE

```
ALTER TABLE tbl DROP PRIMARY KEY,
DROP INDEX id_cat;

DESCRIBE tbl;
+--------+---------+------+-----+---------+-------+
| Field  | Type    | Null | Key | Default | Extra |
+--------+---------+------+-----+---------+-------+
| id     | int(11) | NO   |     |         |       |
| id_cat | int(11) | NO   |     | 0       |       |
+--------+---------+------+-----+---------+-------+
```

3.3. Восстановление индексов

Файл индексов может быть поврежден, в связи с чем запросы с участием проиндексированных столбцов либо просто не будут выполняться с должной производительностью, либо вообще перестанут выполняться.

Если файл данных MYD не поврежден, то файл индексов MYI всегда можно восстановить. Степень повреждения индексов может быть различной и СУБД MySQL включает множество способов устранения повреждения. В большинстве случаев достаточно передачи имени таблицы оператору CHECK TABLE или REPAIR TABLE (листинг 3.18).

Замечание

Один оператор CHECK TABLE или REPAIR TABLE может обработать сразу несколько таблиц, их следует перечислить после оператора через запятую.

Листинг 3.18. Восстановление индексов

```
CHECK TABLE tbl;
REPAIR TABLE tbl;
```

Замечание

Оператор REPAIR TABLE более подробно описывается в *разд. 2.6.1*.

В конце оператора CHECK TABLE может быть указан один из следующих параметров или их комбинация:

- QUICK — не сканировать строки в поисках некорректных связей;
- FAST — проверять только те таблицы, которые были некорректно закрыты;
- CHANGED — проверять только те таблицы, которые изменились с момента последней проверки или не были корректно закрыты;
- MEDIUM — сканировать строки для проверки того, что удаленные связи в порядке. При этом также вычисляется контрольная сумма ключа строк и сверяется с контрольной суммой всех ключей;
- EXTENDED — выполнять полный поиск всех ключей во всех строках. Гарантирует стопроцентную целостность таблиц, но требует значительного времени для проверки таблиц.

В листинге 3.19 приводится пример использования параметра QUICK совместно с оператором CHECK TABLE.

Листинг 3.19. Использование параметра в операторе CHECK TABLE

```
CHECK TABLE tbl QUICK;
```

Если не указана ни одна из опций QUICK, MEDIUM или EXTENDED, то по умолчанию выбирается MEDIUM. Можно комбинировать несколько опций, как это продемонстрировано в листинге 3.20.

Листинг 3.20. Использование комбинации опций

```
mysql> CHECK TABLE orders, users, products, catalogs FAST QUICK;
```

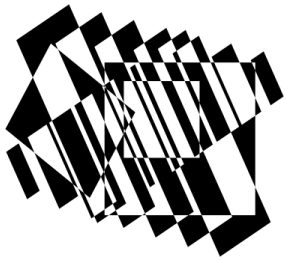
Если операторы CHECK TABLE и REPAIR TABLE не помогают ни в одном из режимов, или файл индексов таблицы MYI вообще уничтожен и остались лишь файл структуры таблицы frm и файл данных MYD, следует приступить к процедуре восстановления файла MYD. Для этого файл данных MYD нужно скопировать в безопасное место и выполнить оператор TRUNCATE TABLE (лис-

тинг 3.21), который очистит таблицу от всех данных, однако восстановит файл MYD, после чего следует скопировать ранее сохраненный файл MYD в каталог данных. Операции копирования файлов в каталоге данных лучше проводить либо когда MySQL-сервер остановлен, либо при помощи утилиты `mysqlhotcopy` *(см. разд. 1.11.1)*.

Листинг 3.21. Обнуление таблицы

```
TRUNCATE TABLE tbl;
```

ГЛАВА 4



Заполнение, обновление и очистка таблиц

Данные в таблицах хранятся в записях, для вставки новых записей в таблицу используется оператор `INSERT`, для обновления — `UPDATE` и `REPLACE`, а для удаления — операторы `DELETE` и `TRUNCATE`.

4.1. Вставка записи в таблицу. Оператор *INSERT*

Для вставки записи в таблицу используется оператор `INSERT`.

Однострочный оператор `INSERT` может использоваться в нескольких формах. Упрощенный синтаксис первой формы выглядит следующим образом:

```
INSERT [IGNORE] [INTO] tbl [(col_name,...)] VALUES (expression,...)
```

Замечание

При описании синтаксиса операторов ключевые слова, которые не являются обязательными и могут быть опущены, заключаются в квадратные скобки.

4.1.1. Вставка числовых значений

Создадим таблицу `tbl`, состоящую из двух числовых столбцов, один из которых `id` по умолчанию будет иметь значение 5, а второй `id_cat` будет принимать по умолчанию значение `NULL` (листинг 4.1).

Листинг 4.1. Таблица `tbl`

```
CREATE TABLE tbl (
  id INT(11) NOT NULL DEFAULT '5',
  id_cat INT(11) DEFAULT NULL
);
```

Существует несколько вариантов использования оператора INSERT, каждый из которых приводит к вставке новой записи (листинг 4.2).

Листинг 4.2. Вставка записей при помощи оператора INSERT

```
INSERT INTO tbl VALUES (10, 20);
INSERT INTO tbl (id_cat, id) VALUES (10, 20);
INSERT INTO tbl (id) VALUES (30);
INSERT INTO tbl () VALUES ();
```

Проверить результат вставки новых записей в таблицу можно при помощи оператора SELECT (листинг 4.3), синтаксис которого подробно разбирается в *главе 5*.

Листинг 4.3. Просмотр содержимого таблицы

```
mysql> SELECT * FROM tbl;
+----+--------+
| id | id_cat |
+----+--------+
| 10 |     20 |
| 20 |     10 |
| 30 |   NULL |
|  5 |   NULL |
+----+--------+
```

Рассмотрим различные формы оператора INSERT из листинга 4.2 более подробно. Первая форма оператора INSERT из листинга 4.2 вставляет в таблицу tbl запись (10, 20), столбцы получают значения по порядку из круглых скобок, следующих за ключевым словом VALUES. Если значений в круглых скобках будет больше или меньше, чем столбцов в таблице, то сервер MySQL вернет ошибку "Column count doesn't match value count at row 1" ("Не совпадает количество значений и столбцов в запросе").

Порядок занесения значений в запись можно изменять. Для этого следует задать порядок следования столбцов в дополнительных круглых скобках после имени таблицы. В листинге 4.2 второй оператор INSERT меняет порядок занесения значений: первое значение получает второй столбец id_cat, а второе значение — первый столбец id.

Часть столбцов можно опускать из списка — в этом случае они получают значение по умолчанию. В листинге 4.2 в третьем операторе заполняется лишь поле id, при этом поле id_cat получает значение по умолчанию — NULL. Четвертый оператор вообще не содержит значений, в этом случае все столб-

цы таблицы `tbl` получают значения по умолчанию, которые определяет ключевое слово `DEFAULT` (см. листинг 4.1). Эффекта последнего оператора можно добиться, если использовать вместо значений ключевое слово `DEFAULT`. В листинге 4.4 оба оператора эквивалентны.

Листинг 4.4. Использование ключевого слова `DEFAULT`

```
INSERT INTO tbl () VALUES ();
INSERT INTO tbl (id, id_cat) VALUES (DEFAULT, DEFAULT);
```

Приведенная в листинге 4.2 форма оператора `INSERT` является стандартной и поддерживается всеми СУБД, которые реализуют стандарт SQL. Однако СУБД MySQL поддерживает альтернативный синтаксис оператора `INSERT` (листинг 4.5).

Листинг 4.5. Альтернативный синтаксис оператора `INSERT`

```
INSERT INTO tbl SET id = 40, id_cat = 50;
INSERT INTO tbl SET id = 50;
```

Как видно из листинга 4.5, значения присваиваются столбцам при помощи ключевого слова `SET`, не перечисленные столбцы принимают значение по умолчанию.

4.1.2. Вставка строковых значений

Создадим таблицу `catalogs`, которая будет содержать два столбца: числовой столбец `id_catalog` и текстовый столбец `name` (листинг 4.6).

Листинг 4.6. Создание таблицы `catalogs`

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL,
  name TINYTEXT NOT NULL
)
```

Добавить новую запись в таблицу `catalogs` можно при помощи запроса, представленного в листинге 4.7.

Листинг 4.7. Добавление новой записи в таблицу `catalogs`

```
INSERT INTO catalogs VALUES (1, 'Процессоры');
SELECT * FROM catalogs;
```

```
+------------+------------+
| id_catalog | name       |
+------------+------------+
|          1 | Процессоры |
+------------+------------+
```

Как видно из листинга 4.7, в таблицу `catalogs` добавилась новая запись с первичным ключом `id_catalogs`, равным единице, и значением поля `name` — "Процессоры". Строковые значения необходимо помещать в кавычки, в то время как числовые значения допускается использовать без них.

Вместо одиночных кавычек можно использовать двойные кавычки (листинг 4.8). Когда в текстовое поле необходимо вставить строку, содержащую двойные кавычки, можно использовать для обрамления строки одиночные кавычки и наоборот.

Листинг 4.8. Использование двойных кавычек

```
INSERT INTO catalogs VALUES (2, "Память");
SELECT * FROM catalogs;
+------------+------------+
| id_catalog | name       |
+------------+------------+
|          1 | Процессоры |
|          2 | Память     |
+------------+------------+
```

Если вставляемая строка содержит и двойные, и одиночные кавычки, следует прибегнуть к экранированию, т. е. помещению символа `\` перед тем видом кавычек, которыми обрамляется строка (листинг 4.9).

Замечание

При помещении в базу данных текста, набранного пользователем, всегда следует экранировать кавычки, для того чтобы предотвратить возникновение ошибки и атаку SQL-инъекцией.

Листинг 4.9. Экранирование кавычек

```
INSERT INTO catalogs VALUES (3, "Память \"DDR\"");
SELECT * FROM catalogs;
+------------+--------------+
| id_catalog | name         |
+------------+--------------+
|          1 | Процессоры   |
```

```
|           2 | Память       |
|           3 | Память "DDR" |
+------------+--------------+
```

4.1.3. Вставка календарных значений

В общем случае вставка календарных значений мало чем отличается от вставки строк. Однако этот тип данных не зря выделяют в отдельный класс. Создадим таблицу `tbl`, которая будет содержать числовое поле `id` и два календарных поля: `putdate` типа `DATETIME` и `lastdate` типа `DATE` (листинг 4.10).

Листинг 4.10. Создание таблицы `tbl`

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  putdate DATETIME NOT NULL,
  lastdate DATE NOT NULL
);
```

В листинге 4.11 приводится пример вставки записи в таблицу, содержащую столбцы календарного типа.

Замечание

Календарный тип данных подробно описывается в *разд. 2.3.1*.

Листинг 4.11. Вставка календарных значений

```
INSERT INTO tbl VALUES (1, '2007-01-03 0:00:00', '2007-01-03');
SELECT * FROM tbl;
+----+---------------------+------------+
| id | putdate             | lastdate   |
+----+---------------------+------------+
|  1 | 2007-01-03 00:00:00 | 2007-01-03 |
+----+---------------------+------------+
```

Если столбцу передается "лишняя" информация, то она отбрасывается. В листинге 4.12 в столбец типа `DATE` передаются часы, минуты и секунды, однако в столбец попадает только информация о дате.

Листинг 4.12. Отбрасывание "лишней" информации

```
INSERT INTO tbl VALUES (2, '2007-01-03 0:00:00', '2007-01-03 0:00:00');
SELECT * FROM tbl;
```

```
+----+---------------------+------------+
| id | putdate             | lastdate   |
+----+---------------------+------------+
|  1 | 2007-01-03 00:00:00 | 2007-01-03 |
|  2 | 2007-01-03 00:00:00 | 2007-01-03 |
+----+---------------------+------------+
```

Зачастую календарные поля предназначены для того, чтобы пометить момент вставки записи в базу данных. Для получения текущего времени удобно воспользоваться встроенной функцией MySQL — `NOW()`. В листинге 4.13 при помощи функции `NOW()` в таблицу `tbl` вставляется запись с текущей временной меткой.

Листинг 4.13. Использование функции `NOW()`

```
INSERT INTO tbl VALUES (3, NOW(), NOW());
SELECT * FROM tbl;
+----+---------------------+------------+
| id | putdate             | lastdate   |
+----+---------------------+------------+
|  1 | 2007-01-03 00:00:00 | 2007-01-03 |
|  2 | 2007-01-03 00:00:00 | 2007-01-03 |
|  3 | 2007-01-03 15:03:33 | 2007-01-03 |
+----+---------------------+------------+
```

Вычисление текущего времени в рамках одного SQL-запроса производится только один раз, сколько бы раз они не вызывались на протяжении данного запроса. Это приводит к тому, что временное значение в рамках всего запроса остается постоянным.

Для того чтобы сместить дату относительно текущей, можно прибавлять и вычитать интервалы. Для этого используется ключевое слово `INTERVAL`, после которого следует временной интервал. В листинге 4.14 в поле `putdate` помещается дата, эквивалентная началу 2007 года за вычетом 3 недель, а в поле `lastdate` помещается дата, равная текущей плюс 3 месяца.

Листинг 4.14. Использование интервалов

```
INSERT INTO tbl
VALUES (4, '2007-01-01 0:00:00' — INTERVAL 3 WEEK,
           NOW() + INTERVAL 3 MONTH);
SELECT * FROM tbl;
+----+---------------------+------------+
| id | putdate             | lastdate   |
+----+---------------------+------------+
|  1 | 2007-01-03 00:00:00 | 2007-01-03 |
```

```
|  2 | 2007-01-03 00:00:00 | 2007-01-03 |
|  3 | 2007-01-03 15:03:33 | 2007-01-03 |
|  4 | 2006-12-11 00:00:00 | 2007-04-03 |
+----+---------------------+------------+
```

Как видно из листинга 4.14, интервалы могут быть различного типа. Полный список допустимых интервалов приводится в табл. 4.1.

Таблица 4.1. *Типы временных интервалов*

Тип	**Описание**	**Формат назначения**
MICROSECOND	Микросекунды	xxxxxx
SECOND	Секунды	ss
MINUTE	Минуты	mm
HOUR	Часы	hh
DAY	Дни	DD
WEEK	Недели	WW
MONTH	Месяцы	MM
QUARTER	Квартал	QQ
YEAR	Год	YY
SECOND_MICROSECOND	Секунды и микросекунды	'ss.xxxxxx'
MINUTE_MICROSECOND	Минуты, секунды и микросекунды	'mm:ss.xxxxxx'
MINUTE_SECOND	Минуты и секунды	'mm:ss'
HOUR_MICROSECOND	Часы, минуты, секунды и микросекунды	'hh:mm:ss.xxxxxx'
HOUR_SECOND	Часы, минуты и секунды	'hh:mm:ss'
HOUR_MINUTE	Часы и минуты	'hh:mm'
DAY_MICROSECOND	Дни, часы, минуты, секунды и микросекунды	'DD hh:mm:ss.xxxxxx'
DAY_SECOND	Дни, часы, минуты и секунды	'DD hh:mm:ss'
DAY_MINUTE	Дни, часы и минуты	'DD hh:mm'
DAY_HOUR	Дни и часы	'DD hh'
YEAR_MONTH	Года и месяцы	'YY-MM'

4.1.4. Тип данных *TIMESTAMP*

Особняком от всех календарных типов стоит тип данных `TIMESTAMP`. Ранее (до версии 4.1) этот тип данных имел по умолчанию формат UNIXSTAMP — т. е. количество секунд, прошедших с полуночи 1 января 1970 года.

Начиная с версии MySQL 4.1, формат типа `TIMESTAMP` совпадает с `DATETIME`, но во внутреннем представлении дата хранится в виде секунд, прошедших с полуночи 1 января 1970 года.

Замечание

Время в формате `TIMESTAMP` хранится как число секунд, прошедшее с полуночи 1 января 1970 года. Такое исчисление принято в операционной системе UNIX, а дата 01.01.1970 часто называется "началом эпохи UNIX" и считается днем рождения данной операционной системы.

Кроме того, поле `TIMESTAMP` может автоматически получать значения текущей даты и времени суток при создании новой записи и изменении уже существующей оператором `UPDATE`.

Замечание

В версиях MySQL 4.0 и более ранних формат типа `TIMESTAMP` представлял собой число `'YYYYMMDDhhmmss'`. Начиная с версии 4.1, формат представления был изменен на `'YYYY-MM-DD hh:mm:ss'`.

Если в таблице несколько столбцов `TIMESTAMP`, при модификации записи текущее время будет записываться только в один из столбцов (по умолчанию в первый). Можно также указать явно столбец, которому необходимо назначать текущую дату при создании новой записи или изменении старой. Для того чтобы поля столбца принимали текущую дату при создании новой записи, следует после определения столбца добавить `DEFAULT CURRENT_TIMESTAMP`. Если требуется, чтобы текущее время выставлялось при модификации уже существующей записи, при использовании оператора `UPDATE` следует добавить конструкцию `ON UPDATE CURRENT_TIMESTAMP`. Так, три оператора `CREATE TABLE`, представленные в листинге 4.15, абсолютно идентичны.

Листинг 4.15. Задание временны́х параметров при создании таблицы

```
CREATE TABLE tbl(puttime TIMESTAMP);
CREATE TABLE tbl(puttime TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                 ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE tbl(puttime TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
                 DEFAULT CURRENT_TIMESTAMP);
```

Для столбца `TIMESTAMP` можно указать только одно из ключевых слов: либо только `DEFAULT CURRENT_TIMESTAMP`, либо только `ON UPDATE CURRENT_TIMESTAMP`.

В листинге 4.16 поле `puttime1` получает текущее время при создании новой записи, а поле `puttime2` — при ее модификации.

Листинг 4.16. Задание текущего времени

```
CREATE TABLE tbl(
  puttime1 TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  puttime2 TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

4.1.5. Вставка данных в формате UNIXSTAMP

В программных средах очень распространен формат хранения данных UNIXSTAMP — количество секунд, прошедших с полуночи 1 января 1970 года. Широкое распространение данного формата вызвано в первую очередь тем, что даты можно легко вычитать друг из друга без предварительных преобразований и хранить дату не в виде строки, а в виде числа (числа всегда обрабатываются быстрее).

Для того чтобы преобразовать дату в формате MySQL в формат UNIXSTAMP, используется функция MySQL — `UNIX_TIMESTAMP()`. В листинге 4.17 создается таблица `tbl`, состоящая из двух целочисленных столбцов: `id` и `putdate`. При помощи функции `UNIX_TIMESTAMP()` в поле `putdate` помещается текущая дата.

Замечание

Если функции `UNIX_TIMESTAMP()` не передаются параметры, возвращается текущая временная метка.

Листинг 4.17. Использование функции `UNIX_TIMESTAMP()`

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  putdate INT(11) NOT NULL
);
INSERT INTO tbl VALUES(1, UNIX_TIMESTAMP());
INSERT INTO tbl VALUES(2, UNIX_TIMESTAMP(NOW()));
INSERT INTO tbl VALUES(3, UNIX_TIMESTAMP(NOW() - INTERVAL 5 DAY));
SELECT * FROM tbl;
+----+------------+
| id | putdate    |
+----+------------+
|  1 | 1167838461 |
|  2 | 1167838461 |
|  3 | 1167406461 |
+----+------------+
```

Как видно из листинга 4.17, в таблицу `tbl` вставляются три записи, первые две эквиваленты и вставляют текущее время, последняя, третья запись вставляет время, соответствующее пятидневной давности.

Замечание

Существует обратная функция — `FROM_UNIXTIME()`, которая преобразует дату в формате UNIXSTAMP в MySQL-формат.

4.1.6. Вставка уникальных значений

Первичный ключ таблицы или столбец, индексированный уникальным индексом, не могут иметь повторяющихся значений. Вставка записи со значением, уже имеющимся в таблице, приводит к возникновению ошибки (листинг 4.18).

Листинг 4.18. Значения первичного ключа должны быть уникальными

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  name TINYTEXT NOT NULL,
  PRIMARY KEY (id)
)
INSERT INTO tbl VALUES (1, 'Видеоадаптеры');
INSERT INTO tbl VALUES (1, 'Видеоадаптеры');
ERROR 1062 (23000): Duplicate entry '1' for key 1
```

Если необходимо, чтобы новые записи с дублирующим ключом отбрасывались без генерации ошибки, следует добавить после оператора `INSERT` ключевое слово `IGNORE`.

Листинг 4.19. Использование ключевого слова `IGNORE`

```
INSERT IGNORE INTO tbl VALUES (1, 'Видеоадаптеры');
SELECT * FROM tbl;
+----+---------------+
| id | name          |
+----+---------------+
|  1 | Видеоадаптеры |
+----+---------------+
```

Как видно из листинга 4.19, генерации ошибки не происходит, тем не менее, новая запись также не добавляется.

4.1.7. Механизм *AUTO_INCREMENT*

При добавлении новой записи с уникальными индексами выбор такого уникального значения может быть непростой задачей. Для того чтобы не осуществлять дополнительный запрос, направленный на выявление максимального значения первичного ключа для генерации нового уникального значения, в MySQL введен механизм его автоматической генерации. Для этого достаточно снабдить первичный ключ атрибутом `AUTO_INCREMENT`. Тогда при создании новой записи в качестве значения достаточно передать `NULL` или 0 — поле автоматически получит значение, равное максимальному значению столбца, плюс единица. В листинге 4.20 создается таблица `tbl`, состоящая из первичного ключа `id` и текстового поля `name`. Первичный ключ `id` снабжен атрибутом `AUTO_INCREMENT`.

Замечание

Текущее значение атрибута `AUTO_INCREMENT` можно выставить при помощи оператора `ALTER TABLE` *(см. разд. 2.5.5)*.

Листинг 4.20. Действие механизма `AUTO_INCREMENT`

```
CREATE TABLE tbl (
  id INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  PRIMARY KEY (id)
)
INSERT INTO tbl VALUES (NULL, 'Процессоры');
INSERT INTO tbl VALUES (NULL, 'Материнские платы');
INSERT INTO tbl VALUES (NULL, 'Видеоадаптеры');
SELECT * FROM tbl;
+----+-------------------+
| id | name              |
+----+-------------------+
|  1 | Процессоры        |
|  2 | Материнские платы |
|  3 | Видеоадаптеры     |
+----+-------------------+
```

Часто в прикладных программах требуется узнать значение, присвоенное столбцу и снабженное атрибутом `AUTO_INCREMENT`. Это может потребоваться для того, чтобы использовать сгенерированное значение первичного ключа в качестве вторичного в другой таблице или для каких-то других целей. Только что сгенерированное значение возвращает встроенная функция MySQL `LAST_INSERT_ID()` *(см. разд. 6.6)*, как это показано в листинге 4.21.

Листинг 4.21. Получение только что добавленного значения

```
SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                3 |
+------------------+
```

Например, если необходимо, чтобы следующее значение поля `id_catalog` приняло значение 10, а не 4, можно воспользоваться запросом, представленным в листинге 4.22.

Замечание

Отличительной чертой MySQL является то, что при использовании функций пробелы между именем функции и круглыми скобками недопустимы, т. е. написание `LAST_INSERT_ID()` является правильным, а `LAST_INSERT_ID ()` уже нет.

Листинг 4.22. Использование функции `LAST_INSERT_ID()`

```
INSERT INTO tbl (id, name)
VALUES (LAST_INSERT_ID() + 7, 'Оперативная память');
SELECT * FROM tbl;
+----+--------------------+
| id | name               |
+----+--------------------+
|  1 | Процессоры         |
|  2 | Материнские платы  |
|  3 | Видеоадаптеры      |
| 10 | Оперативная память |
+----+--------------------+
```

Замечание

При выборе столбца, снабженного атрибутом `AUTO_INCREMENT`, у начинающих программистов часто возникает вопрос, что происходит, когда достигается граница типа. В этом случае возникает ошибка при добавлении новой записи в таблицу, т. к. уникальных значений в пределах типа не остается. На практике достичь данного предела можно только для очень коротких типов, например, `TINYINT`. Выбор типа `INT` и `BIGINT` даже при ежедневном добавлении в таблицу 10 000 новых записей приведет к исчерпанию уникальных значений только через 588 и 2 500 000 000 000 лет соответственно. При этом даже если таблица будет содержать только один столбец, занимаемый объем данных достигнет 8 Гбайт для типа `INT` и 67 108 864 Тбайт для типа `BIGINT`.

4.1.8. Вставка вычисляемых значений

Рассматривая оператор INSERT, можно было заметить, что в качестве значений выступают либо константы, либо функции и их комбинации. Однако, кроме констант и функций, в списке вставляемых значений могут использоваться выражения с участием столбцов.

Листинг 4.23. Использование выражений

```
CREATE TABLE tbl (
  id INT(11) NOT NULL AUTO_INCREMENT,
  id_cat INT(11) NOT NULL,
  PRIMARY KEY (id)
);
INSERT tbl (id, id_cat) VALUES (1, id*3);
INSERT tbl (id, id_cat) VALUES (id_cat*3, 1);
SELECT * FROM tbl;
+----+--------+
| id | id_cat |
+----+--------+
|  1 |      3 |
|  2 |      1 |
+----+--------+
```

Так, в листинге 4.23 для первой записи поле id принимает значение 1, а поле id_cat значение 3 (3×1). Ссылаться в выражении можно только на столбцы, расположенные левее, ссылка на правые столбцы дает 0 (в случае столбца id это привело к автоматической генерации нового уникального значения 2 вместо ожидаемого 3).

Листинг 4.24. Использование выражений совместно с AUTO_INCREMENT

```
INSERT tbl (id, id_cat) VALUES (NULL, id *3);
SELECT * FROM tbl;
+----+--------+
| id | id_cat |
+----+--------+
|  1 |      3 |
|  2 |      1 |
|  3 |      0 |
+----+--------+
```

Если при вставке нового значения задействован механизм AUTO_INCREMENT, то воспользоваться вновь сгенерированным значением не удастся (листинг 4.24), так сначала полю id присваивается значение 0, и лишь после того, как запись добавлена — генерируется новое значение.

4.2. Многострочный оператор *INSERT*

Многострочный оператор INSERT совпадает по форме с однострочным оператором. В нем используется ключевое слово VALUES, после которого добавляется не один, а несколько списков *expression*. В листинге 4.25 добавляется сразу пять записей при помощи одного оператора INSERT.

Замечание

Так же как и в однострочной версии оператора INSERT, допускается использование ключевого слова IGNORE для игнорирования записей, которые имеют значения для уникальных индексов, совпадающие с одним из значений в таблице.

Листинг 4.25. Многострочный оператор INSERT

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL,
  name TINYTEXT NOT NULL
  PRIMARY KEY (id)
);
INSERT INTO catalogs VALUES (0, 'Процессоры'),
(0, 'Материнские платы'),
(0, 'Видеоадаптеры'),
(0, 'Жесткие диски'),
(0, 'Оперативная память');
```

Точно так же, как и в случае однострочного варианта, допускается изменять порядок и состав списка добавляемых значений (листинг 4.26).

Листинг 4.26. Изменение состава столбцов в многострочном INSERT

```
INSERT INTO catalogs (name) VALUES ('Процессоры'),
('Материнские платы'),
('Видеоадаптеры'),
('Жесткие диски'),
('Оперативная память');
```

4.3. Отложенная вставка записей

Совместно с оператором INSERT допускается использовать ключевое слово DELAYED, которое позволяет сразу вернуть управление клиентскому коду, не дожидаясь реальной вставки данных. То есть сервер запоминает введенные в операторе INSERT записи и вставляет их в таблицу, когда у него появляется

свободное время. Разумеется, это приводит к тому, что записи реально появляются лишь спустя некоторое время. В листинге 4.27 демонстрируется пример использования ключевого слова DELAYED совместно с INSERT.

Замечание

Ключевое слово DELAYED является расширением MySQL и отсутствует в стандарте SQL и диалектах других СУБД.

Листинг 4.27. Использование ключевого слова DELAYED совместно с INSERT

```
INSERT catalogs (id_catalog, name) VALUES (NULL, 'Новый каталог');
```

При использовании ключевого слова DELAYED на оператор INSERT накладывается ряд ограничений. Так, отложенная вставка разрешена только для таблиц типа MyISAM, MEMORY и Archive, и ключевое слово DELAYED применяется только для операторов INSERT, в которых указаны списки значений.

4.4. Пакетная загрузка данных. Оператор *INSERT ... SELECT*

Пакетная загрузка данных при помощи оператора LOAD DATA подробно рассматривается в *разд. 1.11.3*. В данном разделе будет рассмотрен еще один вариант оператора вставки данных при помощи оператора INSERT ... SELECT. Оператор INSERT ... SELECT позволяет вставлять записи из одной таблицы в другую, в том числе и осуществляя преобразование над данными.

Замечание

Оператор INSERT ... SELECT схож с оператором CREATE TABLE ... SELECT *(см. разд. 2.3.7)*, только в отличие от последнего не создает новой таблицы.

Пусть существует таблица catalogs, имеющая три поля: целочисленное поле id, снабженное атрибутом AUTO_INCREMENT и выполняющее роль первичного ключа, поле name, в котором хранится название каталога, и поле putdate, в котором помещается дата последнего обращения к каталогу. В листинге 4.28 приводятся SQL-инструкции, воссоздающие таблицу catalogs.

Листинг 4.28. Таблица catalogs

```
CREATE TABLE catalogs (
  id int(11) NOT NULL,
  name tinytext NOT NULL,
```

```
  putdate datetime NOT NULL
);
INSERT INTO tbl VALUES (1, 'Процессоры', '2005-01-19 21:40:30');
INSERT INTO tbl VALUES (2, 'Материнские платы', '2005-02-19 10:30:30');
INSERT INTO tbl VALUES (3, 'Память', '2005-02-22 16:40:00');
SELECT * FROM catalogs;
+----+-------------------+---------------------+
| id | name              | putdate             |
+----+-------------------+---------------------+
|  1 | Процессоры        | 2005-01-19 21:40:30 |
|  2 | Материнские платы | 2005-02-19 10:30:30 |
|  3 | Память            | 2005-02-22 16:40:00 |
+----+-------------------+---------------------+
```

Пусть имеется таблица `cat`, состоящая из двух полей: `name` и `putdate` (листинг 4.29).

Листинг 4.29. Таблица `cat`

```
CREATE TABLE cat (
  name tinytext NOT NULL,
  putdate datetime NOT NULL
)
```

Пусть требуется переместить данные из таблицы `catalogs` в таблицу `cat` таким образом, чтобы в поле `cat.name` помещалось поле `catalogs.name` плюс первичный ключ в круглых скобках, например, "Процессоры (1)", а в поле `cat.putdate` помещалось бы соответствующее значение из столбца `cataglos.putdate` за минусом одного часа. Решение данной задачи можно осуществить при помощи оператора `INSERT ... SELECT` (листинг 4.30).

Листинг 4.30. Использование оператора `INSERT ... SELECT`

```
INSERT INTO cat
SELECT CONCAT(name, ' (', id ,')'),
       putdate - INTERVAL 1 HOUR FROM catalogs;
SELECT * FROM cat;
+-----------------------+---------------------+
| name                  | putdate             |
+-----------------------+---------------------+
| Процессоры (1)        | 2005-01-19 20:40:30 |
| Материнские платы (2) | 2005-02-19 09:30:30 |
| Память (3)            | 2005-02-22 15:40:00 |
+-----------------------+---------------------+
```

В листинге 4.30 для объединения столбцов `name`, `id` и двух подстрок со скобками используется функция MySQL `CONCAT()`. Как видно из листинга, результирующая таблица оператора `SELECT` является основой для оператора `INSERT ... SELECT`, который заполняет таблицу `cat`. Оператор `SELECT` более подробно рассматривается в *главе 5*.

4.5. Удаление данных

Время от времени возникает задача удаления записей из базы данных, эту задачу решают следующие два оператора:

- `DELETE` — удаление всех или части записей из таблицы;
- `TRUNCATE TABLE` — удаление всех записей из таблицы.

4.5.1. Оператор *DELETE*

Оператор `DELETE` имеет следующий синтаксис:

```
DELETE FROM tbl
WHERE where_definition
ORDER BY ...
LIMIT rows
```

Оператор удаляет из таблицы *tbl* записи, удовлетворяющие условию *where_definition*. В листинге 4.31 из таблицы `catalogs` (см. листинг 4.28) удаляются записи, имеющие значение первичного ключа `id` больше двух.

Листинг 4.31. Удаление записей из таблицы `catalogs`

```
DELETE FROM catalogs WHERE id > 2;
SELECT * FROM catalogs;
+----+-------------------+---------------------+
| id | name              | putdate             |
+----+-------------------+---------------------+
|  1 | Процессоры        | 2005-01-19 21:40:30 |
|  2 | Материнские платы | 2005-02-19 10:30:30 |
+----+-------------------+---------------------+
```

Если в операторе `DELETE` отсутствует условие `WHERE`, из таблицы удаляются все записи (листинг 4.32).

Листинг 4.32. Удаление всех записей таблицы `catalogs`

```
DELETE FROM catalogs;
SELECT * FROM catalogs;
Empty set (0.00 sec)
```

Применение ограничения LIMIT позволяет задать максимальное число записей, которые могут быть уничтожены. В листинге 4.33 удаляется не более 3 записей таблицы catalogs.

Листинг 4.33. Удаление записей из таблицы catalogs

```
DELETE FROM catalogs LIMIT 3;
```

4.5.2. Оператор *TRUNCATE*

Оператор TRUNCATE TABLE, в отличие от оператора DELETE, полностью очищает таблицу и не допускает условного удаления. То есть оператор TRUNCATE TABLE аналогичен оператору DELETE без условия WHERE и ограничения LIMIT. В отличие от оператора DELETE, удаление происходит гораздо быстрее, т. к. осуществляется не перебор каждой записи, а полное очищение таблицы. Пример использования оператора TRUNCATE TABLE приводится в листинге 4.34.

Замечание

Оптимизатор запросов СУБД MySQL автоматически использует оператор TRUNCATE TABLE, если оператор DELETE не содержит WHERE-условия или конструкции LIMIT.

Листинг 4.34. Удаление всех записей таблицы products

```
TRUNCATE TABLE products;
```

4.5.3. Удаление из нескольких таблиц

Оператор DELETE позволяет удалять записи одновременно из нескольких таблиц. Для этого сразу после ключевого слова FROM имена таблиц перечисляются через запятую.

В листинге 4.35 представлены две таблицы: catalogs и products. Каждая запись таблицы products содержит внешний ключ id_catalog, который устанавливает связь между таблицами так, как это представлено на рис. 4.1.

Листинг 4.35. Таблицы catalogs и products

```
CREATE TABLE catalogs (
  id_catalog int(11) NOT NULL auto_increment,
  name tinytext NOT NULL,
  PRIMARY KEY  (id_catalog)
);
```

```
INSERT INTO catalogs VALUES (1, 'Процессоры'), (2, 'Материнские платы'),
(3, 'Видеоадаптеры'), (4, 'Жесткие диски'), (5, 'Оперативная память');

CREATE TABLE products (
  id_product int(11) NOT NULL auto_increment,
  name tinytext NOT NULL,
  id_catalog int(11) NOT NULL,
  PRIMARY KEY  (id_product),
  KEY id_catalog (id_catalog)
);

INSERT INTO products VALUES (1, 'Материнская плата N 1', 2),
(2, 'Оперативная память N 1', 5), (3, 'Процессор N 1', 1),
(4, 'Процессор N 2', 1), (5, 'Процессор N 3', 1),
(6, 'Оперативная память N 2', 5), (7, 'Материнская плата N 2', 2),
(8, 'Жесткий диск N 1', 4), (9, 'Видеоадаптер N 1', 3);
```

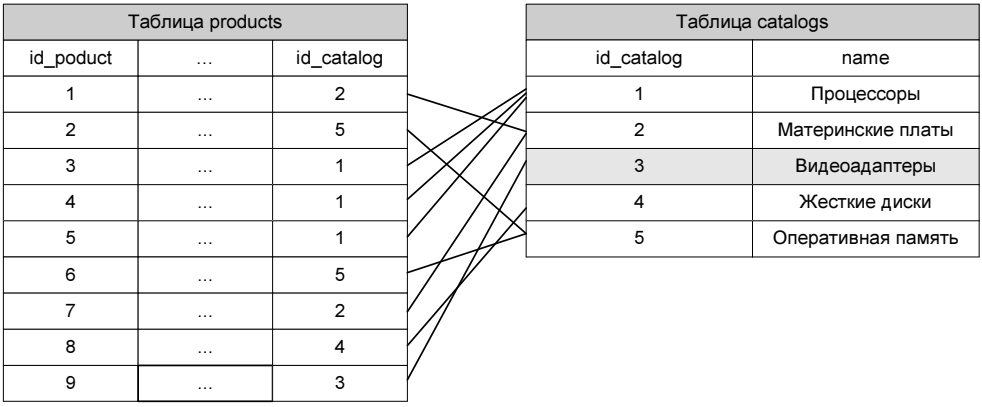


Рис. 4.1. Связанные друг с другом таблицы

Пусть требуется удалить из таблицы `catalogs` все записи, чей первичный ключ `id_catalog` больше 3. Для этого можно использовать запрос, представленный в листинге 4.36.

Замечание

В многотабличных запросах с участием SQL-оператора `DELETE` не допускается использование конструкций `ORDER BY` и `LIMIT`.

Листинг 4.36. Удаление записей из таблицы `catalogs`

```
DELETE FROM catalogs WHERE id_catalog > 3;
SELECT * FROM catalogs;
```

```
+------------+-------------------+
| id_catalog | name              |
+------------+-------------------+
|          1 | Процессоры        |
|          2 | Материнские платы |
|          3 | Видеоадаптеры     |
+------------+-------------------+
```

Запрос из листинга 4.36 уничтожит записи в таблице `catalogs` для элементов каталога "Жесткие диски" и "Оперативная память" с первичными ключами `id_catalog`, равными 4 и 5 соответственно. Однако записи в таблице `products` останутся нетронутыми и будут ссылаться на несуществующие позиции каталога. Сложившуюся ситуацию можно исправить, если воспользоваться многотабличным оператором `DELETE`. При использовании многотабличных запросов с помощью оператора `DELETE` допустимы две формы, представленные в листинге 4.37, где из таблицы `products` удаляются все записи, для которых имеется соответствие в таблице `catalogs`.

Листинг 4.37. Многотабличное удаление из таблицы `products`

```
DELETE products FROM products, catalogs
WHERE products.id_catalog = catalogs.id_catalog;

DELETE FROM products USING products, catalogs
WHERE products.id_catalog = catalogs.id_catalog;
```

Следует отметить, что записи касаются только таблицы `products` — удаление производится лишь из таблиц, перечисленных после ключевого слова `DELETE` в первой форме запроса, и после ключевого слова `FROM` во второй форме запроса. Запросы, представленные в листинге 4.38, затронут уже обе таблицы.

Листинг 4.38. Многотабличное удаление из таблиц `products` и `catalogs`

```
DELETE products, catalogs FROM products, catalogs
WHERE products.id_catalog = catalogs.id_catalog;

DELETE FROM products, catalogs USING products, catalogs
WHERE products.id_catalog = catalogs.id_catalog;
```

Таким образом, возвращаясь к задаче удаления позиций каталога с первичным ключом больше 3, можно сформировать следующий SQL-запрос, который удалит записи как из таблицы `catalogs`, так и из таблицы `products` (листинг 4.39).

Листинг 4.39. Удаление позиций каталога с первичным ключом больше 3

```
DELETE products, catalogs FROM products, catalogs
WHERE products.id_catalog = catalogs.id_catalog AND
      catalogs.id_catalog > 3;
SELECT * FROM catalogs;
+------------+-------------------+
| id_catalog | name              |
+------------+-------------------+
|          1 | Процессоры        |
|          2 | Материнские платы |
|          3 | Видеоадаптеры     |
+------------+-------------------+

SELECT * FROM products;
+------------+----------------------+------------+
| id_product | name                 | id_catalog |
+------------+----------------------+------------+
|          1 | Материнская плата N 1 |          2 |
|          3 | Процессор N 1        |          1 |
|          4 | Процессор N 2        |          1 |
|          5 | Процессор N 3        |          1 |
|          7 | Материнская плата N 2 |          2 |
|          9 | Видеоадаптер N 1     |          3 |
+------------+----------------------+------------+
```

Как видно из листинга 4.39, многотабличное удаление позволяет удалить позиции каталога, не нарушая ссылочной целостности базы данных.

Для достижения совместимости с MS Access в многотабличных операторах `DELETE` допускается использование символов `.*` после имени таблицы (листинг 4.40).

Листинг 4.40. Использование символов `.*`

```
DELETE products.*, catalogs.* FROM products, catalogs
WHERE products.id_catalog = catalogs.id_catalog;
DELETE FROM products.*, catalogs.* USING products, catalogs
WHERE products.id_catalog = catalogs.id_catalog;
```

При использовании псевдонимов таблиц, назначаемых оператором `AS`, в СУБД MySQL 4.0 необходимо обращаться к таблицам с помощью реальных имен (листинг 4.41).

Листинг 4.41. Использование ключевого слова AS в СУБД MySQL 4.0

```
DELETE products, catalogs FROM products AS p, catalogs AS c
WHERE p.id_catalog = c.id_catalog;
```

Начиная с версии СУБД MySQL 4.1, псевдонимы требуется использовать в тексте всего оператора DELETE (листинг 4.42).

Листинг 4.42. Использование ключевого слова AS в СУБД MySQL 4.1 и выше

```
DELETE p, c FROM products AS p, catalogs AS c
WHERE p.id_catalog = c.id_catalog;
```

4.5.4. Каскадное удаление из нескольких таблиц

В предыдущем разделе описывалась ситуация, когда удаление записей может приводить к нарушению ссылочной целостности в двух связанных таблицах. СУБД MySQL предоставляет средства для автоматического отслеживания таких связей. Для этого используется вторичный ключ, задаваемый конструкцией FOREIGN KEY, которая имеет следующий синтаксис:

```
FOREIGN KEY [name_key] (col1, ...) REFERENCES tbl (tbl_col, ...)
  [ON DELETE {CASCADE|SET NULL|NO ACTION|RESTRICT|SET DEFAULT}]
  [ON UPDATE {CASCADE|SET NULL|NO ACTION|RESTRICT|SET DEFAULT}]
```

Данная конструкция позволяет задать внешний ключ с необязательным именем *name_key* на столбцах, которые перечисляются в круглых скобках (один или несколько). Ключевое слово REFERENCES указывает таблицу *tbl*, на которую ссылается внешний ключ, в круглых скобках задаются имена столбцов. Необязательные конструкции ON DELETE и ON UPDATE позволяют определить поведение СУБД при удалении и обновлении строк из таблицы-предка, соответственно. Параметры, следующие за этими ключевыми словами, имеют следующие значения:

- CASCADE — при удалении или обновлении записи, содержащей первичный ключ, записи со ссылками на это значение в таблице-потомке удаляются или обновляются автоматически;
- SET NULL — при удалении или обновлении записи, содержащей первичный ключ, в таблице-потомке значения внешнего ключа, ссылающегося на таблицу-предка, устанавливаются в NULL;
- NO ACTION — при удалении или обновлении записей, содержащих первичный ключ, с таблицей-потомком никаких дополнительных действий не производится;

- `RESTRICT` — если в таблице-потомке имеются записи, ссылающиеся на первичный ключ таблицы-предка, при удалении или обновлении записей с таким первичным ключом возвращается ошибка. СУБД не позволяет изменять или удалять запись с первичным ключом, пока не останется ни одной ссылки из таблицы-потомка;
- `SET DEFAULT` — согласно стандарту SQL, при удалении или обновлении первичного ключа в таблице-потомке для ссылающихся на него записей в поле внешнего ключа должно устанавливаться значение по умолчанию. Однако в СУБД MySQL это ключевое слово зарезервировано, но не обрабатывается.

Снабдим таблицы `catalogs` и `products` внешним ключом (листинг 4.43).

Замечание

Внешние ключи поддерживаются только таблицами типа InnoDB.

Листинг 4.43. Создание внешних ключей

```
CREATE TABLE catalogs (
  id_catalog int(11) NOT NULL AUTO_INCREMENT,
  name tinytext NOT NULL,
  PRIMARY KEY (id_catalog)
 ) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
CREATE TABLE products (
  id_product int(11) NOT NULL AUTO_INCREMENT,
  name tinytext NOT NULL,
  id_catalog int(11) NOT NULL default '0',
  PRIMARY KEY  (id_product),
  FOREIGN KEY (id_catalog) REFERENCES catalogs (id_catalog)
      ON DELETE CASCADE
      ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
```

В листинге 4.43 в табличном пространстве InnoDB создаются таблицы `catalogs` и `products`. В таблице `products` поле `id_catalog` объявлено как внешний ключ (`FOREIGN KEY`) с правилом удаления и обновления `CASCADE`. Это означает, что обновление удаления любого из элементов каталога в таблице `catalogs` приведет к тому, что все записи таблицы `products`, поле `id_catalog` которых равно первичному ключу удаляемой записи каталога, также будут удалены из таблицы (листинг 4.44).

Листинг 4.44. Каскадное удаление

```
DELETE FROM catalogs WHERE id_catalog > 2;
SELECT * FROM catalogs;
```

```
+------------+-------------------+
| id_catalog | name              |
+------------+-------------------+
|          1 | Процессоры        |
|          2 | Материнские платы |
+------------+-------------------+
SELECT id_product, id_catalog, name FROM products;
+------------+------------+-----------------------+
| id_product | id_catalog | name                  |
+------------+------------+-----------------------+
|          1 |          2 | Материнская плата N 1 |
|          3 |          1 | Процессор N 1         |
|          4 |          1 | Процессор N 2         |
|          5 |          1 | Процессор N 3         |
|          7 |          2 | Материнская плата N 2 |
+------------+------------+-----------------------+
```

Как видно из листинга 4.44, удаление записей с первичным ключом приводит к автоматическому удалению ссылающихся на него записей с внешним ключом. Записи с внешними ключами сами могут выступать в качестве ссылок для внешних ключей других таблиц, и один-единственный запрос может осуществлять достаточно сложные преобразования базы данных. Удаление и обновление базы данных, когда таблицы связаны внешним ключом с правилом `CASCADE`, называется *каскадным удалением или обновлением*.

Замечание

Внешний ключ может рекурсивно ссылаться на записи своей же таблицы. Это часто необходимо, когда требуется моделировать реферальные системы и вложенные каталоги при помощи одной и той же таблицы.

СУБД MySQL позволяет отключить проверку ограничений внешнего ключа. Для этого достаточно установить для системной переменной `FOREIGN_KEY_CHECKS` значение 0. Установка системных переменных осуществляется при помощи оператора `SET` (листинг 4.45).

Листинг 4.45. Управление системной переменной `FOREIGN_KEY_CHECKS`

```
SELECT @@FOREIGN_KEY_CHECKS;
+----------------------+
| @@FOREIGN_KEY_CHECKS |
+----------------------+
|                    1 |
+----------------------+
FOREIGN_KEY_CHECKS = 1;
```

Для того чтобы извлечь текущее значение системной переменной, ее имя следует предварить двумя символами `@`. Более подробно системные переменные и синтаксис оператора `SET` обсуждается в *главе 9*.

4.6. Обновление записей

Операция обновления позволяет менять значения полей в уже существующих записях. Для обновления данных предназначены операторы `UPDATE` и `REPLACE`. Первый позволяет обновлять отдельные поля в уже существующих записях, тогда как оператор `REPLACE` больше похож на `INSERT`, за исключением того, что если старая запись в данной таблице имеет то же значение индекса `UNIQUE` или `PRIMARY KEY`, что и новая, то старая запись перед занесением новой записи будет удалена.

4.6.1. Оператор *UPDATE*

Оператор `UPDATE` имеет следующий синтаксис:

```
UPDATE [IGNORE] tbl
SET col1=expr1 [, col2=expr2 ...]
[WHERE where_definition]
[ORDER BY ...]
[LIMIT rows]
```

В инструкции сразу после ключевого слова `UPDATE` указывается таблица *`tbl`*, которая подвергается изменению. В предложении `SET` перечисляются столбцы, которые подвергаются обновлению, и устанавливаются их новые значения. Необязательное условие `WHERE` позволяет задать критерий отбора строк — обновлению будут подвергаться только те строки, которые удовлетворяют условию *`where_definition`*.

Пусть имеются таблицы `catalogs` и `products`, структура и содержимое которых определяется листингом 4.35. Заменим название элемента каталога "Процессоры" в таблице `catalogs` на "Процессоры (Intel)" (листинг 4.46).

Листинг 4.46. Обновление таблицы `catalogs`

```
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
```

```
|           4 | Жесткие диски      |
|           5 | Оперативная память |
+------------+--------------------+
UPDATE catalogs SET name = 'Процессоры (Intel)'
WHERE name = 'Процессоры';
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|           1 | Процессоры (Intel) |
|           2 | Материнские платы  |
|           3 | Видеоадаптеры      |
|           4 | Жесткие диски      |
|           5 | Оперативная память |
+------------+--------------------+
```

4.6.2. Многотабличный оператор *UPDATE*

Использование многотабличного оператора `UPDATE` сходно с вызовом оператора `DELETE` *(см. разд. 4.5.3)*. Имена таблиц, которые подвергаются изменению, перечисляются после ключевого слова `UPDATE` через запятую. Пусть требуется изменить первичный ключ `id_catalog` таблицы `catalogs` для элемента каталога "Оперативная память" с 5 на 10. Для этого можно использовать запрос, представленный в листинге 4.47.

Листинг 4.47. Редактирование таблицы `catalogs`

```
UPDATE catalogs SET id_catalog = 10 WHERE id_catalog = 5
```

Однако изменения коснулись только таблицы `catalogs`, значения внешних ключей в таблице `products` не изменились. Исправить ситуацию может многотабличный запрос, представленный в листинге 4.48.

Листинг 4.48. Обновление сразу двух таблиц

```
UPDATE catalogs, products
SET catalogs.id_catalog = 10, products.id_catalog = 10
WHERE catalogs.id_catalog = 5 AND products.id_catalog = 5;
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|           1 | Процессоры (Intel) |
|           2 | Материнские платы  |
|           3 | Видеоадаптеры      |
```

```
|           4 | Жесткие диски      |
|          10 | Оперативная память |
+------------+--------------------+
SELECT id_catalog FROM products;
+------------+
| id_catalog |
+------------+
|           1 |
|           1 |
|           1 |
|           2 |
|           2 |
|           3 |
|           4 |
|          10 |
|          10 |
+------------+
```

Пусть в таблицу `products` требуется добавить столбец `catalog`, который необходимо заполнить названиями элементов каталога из таблицы `catalogs`. Добавить новый столбец можно при помощи оператора `ALTER TABLE` (листинг 4.49), синтаксис которого подробно рассматривается в *главе 3*.

Листинг 4.49. Добавление нового столбца `catalog` в таблицу `products`

```
ALTER TABLE products ADD catalog TINYTEXT AFTER id_catalog;
DESCRIBE products;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| id_product | int(11)  | NO   | PRI | NULL    | auto_increment |
| name       | tinytext | NO   |     |         |                |
| id_catalog | int(11)  | NO   | MUL | 0       |                |
| catalog    | tinytext | YES  |     | NULL    |                |
+------------+----------+------+-----+---------+----------------+
```

Однако поле `catalog` в настоящий момент является пустым. Для того чтобы заполнить его, можно прибегнуть к многотабличному запросу `UPDATE` (листинг 4.50).

Листинг 4.50. Заполнение поля `catalog`

```
SELECT id_catalog, catalog FROM products;
+------------+---------+
| id_catalog | catalog |
+------------+---------+
|           2 | NULL    |
```

```
|          10 | NULL    |
|           1 | NULL    |
|           1 | NULL    |
|           1 | NULL    |
|          10 | NULL    |
|           2 | NULL    |
|           4 | NULL    |
|           3 | NULL    |
+------------+---------+

UPDATE products, catalogs SET products.catalog = catalogs.name
WHERE products.id_catalog = catalogs.id_catalog;

SELECT id_catalog, catalog FROM products;
+------------+--------------------+
| id_catalog | catalog            |
+------------+--------------------+
|           2 | Материнские платы  |
|          10 | Оперативная память |
|           1 | Процессоры (Intel) |
|           1 | Процессоры (Intel) |
|           1 | Процессоры (Intel) |
|          10 | Оперативная память |
|           2 | Материнские платы  |
|           4 | Жесткие диски      |
|           3 | Видеоадаптеры      |
+------------+--------------------+
```

Как видно из листинга 4.50, таблица `catalogs` не подвергается изменениям, но активно участвует в многотабличном запросе с применением оператора `UPDATE`.

Замечание

В многотабличных запросах с участием SQL-оператора `UPDATE` не допускается использование конструкций `ORDER BY` и `LIMIT`.

Рассмотренные выше примеры используют перекрестное объединение, но многотабличные запросы с участием оператора `UPDATE` допускают любой тип объединения. Подробнее объединения рассматриваются в *главе 5*.

4.6.3. Оператор *REPLACE*

Оператор `REPLACE` работает аналогично `INSERT`, за исключением того, что если старая запись в данной таблице имеет то же значение индекса `UNIQUE` или `PRIMARY KEY`, что и новая, то старая запись перед занесением новой будет удалена. Следует учитывать, что если не используются индексы `UNIQUE` или

PRIMARY KEY, то применение команды REPLACE не имеет смысла, т. к. она работает просто как INSERT.

Синтаксис оператора RENAME аналогичен синтаксису оператора INSERT:

REPLACE [INTO] *tbl* [(*col_name*,...)] VALUES (*expression*,...),(...),...

В таблицу *tbl* вставляются значения, определяемые в списке после ключевого слова VALUES. Задать порядок столбцов можно при помощи необязательного списка *col_name*, следующего за именем таблицы *tbl*. Точно так же, как и в случае оператора INSERT, оператор REPLACE допускает многострочный формат. Пример использования оператора приведен в листинге 4.51, где в таблицу catalogs добавляется пять новых записей.

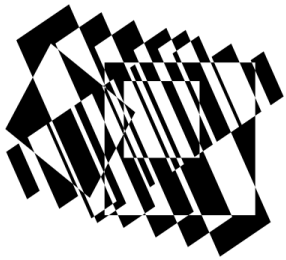
Замечание

Многотабличный синтаксис для операторов REPLACE и INSERT не предусмотрен.

Листинг 4.51. Использование оператора REPLACE

```
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры (Intel) |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
REPLACE INTO catalogs VALUES
(4, 'Сетевые адаптеры'),
(5, 'Программное обеспечение'),
(6, 'Мониторы'),
(7, 'Периферия'),
(8, 'CD-RW/DVD');
SELECT * FROM catalogs ORDER BY id_catalog;
+------------+-------------------------+
| id_catalog | name                    |
+------------+-------------------------+
|          1 | Процессоры (Intel)      |
|          2 | Материнские платы       |
|          3 | Видеоадаптеры           |
|          4 | Сетевые адаптеры        |
|          5 | Программное обеспечение |
|          6 | Мониторы                |
|          7 | Периферия               |
|          8 | CD-RW/DVD               |
+------------+-------------------------+
```

ГЛАВА 5



Извлечение данных

В данной главе описываются средства извлечения данных из таблиц СУБД MySQL. Основным оператором, при помощи которого производится выборка, является оператор `SELECT`. Упрощенный синтаксис оператора `SELECT` выглядит следующим образом:

```
SELECT select_expr [FROM tbl]
WHERE where_condition
GROUP BY col
HAVING having_condition
ORDER BY col
LIMIT start, col
```

Каждый запрос начинается с ключевого слова `SELECT`, за ним следует список полей, разделенных запятыми, *`select_expr`*, которые будут возвращены в результате запроса. Ключевое слово `FROM`, указывающее, из какой таблицы извлекаются данные, является необязательным и может быть опущено.

Результатом запроса `SELECT` всегда является таблица, которая ничем не отличается от таблиц, расположенных в базе данных, и называется *результирующей таблицей*. Такая результирующая таблица может быть использована для создания новой или заполнения другой таблицы, или выступать в качестве таблицы базы данных для других SQL-запросов.

5.1. Простейшая выборка из таблицы

Рассмотрим запросы на выборку данных на примере таблицы `catalogs`. Таблица имеет два поля: первичный ключ `id_catalog` и название элемента каталога `name` (листинг 5.1).

Листинг 5.1. Таблица `catalogs`

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL,
  name TINYTEXT NOT NULL
  PRIMARY KEY (id)
);
INSERT INTO catalogs VALUES (0, 'Процессоры'),
(0, 'Материнские платы'),
(0, 'Видеоадаптеры'),
(0, 'Жесткие диски'),
(0, 'Оперативная память');
```

Выбрать все записи таблицы `catalogs` можно при помощи запроса, представленного в листинге 5.2.

Листинг 5.2. Выборка данных при помощи оператора `SELECT`

```
SELECT id_catalog, name FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры         |
+------------+--------------------+
```

Если требуется вывести все столбцы таблицы, необязательно перечислять их имена после ключевого слова `SELECT`, достаточно заменить этот список символом `*` (все столбцы) (листинг 5.3).

Листинг 5.3. Использование символа `*` (все столбцы)

```
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры         |
+------------+--------------------+
```

К списку столбцов в операторе `SELECT` прибегают в том случае, если необходимо изменить порядок следования столбцов в результирующей таблице или выбрать только часть столбцов (листинг 5.4).

Листинг 5.4. Изменение числа и порядка выводимых столбцов

```
SELECT name, id_catalog FROM catalogs;
+--------------------+------------+
| name               | id_catalog |
+--------------------+------------+
| Видеоадаптеры      |          3 |
| Жесткие диски      |          4 |
| Материнские платы  |          2 |
| Оперативная память |          5 |
| Процессоры         |          1 |
+--------------------+------------+
```

В списке *`select_expr`* наряду с именем столбца допустимо использование констант, например, числа 5 или строки `'comments'`. В этом случае каждая строка результирующей таблицы содержит поля `5` и `'comments'` (листинг 5.5).

Листинг 5.5. Использование констант в списке столбцов

```
SELECT name, id_catalog, 5, 'comments' FROM catalogs;
+--------------------+------------+---+----------+
| name               | id_catalog | 5 | comments |
+--------------------+------------+---+----------+
| Видеоадаптеры      |          3 | 5 | comments |
| Жесткие диски      |          4 | 5 | comments |
| Материнские платы  |          2 | 5 | comments |
| Оперативная память |          5 | 5 | comments |
| Процессоры         |          1 | 5 | comments |
+--------------------+------------+---+----------+
```

5.2. Условная выборка

Ситуация, когда требуется изменить количество выводимых строк, встречается гораздо чаще, чем ситуация, когда требуется изменить число и порядок выводимых столбцов. Для ввода в SQL-запрос такого рода ограничений в операторе `SELECT` предназначено специальное ключевое слово `WHERE`, после которого следует логическое условие. Если запись удовлетворяет такому условию, она попадает в результат выборки, в противном случае такая запись отбрасывается.

В листинге 5.6 приводится пример запроса, извлекающего из таблицы `catalogs` записи, чей первичный ключ `id_catalog` больше (оператор >) 2.

Листинг 5.6. Использование конструкции `WHERE`

```
SELECT * FROM catalogs WHERE id_catalog > 2;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
```

Оператор больше >, который возвращает `true` (истина), если левый аргумент больше правого, и `false` (ложь), если правый аргумент меньше левого. Если логическое выражение возвращает `true` для текущей записи, запись попадает в результирующую таблицу.

Помимо оператора > имеется еще несколько логических операторов, представленных в табл. 5.1.

Таблица 5.1. *Логические операторы*

Оператор	Описание
`a > b`	Возвращает `true` (истина), если аргумент `a` больше `b`, в противном случае возвращается `false` (ложь)
`a < b`	Возвращает `true` (истина), если аргумент `a` меньше аргумента `b`, в противном случае возвращается `false` (ложь)
`a >= b`	Возвращает `true` (истина), если аргумент `a` больше или равен аргументу `b`, в противном случае возвращается `false` (ложь)
`a <= b`	Возвращает `true` (истина), если аргумент `a` меньше или равен аргументу `b`, в противном случае возвращается `false` (ложь)
`a = b`	Возвращает `true` (истина), если аргумент `a` равен аргументу `b`, в противном случае возвращается `false` (ложь)
`a <> b`	Возвращает `true` (истина), если аргумент `a` не равен аргументу `b`, в противном случае возвращается `false` (ложь)
`a != b`	Синоним оператора <>
`a <=> b`	Оператор эквивалентности по своему действию аналогичен оператору равенства =, однако допускает в качестве одного из аргументов `NULL`

Следует отметить, что логические операторы возвращают `true` (истина) и `false` (ложь) в стиле языка программирования C, т. е. специальных констант и обозначений нет. За ложь считается 0, а за истину — любое число, отличное от нуля. В этом легко убедиться, если вывести логическое выражение в результирующую таблицу (листинг 5.7).

Листинг 5.7. Вывод логических значений в результирующую таблицу

```
SELECT id_catalog, id_catalog > 2 FROM catalogs;
+------------+----------------+
| id_catalog | id_catalog > 2 |
+------------+----------------+
|          1 |              0 |
|          2 |              0 |
|          3 |              1 |
|          4 |              1 |
|          5 |              1 |
+------------+----------------+
```

5.3. Комбинированные условия

Условие может быть составным и объединяться при помощи логических операторов. В листинге 5.8 используется составное условие: первичный ключ должен быть больше двух и меньше или равен 4. Для объединения этих двух условий используется оператор `AND` (И).

Замечание

Помимо оператора `AND` (И), для объединения логических выражений может использоваться оператор `OR` (ИЛИ).

Листинг 5.8. Использование составного логического условия

```
SELECT * FROM catalogs
WHERE id_catalog > 2 AND id_catalog <= 4;
+------------+---------------+
| id_catalog | name          |
+------------+---------------+
|          3 | Видеоадаптеры |
|          4 | Жесткие диски |
+------------+---------------+
```

В табл. 5.2 и 5.3 представлены правила, по которым операторы `AND` и `OR` объединяют свои операнды.

Таблица 5.2. *Оператор* AND

Операнд/результат	Значения			
Первый операнд	true	true	false	false
Второй операнд	true	false	true	false
Результат	true	false	false	false

Как видно из табл. 5.2, оператор AND возвращает true только в том случае, если оба его операнда принимают истинное значение (true).

Таблица 5.3. *Оператор* OR

Операнд/результат	Значения			
Первый операнд	true	true	false	false
Второй операнд	true	false	true	false
Результат	true	true	true	false

Как видно из табл. 5.3, оператор OR возвращает false только в том случае, если оба его операнда принимают ложное значение (false).

Помимо бинарных логических операторов AND и OR, СУБД MySQL поддерживает унарный оператор отрицания NOT. Оператор возвращает истину для ложного аргумента и ложь для истинного аргумента. В листинге 5.9 демонстрируется использование оператор NOT.

Листинг 5.9. Использование оператора NOT

```
SELECT id_catalog, id_catalog > 2, NOT id_catalog > 2 FROM catalogs;
+------------+----------------+--------------------+
| id_catalog | id_catalog > 2 | NOT id_catalog > 2 |
+------------+----------------+--------------------+
|          1 |              0 |                  1 |
|          2 |              0 |                  1 |
|          3 |              1 |                  0 |
|          4 |              1 |                  0 |
|          5 |              1 |                  0 |
+------------+----------------+--------------------+
```

Помимо операторов OR и AND, язык SQL предоставляет еще один логический оператор: исключающее ИЛИ — XOR. Правила, по которым работает данный оператор, представлены в табл. 5.4.

Таблица 5.4. *Оператор* `XOR`

Операнд/результат	Значения			
Первый операнд	`true`	`true`	`false`	`false`
Второй операнд	`true`	`false`	`true`	`false`
Результат	`false`	`true`	`true`	`false`

Оператор `XOR` можно эмулировать при помощи остальных логических операторов по формуле — `(a AND (NOT b)) OR ((NOT a) and b)`.

5.4. *NULL* и логические операторы

Значение `NULL` является специальным значением в SQL и призвано обозначать объект, информация о котором отсутствует. Все операции с оператором `NULL` приводят к результату `NULL`. Это легко можно понять, ведь если мы неизвестное значение сравниваем с числом (`2 > NULL`), то не знаем, больше оно этого числа или нет, т. е. в результате опять получаем неопределенное значение, т. е. `NULL` (листинг 5.10).

Листинг 5.10. Сравнение числа с `NULL`

```
SELECT 2 > NULL;
+----------+
| 2 > NULL |
+----------+
|     NULL |
+----------+
```

Точно так же и арифметические операции со значением `NULL` возвращают `NULL`, т. к., сложив число с неизвестным значением, в результате опять получается неизвестное значение (`NULL`).

Иногда, впрочем, необходимо доподлинно выяснить, равно значение столбца `NULL` или нет. При помощи обычного оператора равенства `=` осуществить такое сравнение невозможно, т. к. любое сравнение со значением `NULL` приводит к `NULL`. Поэтому используется специальный оператор эквивалентности `<=>` (см. табл. 5.1). В листинге 5.11 демонстрируется поведение оператора `<=>`.

Листинг 5.11. Сравнение с использованием оператора эквивалентности <=>

```
SELECT 1<=>1, 1<=>2, NULL<=>1, NULL<=>NULL;
```

```
+-------+-------+----------+-------------+
| 1<=>1 | 1<=>2 | NULL<=>1 | NULL<=>NULL |
+-------+-------+----------+-------------+
|     1 |     0 |        0 |           1 |
+-------+-------+----------+-------------+
```

Помимо оператора эквивалентности, СУБД MySQL предоставляет еще два оператора для работы с NULL-значениями: IS NULL и IS NOT NULL. Оператор IS NULL позволяет определить, равно ли проверяемое значение NULL или нет (листинг 5.12). Если значение равно NULL, оператор возвращает 1 (истина), в противном случае возвращается 0 (ложь).

Листинг 5.12. Использование конструкции IS NULL

```
SELECT NULL IS NULL, 0 IS NULL;
+--------------+-----------+
| NULL IS NULL | 0 IS NULL |
+--------------+-----------+
|            1 |         0 |
+--------------+-----------+
```

Замечание

Конструкция IS NULL также возвращает 1 (истина) для нулевых полей типа DATETIME — '0000-00-00 00:00:00'. Это необходимо для ODBC-приложений, т. к. этот интерфейс не поддерживает нулевое время, вместо него используется NULL.

Существует альтернативная форма записи конструкции IS NULL — функция ISNULL(), применение которой демонстрируется в листинге 5.13.

Листинг 5.13. Использование функции ISNULL()

```
SELECT ISNULL(NULL), ISNULL(0);
+--------------+-----------+
| ISNULL(NULL) | ISNULL(0) |
+--------------+-----------+
|            1 |         0 |
+--------------+-----------+
```

Конструкция IS NOT NULL является противопоставлением оператора IS NULL. Если значение равно NULL, оператор возвращает 0 (ложь), в противном случае возвращается 1 (истина) (листинг 5.14).

Листинг 5.14. Использование конструкции `IS NOT NULL`

```
SELECT NULL IS NOT NULL, 0 IS NOT NULL;
+------------------+---------------+
| NULL IS NOT NULL | 0 IS NOT NULL |
+------------------+---------------+
|                0 |             1 |
+------------------+---------------+
```

5.5. Выборка из диапазона

Для выборки записей из определенного интервала используется оператор `BETWEEN` *`min`* `AND` *`max`*, который возвращает записи, значения которых лежат в диапазоне от *`min`* до *`max`* (листинг 5.15).

Листинг 5.15. Использование конструкции `BETWEEN`

```
SELECT * FROM catalogs WHERE id_catalog BETWEEN 3 AND 4;
+------------+---------------+
| id_catalog | name          |
+------------+---------------+
|          3 | Видеоадаптеры |
|          4 | Жесткие диски |
+------------+---------------+
```

Как видно из листинга 5.15, в результирующей таблице возвращаются записи в диапазоне от 3 до 4.

Существует конструкция, противоположенная конструкции `BETWEEN` — `NOT BETWEEN`, которая возвращает записи, не попадающие в интервал между *`min`* и *`max`* (листинг 5.16).

Листинг 5.16. Использование конструкции `NOT BETWEEN`

```
SELECT * FROM catalogs WHERE id_catalog NOT BETWEEN 3 AND 4;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры         |
+------------+--------------------+
```

5.6. Выборка множества

Иногда требуется извлечь записи, удовлетворяющие не диапазону, а списку, например, записи с `id_catalog` из списка `(1,2,5)`, как показано в листинге 5.17. Для этого предназначена конструкция `IN`.

Листинг 5.17. Использование оператора `IN`

```
SELECT * FROM catalogs WHERE id_catalog IN (1,2,5);
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры         |
+------------+--------------------+
```

Если проверяемое значение равно `NULL`, результат всегда равен `NULL` (листинг 5.18).

Листинг 5.18. `NULL` в качестве проверяемого значения

```
SELECT NULL IN (0,3,5,'wefwf');
+-------------------------+
| NULL IN (0,3,5,'wefwf') |
+-------------------------+
|                    NULL |
+-------------------------+
```

Для совместимости со стандартом SQL в MySQL, начиная с версии 4.1.0, конструкция `IN` также возвращает `NULL`, если один из элементов списка равен `NULL` (листинг 5.19).

Листинг 5.19. `NULL` в качестве элемента списка

```
SELECT 2 IN (0,NULL,5,'wefwf');
+-------------------------+
| 2 IN (0,NULL,5,'wefwf') |
+-------------------------+
|                    NULL |
+-------------------------+
```

Важно отметить, что в версиях, предшествующих MySQL 4.1.0, в этом случае возвращалось либо значение 0 (ложь), либо 1 (истина).

Конструкция `NOT IN` является противоположной оператору `IN` и возвращает 1 (истина), если проверяемое значение не входит в список, и 0 (ложь), если оно присутствует в списке.

Листинг 5.20. Использование конструкции `NOT IN`

```
SELECT * FROM catalogs WHERE id_catalog NOT IN (1,2,5);
+------------+---------------+
| id_catalog | name          |
+------------+---------------+
|          3 | Видеоадаптеры |
|          4 | Жесткие диски |
+------------+---------------+
```

Помимо представленного в листинге 5.20 синтаксиса конструкции `NOT IN`, существует альтернативная форма записи этого оператора, которая приведена в листинге 5.21.

Листинг 5.21. Альтернативная форма записи конструкции `NOT IN`

```
SELECT * FROM catalogs WHERE NOT id_catalog IN (1,2,5);
+------------+---------------+
| id_catalog | name          |
+------------+---------------+
|          3 | Видеоадаптеры |
|          4 | Жесткие диски |
+------------+---------------+
```

5.7. Использование строк в условии

В конструкции `WHERE` могут использоваться не только числовые столбцы. В листинге 5.22 из таблицы `catalogs` извлекается запись, соответствующая элементу каталога "Процессоры".

Листинг 5.22. Работа с текстовыми полями

```
SELECT * FROM catalogs WHERE name = 'процессоры';
+------------+------------+
| id_catalog | name       |
+------------+------------+
|          1 | Процессоры |
+------------+------------+
```

Как видно из листинга 5.22, сравнение строк в MySQL не зависит от регистра. Начиная с версии 4.1.0, можно выбрать несколько режимов работы

с СУБД MySQL, при которых либо будет различаться регистр строк, либо игнорироваться. Подробнее с настройкой этих режимов можно ознакомиться в *разд. 2.3.3*.

Зачастую условную выборку с участием строк удобнее производить не при помощи оператора равенства `=`, а при помощи оператора `LIKE`, который позволяет использовать простейшие регулярные выражения. Оператор `LIKE` имеет следующий синтаксис:

```
expr LIKE pat
```

Оператор часто используется в конструкции `WHERE` и возвращает 1 (истина), если выражение *`expr`* соответствует выражению *`pat`*, и 0 (ложь) в противном случае (листинг 5.23).

Листинг 5.23. Использование оператора `LIKE`

```
SELECT 'MySQL' LIKE 'MySQL', 'MySQL' LIKE 'mysql';
+----------------------+----------------------+
| 'MySQL' LIKE 'MySQL' | 'MySQL' LIKE 'mysql' |
+----------------------+----------------------+
|                    1 |                    1 |
+----------------------+----------------------+
SELECT 'MySQL' LIKE 'MySQL1', 'MySQL' LIKE BINARY 'mysql';
+-----------------------+-----------------------------+
| 'MySQL' LIKE 'MySQL1' | 'MySQL' LIKE BINARY 'mysql' |
+-----------------------+-----------------------------+
|                     0 |                           0 |
+-----------------------+-----------------------------+
```

Замечание

Важно помнить, что сравнение строк не зависит от регистра, если не используется представитель или ключевое слово `BINARY`, указывающее, что строку следует рассматривать как двоичную последовательность.

Замечание

Передача в качестве любого операнда значения `NULL` приводит к тому, что оператор `LIKE` возвращает `NULL`.

Рассмотренные в листинге 5.23 операции аналогичны оператору `=`. Главное преимущество оператора `LIKE` перед оператором равенства заключается в использовании специальных символов, приведенных в табл. 5.5.

При помощи специальных символов, представленных в табл. 5.5, можно задать различные шаблоны соответствия (листинг 5.24).

Таблица 5.5. *Специальные символы, используемые в операторе* `LIKE`

Символ	Описание
`%`	Соответствует любому количеству символов, даже их отсутствию
`_`	Соответствует ровно одному символу

Листинг 5.24. Использование спецсимвола `%`

```
SELECT 'Программист' LIKE 'Программ%';
+-------------------------------+
| 'Программист' LIKE 'Программ%' |
+-------------------------------+
|                             1 |
+-------------------------------+
SELECT 'Программа' LIKE 'Программ%';
+------------------------------+
| 'Программа' LIKE 'Программ%' |
+------------------------------+
|                            1 |
+------------------------------+
SELECT 'Программ' LIKE 'Программ%';
+-----------------------------+
| 'Программ' LIKE 'Программ%' |
+-----------------------------+
|                           1 |
+-----------------------------+
```

Как видно из листинга 5.24, специальный символ `%` заменяет собой любую последовательность символов, в том числе и пустую строку, поэтому шаблон `'Программ%'` удовлетворяет словам `'Программист'`, `'Программа'` и `'Программ'`, а также любому слову, начинающемуся с выражения `'Программ'`.

Специальный символ `%` может быть размещен в любом месте шаблона, как в начале, так и в середине слова (листинг 5.25).

Листинг 5.25. Использование спецсимвола `%` в начале и середине шаблона

```
SELECT 'Программирование' LIKE 'П%е','Печенье' LIKE 'П%е';
+------------------------------+----------------------+
| 'Программирование' LIKE 'П%е' | 'Печенье' LIKE 'П%е' |
+------------------------------+----------------------+
|                            1 |                    1 |
+------------------------------+----------------------+
```

```
SELECT 'Программирование' LIKE '%ние','Кодирование' LIKE '%ние';
+-------------------------------+--------------------------+
| 'Программирование' LIKE '%ние' | 'Кодирование' LIKE '%ние' |
+-------------------------------+--------------------------+
|                             1 |                        1 |
+-------------------------------+--------------------------+
```

Другой специальный символ _ соответствует одному любому символу. Так, шаблон из трех знаков подчеркивания '___' будет соответствовать любому слову, состоящему из трех символов: 'код', 'рот', 'абв'.

Листинг 5.26. Использование специального символа _

```
SELECT 'код' LIKE '___','рот' LIKE '___', 'абв' LIKE '___';
+------------------+------------------+------------------+
| 'код' LIKE '___' | 'рот' LIKE '___' | 'абв' LIKE '___' |
+------------------+------------------+------------------+
|                1 |                1 |                1 |
+------------------+------------------+------------------+
```

Ограничений на позицию специального символа _, так же как и в случае %, нет — можно располагать его в любой части шаблона. Пример использования символа _ приведен в листингах 5.26 и 5.27.

Листинг 5.27. Использование спецсимвола _ в разных частях шаблона

```
SELECT 'код' LIKE '_од', 'код' LIKE 'к_д', 'код' LIKE 'ко_';
+------------------+------------------+------------------+
| 'код' LIKE '_од' | 'код' LIKE 'к_д' | 'код' LIKE 'ко_' |
+------------------+------------------+------------------+
|                1 |                1 |                1 |
+------------------+------------------+------------------+
```

Для того чтобы поместить в шаблон сами символы % и _ без их специальной интерпретации, необходимо экранировать их при помощи обратного слэша, как это продемонстрировано в листинге 5.28.

Листинг 5.28. Поиск символов % и _

```
SELECT '15 %' LIKE '15 \%', 'my_sql' LIKE 'my\_sql';
+---------------------+-------------------------+
| '15 %' LIKE '15 \%' | 'my_sql' LIKE 'my\_sql' |
+---------------------+-------------------------+
|                   1 |                       1 |
+---------------------+-------------------------+
```

```
SELECT '15' LIKE '15 \%', 'my sql' LIKE 'my\_sql';
+-------------------+------------------------+
| '15' LIKE '15 \%' | 'my sql' LIKE 'my\_sql' |
+-------------------+------------------------+
|                 0 |                      0 |
+-------------------+------------------------+
```

В заключение раздела рассмотрим использование оператора `LIKE` на примере таблицы `catalogs`. В листинге 5.29 приводится пример выборки записей для названий элементов каталога, которые заканчиваются на символ `'ы'`.

Листинг 5.29. Извлечение из таблицы `catalogs` записей, названия которых заканчиваются на `'ы'`

```
SELECT * FROM catalogs WHERE name LIKE '%ы';
+------------+-------------------+
| id_catalog | name              |
+------------+-------------------+
|          3 | Видеоадаптеры     |
|          2 | Материнские платы |
|          1 | Процессоры        |
+------------+-------------------+
```

Оператор `NOT LIKE`, показанный в листинге 5.30, противоположен по действию оператору `LIKE` и имеет следующий синтаксис:

expr `NOT LIKE` *pat*

Оператор возвращает 0, если выражение *expr* соответствует выражению *pat*, и 1 в противном случае, т. е. с его помощью можно извлечь записи, которые не удовлетворяют указанному условию.

Замечание

Оператор `NOT LIKE` можно представить в виде `NOT (`*expr* `LIKE` *pat*`)`.

Листинг 5.30. Использование оператора `NOT LIKE`

```
SELECT * FROM catalogs WHERE name NOT LIKE '%ы';
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
```

5.8. Использование календарных значений в условии

Очень часто условия связаны с календарными столбцами. Создадим таблицу `catalogs`, состоящую из трех полей:

- `id_catalog` — первичный ключ;
- `name` — название элемента каталога;
- `putdate` — время последней модификации каталога.

В листинге 5.31 представлен дамп для разворачивания таблицы `catalogs`.

Листинг 5.31. Создание таблицы `catalogs` с календарным столбцом

```
CREATE TABLE catalogs (
  id_catalog int(11) NOT NULL,
  name tinytext NOT NULL,
  putdate datetime NOT NULL,
  PRIMARY KEY  (id_catalog)
);
INSERT INTO catalogs VALUES (3, 'Видеоадаптеры', '2007-01-10 16:48:05'),
                    (4, 'Жесткие диски', '2007-01-05 20:01:58'),
                    (2, 'Материнские платы', '2006-12-28 18:32:41'),
                    (5, 'Оперативная память', '2006-12-20 10:00:13'),
                    (1, 'Процессоры', '2007-01-10 12:47:00');
```

В листинге 5.32 извлекаются записи, соответствующие записям, которые подверглись модификации в январе 2007 года.

Листинг 5.32. Извлечение записей за январь 2007 года

```
SELECT * FROM catalogs
WHERE putdate >= '2007-01-01' AND putdate < '2007-01-31';
+------------+---------------+---------------------+
| id_catalog | name          | putdate             |
+------------+---------------+---------------------+
|          3 | Видеоадаптеры | 2007-01-10 16:48:05 |
|          4 | Жесткие диски | 2007-01-05 20:01:58 |
|          1 | Процессоры    | 2007-01-10 12:47:00 |
+------------+---------------+---------------------+
```

Точно так же, как и в случае числовых данных, допускается использование операторов `BETWEEN` и `NOT BETWEEN` (листинг 5.33).

Листинг 5.33. Использование оператора BETWEEN

```
SELECT * FROM catalogs
WHERE putdate BETWEEN '2007-01-01' AND '2007-01-31';
+------------+---------------+---------------------+
| id_catalog | name          | putdate             |
+------------+---------------+---------------------+
|          3 | Видеоадаптеры | 2007-01-10 16:48:05 |
|          4 | Жесткие диски | 2007-01-05 20:01:58 |
|          1 | Процессоры    | 2007-01-10 12:47:00 |
+------------+---------------+---------------------+
```

Календарные данные хранятся как строковые значения, поэтому применительно к ним можно использовать оператор LIKE. В листинге 5.34 при помощи оператора LIKE из таблицы catalogs извлекаются записи, которые подверглись модификации в декабре 2006 года.

Листинг 5.34. Извлечение записей за декабрь 2006 года

```
SELECT * FROM catalogs WHERE putdate LIKE '2006-12%';
+------------+--------------------+---------------------+
| id_catalog | name               | putdate             |
+------------+--------------------+---------------------+
|          2 | Материнские платы  | 2006-12-28 18:32:41 |
|          5 | Оперативная память | 2006-12-20 10:00:13 |
+------------+--------------------+---------------------+
```

Помимо оператора LIKE, СУБД MySQL предоставляет широкий спектр функций, позволяющих извлекать различные фрагменты календарного значения. Например, задача извлечения записей из таблицы catalogs за декабрь 2006 года может решаться альтернативным запросом, представленным в листинге 5.35.

Замечание

Более подробно функции MySQL рассматриваются в *главе 6*.

Листинг 5.35. Альтернативный способ извлечения записей за декабрь 2006 года

```
SELECT * FROM catalogs
WHERE YEAR(putdate) = 2006 AND MONTH(putdate) = 12;
```

```
+------------+--------------------+---------------------+
| id_catalog | name               | putdate             |
+------------+--------------------+---------------------+
|          2 | Материнские платы  | 2006-12-28 18:32:41 |
|          5 | Оперативная память | 2006-12-20 10:00:13 |
+------------+--------------------+---------------------+
```

В листинге 5.35 выбираются только те даты, для которых функция `YEAR()` возвращает 2006 год, а функция `MONTH()` возвращает 12-й месяц.

5.9. Форматирование календарных значений

Календарные функции могут использоваться не только в условных выражениях, но также и списке столбцов, после ключевого слова `SELECT`. В листинге 5.36 при помощи функции `DATE()` от поля `putdate` отсекается время суток, таким образом, что в результирующей таблице остается только дата.

Листинг 5.36. Использование функции `DATE()`

```
SELECT id_catalog, name, DATE(putdate) FROM catalogs;
+------------+--------------------+---------------+
| id_catalog | name               | DATE(putdate) |
+------------+--------------------+---------------+
|          3 | Видеоадаптеры      | 2007-01-10    |
|          4 | Жесткие диски      | 2007-01-05    |
|          2 | Материнские платы  | 2006-12-28    |
|          5 | Оперативная память | 2006-12-20    |
|          1 | Процессоры         | 2007-01-10    |
+------------+--------------------+---------------+
```

Наиболее удобной для форматирования является функция `DATE_FORMAT(`*`date`*`, `*`format`*`)`, которая форматирует время *`date`* в соответствии со строкой *`format`*. В строке *`format`* могут использоваться определители, представленные в табл. 5.6.

Таблица 5.6. *Определители строки форматирования функции `DATE_FORMAT()`*

Определитель	Описание
`%a`	Сокращенное наименование дня недели (Sun, ..., Sat)
`%b`	Сокращенное наименование месяца (Jan, ..., Dec)
`%c`	Месяц в числовой форме (1, ..., 12)
`%D`	День месяца с английским суффиксом (1st, 2nd, 3rd и т. д.)

Таблица 5.6 *(окончание)*

Определитель	Описание
`%d`	День месяца в числовой форме с ведущим нулем (01, ..., 31)
`%e`	День месяца в числовой форме (1, ..., 31)
`%f`	Микросекунды (000000, ..., 999999)
`%H`	Час с ведущим нулем (00, ..., 23)
`%h`	Час с ведущим нулем (01, ..., 12)
`%I`	Час с ведущим нулем (01, ..., 12)
`%i`	Минуты с ведущим нулем (00, ...,59)
`%j`	День года (001, ..., 366)
`%k`	Час с ведущим нулем (0, ..., 23)
`%l`	Час без ведущего нуля (1, ..., 12)
`%M`	Название месяца (January, ..., December)
`%m`	Месяц в числовой форме с ведущим нулем (01, ..., 12)
`%p`	AM или PM (для 12-часового формата)
`%r`	Время, 12-часовой формат (`hh:mm:ss AM` или `hh:mm:ss PM`)
`%S`	Секунды (00, ..., 59)
`%s`	Секунды (00, ..., 59)
`%T`	Время, 24-часовой формат (`hh:mm:ss`)
`%U`	Неделя (00, ..., 52), где воскресенье считается первым днем недели
`%u`	Неделя (00, ..., 52), где понедельник считается первым днем недели
`%V`	Неделя (01, ..., 53), где воскресенье считается первым днем недели. Используется с `%X`
`%v`	Неделя (01, ..., 53), где понедельник считается первым днем недели. Используется с `%x`
`%W`	Название дня недели (Sunday, ..., Saturday)
`%w`	День недели (0, ..., 6), где 0 — воскресенье, ..., 6 — суббота
`%X`	Год для недели, где воскресенье считается первым днем недели, 4 разряда, используется с `%V`
`%x`	Год для недели, где воскресенье считается первым днем недели, число, 4 разряда, используется с `%v`
`%Y`	Год, 4 разряда (`YYYY`)
`%y`	Год, 2 разряда (`YY`)
`%%`	Литерал `%`

Все другие символы, которые не указаны в табл. 5.6, выводятся без изменений. Пример использования функции DATE_FORMAT() представлен в листинге 5.37.

Листинг 5.37. Использование функции DATE_FORMAT()

```
SELECT DATE_FORMAT('2007-01-01 01:59:59','На дворе %Y год');
+-----------------------------------------------------+
| DATE_FORMAT('2007-01-01 01:59:59','На дворе %Y год') |
+-----------------------------------------------------+
| На дворе 2007 год                                   |
+-----------------------------------------------------+
```

В СУБД MySQL по умолчанию в представлении даты сначала следует год, затем месяц и день, т. к. в таком порядке удобнее сортировать результат. Это не всегда является удобным, поскольку часто требуется вначале разместить число, затем месяц и лишь потом год. Именно для таких задач и предназначена функция DATE_FORMAT() (листинг 5.38).

Листинг 5.38. Преобразование даты из формата YYYY-MM-DD в DD.MM.YYYY

```
SELECT id_catalog, name, DATE_FORMAT(putdate, '%d.%m.%Y') FROM catalogs;
+------------+--------------------+----------------------------------+
| id_catalog | name               | DATE_FORMAT(putdate, '%d.%m.%Y') |
+------------+--------------------+----------------------------------+
|          3 | Видеоадаптеры      | 10.01.2007                       |
|          4 | Жесткие диски      | 05.01.2007                       |
|          2 | Материнские платы  | 28.12.2006                       |
|          5 | Оперативная память | 20.12.2006                       |
|          1 | Процессоры         | 10.01.2007                       |
+------------+--------------------+----------------------------------+
```

5.10. Модификация календарных значений

Как было показано в *разд. 4.1.3*, календарные значения можно складывать и вычитать из них временны́е интервалы. В листинге 5.39 приводится пример выборки из таблицы catalogs, в которой каждое значение putdate уваливается на три месяца.

Замечание

Вместо ключевого слова MONTH, обозначающего месяц, могут использоваться другие ключевые слова для других временны́х интервалов. Полный их список приводится в табл. 4.1.

Листинг 5.39. Увеличение даты на три месяца

```
SELECT id_catalog, name, putdate + INTERVAL 3 MONTH FROM catalogs;
+------------+--------------------+---------------------------+
| id_catalog | name               | putdate + INTERVAL 3 MONTH |
+------------+--------------------+---------------------------+
|          3 | Видеоадаптеры      | 2007-04-10 16:48:05        |
|          4 | Жесткие диски      | 2007-04-05 20:01:58        |
|          2 | Материнские платы  | 2007-03-28 18:32:41        |
|          5 | Оперативная память | 2007-03-20 10:00:13        |
|          1 | Процессоры         | 2007-04-10 12:47:00        |
+------------+--------------------+---------------------------+
```

Вместо константы в выражении `INTERVAL` может использоваться столбец. В листинге 5.40 приводится пример выборки, где значение `putdate` уменьшается на количество дней, равное значению `id_catalog`.

Листинг 5.40. Использование столбца `id_catalog` в выражении `INTERVAL`

```
SELECT id_catalog, name, putdate - INTERVAL id_catalog DAY FROM catalogs;
+------------+--------------------+-----------------------------------+
| id_catalog | name               | putdate - INTERVAL id_catalog DAY |
+------------+--------------------+-----------------------------------+
|          3 | Видеоадаптеры      | 2007-01-07 16:48:05               |
|          4 | Жесткие диски      | 2007-01-01 20:01:58               |
|          2 | Материнские платы  | 2006-12-26 18:32:41               |
|          5 | Оперативная память | 2006-12-15 10:00:13               |
|          1 | Процессоры         | 2007-01-09 12:47:00               |
+------------+--------------------+-----------------------------------+
```

5.11. Псевдонимы столбцов

Как видно из листингов 5.38—5.40, имена вычисляемых столбцов (формируемых выражениями или функциями) имеют длинные имена. Это часто не удобно, особенно в прикладных программах, где имя столбца используется для доступа к элементам в результирующей таблице. В `SELECT`-запросе столбцу можно назначить новое имя, для этого предназначен оператор `AS`. В листинге 5.41 результату функции `DATE_FORMAT()` присваивается новый псевдоним `printdate`.

Листинг 5.41. Использование оператора `AS`

```
SELECT id_catalog, name, DATE_FORMAT(putdate, '%d.%m.%Y') AS printdate
FROM catalogs;
```

```
+------------+--------------------+------------+
| id_catalog | name               | printdate  |
+------------+--------------------+------------+
|          3 | Видеоадаптеры      | 10.01.2007 |
|          4 | Жесткие диски      | 05.01.2007 |
|          2 | Материнские платы  | 28.12.2006 |
|          5 | Оперативная память | 20.12.2006 |
|          1 | Процессоры         | 10.01.2007 |
+------------+--------------------+------------+
```

5.12. Сортировка записей

Как видно из предыдущих листингов данной главы, результат выборки представляет собой записи, которые располагаются в порядке, в котором они хранятся в базе данных. Однако часто требуется отсортировать значения по одному из столбцов. Это осуществляется при помощи конструкции `ORDER BY`, которая следует за выражением `SELECT`. После конструкции `ORDER BY` указывается столбец (или столбцы), по которому следует сортировать данные.

Замечание

Отсортировать данные по столбцу можно на физическом уровне при помощи оператора `ALTER ... ORDER BY ...` *(см. разд. 2.5.5)*.

Как видно из листинга 5.42, первый запрос сортирует результат выборки по полю `id_catalog`, а второй — по полю `name`.

Листинг 5.42. Использование конструкции `ORDER BY`

```
SELECT id_catalog, name FROM catalogs ORDER BY id_catalog;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
SELECT id_catalog, name FROM catalogs ORDER BY name;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
```

```
|           2 | Материнские платы  |
|           5 | Оперативная память |
|           1 | Процессоры         |
+------------+--------------------+
```

По умолчанию сортировка производится в прямом порядке, однако, добавив после имени столбца ключевое слово `DESC`, можно добиться сортировки в обратном порядке (листинг 5.43).

Листинг 5.43. Обратная сортировка

```
SELECT id_catalog, name FROM catalogs ORDER BY id_catalog DESC;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|           5 | Оперативная память |
|           4 | Жесткие диски      |
|           3 | Видеоадаптеры      |
|           2 | Материнские платы  |
|           1 | Процессоры         |
+------------+--------------------+
```

Сортировку записей можно производить и по нескольким столбцам. Пусть имеется таблица `tbl`, состоящая из двух столбцов: `id_catalog` и `putdate`. В таблицу помещается запись при обращении к каталогу с первичным ключом `id_catalog`, при этом в поле `putdate` помещается дата обращения. В листинге 5.44 приводится дамп, позволяющий развернуть таблицу `tbl`.

Листинг 5.44. Таблица `tbl`

```
CREATE TABLE tbl (
  id_catalog int(11) NOT NULL,
  putdate datetime NOT NULL
);
INSERT INTO tbl VALUES (5, '2007-01-04 05:01:58');
INSERT INTO tbl VALUES (3, '2007-01-03 12:10:45');
INSERT INTO tbl VALUES (4, '2007-01-10 16:10:25');
INSERT INTO tbl VALUES (1, '2006-12-20 08:34:09');
INSERT INTO tbl VALUES (2, '2007-01-06 20:57:42');
INSERT INTO tbl VALUES (2, '2006-12-24 18:42:41');
INSERT INTO tbl VALUES (5, '2006-12-25 09:35:01');
INSERT INTO tbl VALUES (1, '2006-12-23 15:14:26');
INSERT INTO tbl VALUES (4, '2006-12-26 21:32:00');
INSERT INTO tbl VALUES (3, '2006-12-25 12:11:10');
```

Для того чтобы отсортировать таблицу `tbl` сначала по полю `id_catalog`, а затем по полю `putdate`, можно воспользоваться запросом, представленным в листинге 5.45.

Листинг 5.45. Сортировка таблицы по двум столбцам

```
SELECT * FROM tbl ORDER BY id_catalog, putdate DESC;
+------------+---------------------+
| id_catalog | putdate             |
+------------+---------------------+
|          1 | 2006-12-23 15:14:26 |
|          1 | 2006-12-20 08:34:09 |
|          2 | 2007-01-06 20:57:42 |
|          2 | 2006-12-24 18:42:41 |
|          3 | 2007-01-03 12:10:45 |
|          3 | 2006-12-25 12:11:10 |
|          4 | 2007-01-10 16:10:25 |
|          4 | 2006-12-26 21:32:00 |
|          5 | 2007-01-04 05:01:58 |
|          5 | 2006-12-25 09:35:01 |
+------------+---------------------+
```

Как видно из листинга 5.45, записи в таблице `tbl` сначала сортируются по столбцу `id_catalog`, а совпадающие в рамках одного значения `id_catalog` записи сортируются по полю `putdate` в обратном порядке. Следует отметить, что ключевое слово `DESC` относится только к полю `putdate`. Для того чтобы отсортировать оба столбца в обратном порядке, потребуется снабдить ключевым словом как столбец `id_catalog`, так и `putdate` (листинг 5.46).

Листинг 5.46. Обратная сортировка по двум столбцам

```
SELECT * FROM tbl ORDER BY id_catalog DESC, putdate DESC;
+------------+---------------------+
| id_catalog | putdate             |
+------------+---------------------+
|          5 | 2007-01-04 05:01:58 |
|          5 | 2006-12-25 09:35:01 |
|          4 | 2007-01-10 16:10:25 |
|          4 | 2006-12-26 21:32:00 |
|          3 | 2007-01-03 12:10:45 |
|          3 | 2006-12-25 12:11:10 |
|          2 | 2007-01-06 20:57:42 |
|          2 | 2006-12-24 18:42:41 |
|          1 | 2006-12-23 15:14:26 |
|          1 | 2006-12-20 08:34:09 |
+------------+---------------------+
```

Для прямой сортировки также существует ключевое слово ASC (в противовес ключевому слову DESC), но поскольку по умолчанию записи сортируются в прямом порядке, данное ключевое слово часто опускают.

5.13. Вывод записей в случайном порядке

Для вывода записей в случайном порядке используется конструкция ORDER BY RAND(). В листинге 5.47 демонстрируется вывод содержимого таблицы catalogs (см. листинг 5.1) в случайном порядке.

Листинг 5.47. Вывод содержимого таблицы в случайном порядке

```
SELECT id_catalog, name FROM catalogs ORDER BY RAND();
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          4 | Жесткие диски      |
|          5 | Оперативная память |
|          2 | Материнские платы  |
|          1 | Процессоры         |
|          3 | Видеоадаптеры      |
+------------+--------------------+
```

Если требуется вывести лишь одну случайную запись, используется конструкция LIMIT 1 (листинг 5.48).

Листинг 5.48. Вывод одной случайной записи

```
SELECT id_catalog, name FROM catalogs ORDER BY RAND() LIMIT 1;
+------------+-------------------+
| id_catalog | name              |
+------------+-------------------+
|          2 | Материнские платы |
+------------+-------------------+
```

5.14. Ограничение выборки

Результат выборки может содержать сотни и тысячи записей. Их вывод и обработка занимают значительное время и серьезно загружают сервер базы данных. Поэтому информацию часто разбивают на страницы и предоставляют ее пользователю порциями. Извлечение только части запроса требует меньше времени и вычислений, кроме того, пользователю часто бывает достаточно просмотреть первые несколько записей. Постраничная навигация

используется при помощи ключевого слова `LIMIT`, за которым следует число выводимых записей. В листинге 5.49 извлекаются первые две записи таблицы `catalogs`, при этом одновременно осуществляется обратная сортировка по полю `id_catalog`.

Листинг 5.49. Использование ключевого слова `LIMIT`

```
SELECT id_catalog, name FROM catalogs
ORDER BY id_catalog DESC
LIMIT 2;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          5 | Оперативная память |
|          4 | Жесткие диски      |
+------------+--------------------+
```

Для того чтобы извлечь следующие две записи, используется ключевое слово `LIMIT` с двумя числами, первое указывает позицию, начиная с которой необходимо вернуть результат, а второе число — это количество извлекаемых записей (листинг 5.50).

Листинг 5.50. Извлечение записей, начиная со второй позиции

```
SELECT id_catalog, name FROM catalogs
ORDER BY id_catalog DESC
LIMIT 2, 2;
+------------+-------------------+
| id_catalog | name              |
+------------+-------------------+
|          3 | Видеоадаптеры     |
|          2 | Материнские платы |
+------------+-------------------+
```

Для извлечения следующих двух записей необходимо использовать конструкцию `LIMIT 4, 2`.

5.15. Текущая версия MySQL

При получении выборки из базы данных часто требуется выполнять специфические задачи, для решения которых удобно воспользоваться встроенными функциями MySQL. Каждая функция имеет уникальное имя и может иметь несколько аргументов (в том числе и ни одного), которые перечисляются через запятую в круглых скобках вслед за названием. Если аргументы у функ-

ции отсутствуют, круглые скобки все равно следует указывать, например, `NOW()`. Отличительной чертой MySQL является то, что при использовании функций пробелы между именем функции и круглыми скобками недопустимы, т. е. написание `NOW()` правильное, а `NOW ()` уже нет. Результат функции подставляется в место вызова функции.

Замечание

Более подробно функции MySQL рассматриваются в *главе 6*.

В листинге 5.51 приведен пример использования функции `VERSION()`, которая возвращает версию сервера MySQL.

Листинг 5.51. Использование функции `VERSION()`

```
SELECT VERSION();
+-----------------+
| VERSION()       |
+-----------------+
| 5.0.18-standard |
+-----------------+
```

Как видно из листинга 5.51, для использования функции `VERSION()` не требуется применение ключевого слова `FROM`, т. к. параметры таблицы не нужны. Однако использование его не возбраняется (листинг 5.52).

Листинг 5.52. Альтернативное использование функции `VERSION()`

```
SELECT VERSION() FROM catalogs;
+-----------------+
| VERSION()       |
+-----------------+
| 5.0.18-standard |
| 5.0.18-standard |
| 5.0.18-standard |
| 5.0.18-standard |
| 5.0.18-standard |
+-----------------+
```

В этом случае результат функции `VERSION()` выступает в качестве строковой константы при выборке из таблицы `catalogs`.

Для того чтобы исключить повторение результата по числу строк в таблице, в MySQL используется таблица `DUAL`, на самом деле не существующая, но которая может быть использована в предложении `FROM` (листинг 5.53).

Замечание

Таблица DUAL введена в MySQL, начиная с версии 4.1.

Листинг 5.53. Использование псевдотаблицы DUAL

```
SELECT VERSION(), 2+2 FROM DUAL;
+-----------------+-----+
| VERSION()       | 2+2 |
+-----------------+-----+
| 5.0.18-standard |   4 |
+-----------------+-----+
```

5.16. Количество записей в таблице

Количество записей в таблице можно узнать при помощи функции COUNT(), которая принимает в качестве аргумента имя столбца. Функция возвращает число строк в таблице, значения столбца для которых отличны от NULL (листинг 5.54).

Листинг 5.54. Использование функции COUNT()

```
SELECT COUNT(id_catalog) FROM catalogs;
+-------------------+
| COUNT(id_catalog) |
+-------------------+
|                 5 |
+-------------------+
```

В качестве параметра функции наряду с именами столбцов может выступать символ звездочки (*). При использовании символа * будет возвращено число строк таблицы, независимо от того, принимают какие-то из них значение NULL или нет. Пример использования функции COUNT(*) приводится в листинге 5.55.

Листинг 5.55. Использование функции COUNT(*)

```
SELECT COUNT(*) FROM catalogs;
+----------+
| COUNT(*) |
+----------+
|        5 |
+----------+
```

5.17. Поиск минимального и максимального значений

В качестве дополнительного примера можно привести функции MIN() и MAX(), возвращающие минимальное и максимальное значения столбца, имя которого передано в качестве параметра (листинг 5.56).

Листинг 5.56. Использование функций MIN() и MAX()

```
SELECT MIN(id_catalog) AS min,
MAX(id_catalog) AS max FROM catalogs;
+------+------+
| min  | max  |
+------+------+
|    1 |    5 |
+------+------+
```

Однако использование функций MIN() и MAX() в выражении WHERE приведет к ошибке. В листинге 5.57 показана попытка извлечения записи из таблицы catalogs с максимальным значением поля id_catalog.

Листинг 5.57. Использование функций MIN() и MAX() в WHERE недопустимо

```
SELECT * FROM catalogs WHERE id_catalog = MAX(id_catalog);
ERROR 1111: Invalid use of group function
```

Решение поставленной выше задачи следует искать с привлечением выражения ORDER BY. В листинге 5.58 первый запрос извлекает запись с наименьшим значением поля id_catalog, а второй — с наибольшим.

Листинг 5.58. Использование функций MIN() и MAX() совместно с ORDER BY

```
SELECT * FROM catalogs ORDER BY id_catalog LIMIT 1;
+------------+------------+
| id_catalog | name       |
+------------+------------+
|          1 | Процессоры |
+------------+------------+
SELECT * FROM catalogs ORDER BY id_catalog DESC LIMIT 1;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          5 | Оперативная память |
+------------+--------------------+
```

5.18. Вывод уникальных значений

Очень часто встает задача вывода уникальных значений из таблицы. Воспользуемся таблицей `tbl` из листинга 5.44. Пусть требуется вывести все значения поля `id_catalog`, для чего можно воспользоваться запросом, представленным в листинге 5.59.

Листинг 5.59. Выборка значений поля `id_catalog` из таблицы `tbl`

```
SELECT id_catalog FROM tbl ORDER BY id_catalog;
+------------+
| id_catalog |
+------------+
|          1 |
|          1 |
|          2 |
|          2 |
|          3 |
|          3 |
|          4 |
|          4 |
|          5 |
|          5 |
+------------+
```

Как видно из листинга 5.59, результат не совсем удобен для восприятия. Было бы лучше, если бы запрос вернул уникальные значения столбца `id_catalog`. Для этого перед именем столбца можно использовать ключевое слово `DISTINCT`, которое предписывает MySQL извлекать только уникальные значения (листинг 5.60).

Замечание

Ключевое слово `DISTINCT` имеет синоним — `DISTINCTROW`.

Листинг 5.60. Выборка уникальных значений

```
SELECT DISTINCT id_catalog FROM tbl ORDER BY id_catalog;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
```

```
|           4 |
|           5 |
+------------+
```

Как показано в листинге 5.60, результат запроса не содержит ни одного повторяющегося значения. Использование ключевого слова DISTINCT допускается совместно с функцией COUNT(). В листинге 5.61 первый запрос возвращает общее число записей в таблице tbl, а второй — число уникальных значений id_catalog.

Листинг 5.61. Совместное использование ключевого слова DISTINCT и функции COUNT()

```
SELECT COUNT(id_catalog) FROM tbl;
+-------------------+
| COUNT(id_catalog) |
+-------------------+
|                10 |
+-------------------+
SELECT COUNT(DISTINCT id_catalog) FROM tbl;
+----------------------------+
| COUNT(DISTINCT id_catalog) |
+----------------------------+
|                          5 |
+----------------------------+
```

Для ключевого слова DISTINCT имеется противоположенное слово ALL, которое предписывает извлечение всех значений столбца, в том числе и повторяющихся. Поскольку такое поведение установлено по умолчанию, ключевое слово ALL часто опускают.

Для извлечения уникальных записей чаще прибегают к конструкции GROUP BY, за ней указывается имя столбца, по которому группируется результат (листинг 5.62).

Замечание

Конструкция GROUP BY располагается в SELECT-запросе перед конструкциями ORDER BY и LIMIT.

Листинг 5.62. Использование конструкции GROUP BY

```
SELECT id_catalog FROM tbl
GROUP BY id_catalog ORDER BY id_catalog;
```

```
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
```

Однако в отличие от ключевого слова `DISTINCT`, использование функции `COUNT()` совместно с `GROUP BY` приводит не к подсчету уникальных значений `id_catalog`, а к выводу числа записей, соответствующих каждому из уникальных значений `id_catalog` (листинг 5.63).

Листинг 5.63. Совместное использование `GROUP BY` и функции `COUNT()`

```
SELECT id_catalog, COUNT(id_catalog) FROM tbl
GROUP BY id_catalog ORDER BY id_catalog;
+------------+-------------------+
| id_catalog | COUNT(id_catalog) |
+------------+-------------------+
|          1 |                 2 |
|          2 |                 2 |
|          3 |                 2 |
|          4 |                 2 |
|          5 |                 2 |
+------------+-------------------+
```

Замечание

Помимо функции `COUNT()`, в выражениях с использованием конструкции `GROUP BY` может применяться целый класс функций, описанию которых посвящена *глава 8*.

При использовании конструкции `GROUP BY` точно так же возможно применение условия `WHERE` (листинг 5.64).

Листинг 5.64. Использование ключевого слова `WHERE` совместно с `GROUP BY`

```
SELECT id_catalog, COUNT(id_catalog) FROM tbl
WHERE id_catalog > 2
GROUP BY id_catalog
ORDER BY id_catalog;
```

```
+------------+-------------------+
| id_catalog | COUNT(id_catalog) |
+------------+-------------------+
|          3 |                 2|
|          4 |                 2|
|          5 |                 2|
+------------+-------------------+
```

5.19. Групповые условия. Условие *HAVING*

Чаще при составлении условий требуется ограничить выборку по результату функции, например, выбрать группы, где количество записей больше или равно двум. Использование для этих целей конструкции `WHERE` приводит к ошибке (листинг 5.65).

Листинг 5.65. Ошибочное использование конструкции `WHERE`

```
SELECT id_catalog, COUNT(id_catalog) AS total FROM tbl
WHERE total >= 2
GROUP BY id_catalog
ORDER BY id_catalog;
ERROR 1054 (42S22): Unknown column 'total' in 'where clause'
```

Для решения этой проблемы вместо ключевого слова `WHERE` используется ключевое слово `HAVING`, которое располагается вслед за конструкцией `GROUP BY` (листинг 5.66).

Листинг 5.66. Использование ключевого слова `GROUP BY`

```
SELECT id_catalog, COUNT(id_catalog) AS total FROM tbl
GROUP BY id_catalog
HAVING total >= 2
ORDER BY id_catalog;
+------------+-------+
| id_catalog | total |
+------------+-------+
|          1 |     2 |
|          2 |     2 |
|          3 |     2 |
|          4 |     2 |
|          5 |     2 |
+------------+-------+
```

В условии HAVING можно использовать все столбцы результирующей таблицы, не только вычисляемые. Например, в листинге 5.67 приводится пример запроса, извлекающего уникальные значения столбца id_catalog, большие двух.

Листинг 5.67. Использование в конструкции HAVING невычисляемых столбцов

```
SELECT id_catalog, COUNT(id_catalog) FROM products
GROUP BY id_catalog
HAVING id_catalog > 2
ORDER BY id_catalog;
+------------+-------------------+
| id_catalog | COUNT(id_catalog) |
+------------+-------------------+
|          3 |                 4 |
|          4 |                 5 |
|          5 |                 6 |
+------------+-------------------+
```

Отличие запросов, представленных в листингах 5.64 и 5.67, заключается в том, что в случае использования ключевого слова WHERE сначала производится выборка из таблицы с применением условия и лишь затем группировка результата, а в случае использования ключевого слова HAVING сначала происходит группировка таблицы и лишь затем выборка с применением условия.

Допускается использование условия HAVING без группировки GROUP BY (листинг 5.68).

Листинг 5.68. Использование ключевого слова HAVING без GROUP BY

```
SELECT id_catalog FROM tbl
HAVING id_catalog > 2
ORDER BY id_catalog;
+------------+
| id_catalog |
+------------+
|          3 |
|          3 |
|          4 |
|          4 |
|          5 |
|          5 |
+------------+
```

В этом случае каждая строка таблицы рассматривается как отдельная группа.

5.20. Объединение таблиц

Как было сказано ранее, оператор SELECT возвращает результат в виде таблицы. Если формат результирующих таблиц (число, порядок следования и тип столбцов) совпадает, то возможно объединение результатов выполнения двух операторов SELECT в одну результирующую таблицу. Это достигается с использованием оператора UNION. Пусть имеются два SELECT-запроса, представленные в листинге 5.69.

Листинг 5.69. Одиночные SELECT-запросы

```
SELECT id_catalog FROM catalogs;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
SELECT id + 5 FROM tbl;
+--------------+
| id_order + 5 |
+--------------+
|            6 |
|            7 |
|            8 |
|            9 |
|           10 |
+--------------+
```

Объединить результаты из этих двух таблиц можно, соединив два запроса SELECT при помощи ключевого слова UNION, как это продемонстрировано в листинге 5.70.

Листинг 5.70. Использование ключевого слова UNION

```
SELECT id_catalog FROM catalogs
UNION
SELECT id + 5 FROM tbl;
+------------+
| id_catalog |
+------------+
|          1 |
```

```
|          2 |
|          3 |
|          4 |
|          5 |
|          6 |
|          7 |
|          8 |
|          9 |
|         10 |
+------------+
```

Во втором запросе `SELECT`, производящем выборку из таблицы `tbl`, к значению первичного ключа `id` добавляется значение 5, и, таким образом, все значения в результирующей таблице становятся уникальными. Однако если результирующая таблица содержит повторяющиеся строки, СУБД MySQL автоматически отбрасывает дубликаты (листинг 5.71).

Листинг 5.71. Исключение дублирующих записей при использовании конструкции `UNION`

```
SELECT id_catalog FROM catalogs
UNION
SELECT id FROM tbl;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
```

Как видно из листинга 5.71, вместо десяти записей выводится только пять. Изменить поведение по умолчанию можно при помощи ключевого слова `ALL`, которое добавляется после оператора `UNION`. Использование `UNION ALL` требует, чтобы возвращались все строки из обеих результирующих таблиц (листинг 5.72).

Листинг 5.72. Использование ключевого слова `UNION ALL`

```
SELECT id_catalog FROM catalogs
UNION ALL
SELECT id FROM tbl;
```

```
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
```

Для формы UNION ALL существует противоположенный оператор UNION DISTINCT, но т. к. оператор UNION по умолчанию отбрасывает неуникальные значения, ключевое слово DISTINCT часто опускают.

Использование конструкции ORDER BY имеет смысл только после второго SELECT-запроса, в первом SELECT-запросе конструкция ORDER BY приводит к ошибке (листинг 5.73).

Листинг 5.73. Ошибочное использование конструкции ORDER BY

```
SELECT id_catalog FROM catalogs ORDER BY id_catalog
UNION ALL
SELECT id FROM tbl;
ERROR 1221: Incorrect usage of UNION and ORDER BY
```

Это связано с тем, что сортировка производится только для объединенной таблицы, и значение имеет только последняя конструкция ORDER BY (листинг 5.74).

Листинг 5.74. Сортировка результирующей таблицы

```
SELECT id_catalog FROM catalogs
UNION ALL
SELECT id FROM tbl
ORDER BY id_catalog;
+------------+
| id_catalog |
+------------+
|          1 |
|          1 |
```

```
|          2 |
|          2 |
|          3 |
|          3 |
|          4 |
|          4 |
|          5 |
|          5 |
+------------+
```

Следует помнить, что столбцы в результирующей таблице именуются по столбцам первого SELECT-запроса. Как видно из листинга 5.74, объединению подвергаются данные из двух совершенно разных таблиц: catalogs и tbl. Однако обращение к столбцу id_order приводит к ошибке (листинг 5.75) — столбца с таким именем в таблице просто нет, MySQL ориентируется на столбцы первого SELECT-запроса.

Листинг 5.75. Ошибочное обращение к столбцу id_order в конструкции ORDER BY

```
SELECT id_catalog FROM catalogs
UNION ALL
SELECT id FROM tbl
ORDER BY id_order;
ERROR 1054: Unknown column 'id_order' in 'order clause'
```

В данном разделе рассматривались в основном UNION-запросы, объединяющие два SELECT-запроса. Однако это вовсе не значит, что оператор UNION не может объединять большее количество запросов.

5.21. Перекрестное объединение таблиц

Для обсуждения многотабличных запросов создадим две таблицы: tbl1 и tbl2, каждая из которых будет содержать по два столбца: числовой и текстовый. В листинге 5.76 представлены операторы CREATE TABLE, создающие указанные таблицы.

Листинг 5.76. Создание таблиц tbl1 и tbl2

```
CREATE TABLE tbl1 (
  id int(11) NOT NULL default '0',
  name tinytext NOT NULL
) ENGINE=MyISAM;
```

```
INSERT INTO tbl1 VALUES (1, 'А');
INSERT INTO tbl1 VALUES (2, 'Б');
INSERT INTO tbl1 VALUES (3, 'В');
CREATE TABLE tbl2 (
  id int(11) NOT NULL default '0',
  letter tinytext NOT NULL
) ENGINE=MyISAM;
INSERT INTO tbl2 VALUES (2, 'Г');
INSERT INTO tbl2 VALUES (3, 'Д');
INSERT INTO tbl2 VALUES (4, 'Е');
```

После создания таблиц заполним каждую таблицу тремя записями, состоящими из целого числа и соответствующей ему буквы русского алфавита.

Для осуществления запроса сразу к нескольким таблицам их имена перечисляются после ключевого слова `FROM` через запятую. В листинге 5.77 представлена выборка из каждой таблицы, а также двухтабличный запрос с использованием оператора `SELECT` к таблицам `tbl1`, `tbl2`.

Листинг 5.77. Обычные и двухтабличный `SELECT`-запросы к таблицам `tbl1` и `tbl2`

```
SELECT * FROM tbl1;
+----+------+
| id | name |
+----+------+
|  1 | А    |
|  2 | Б    |
|  3 | В    |
+----+------+
SELECT * FROM tbl2;
+----+--------+
| id | letter |
+----+--------+
|  2 | Г      |
|  3 | Д      |
|  4 | Е      |
+----+--------+
SELECT * FROM tbl1, tbl2;
+----+------+----+--------+
| id | name | id | letter |
+----+------+----+--------+
|  1 | А    |  2 | Г      |
|  2 | Б    |  2 | Г      |
|  3 | В    |  2 | Г      |
```

```
|  1 | А    |  3 | Д      |
|  2 | Б    |  3 | Д      |
|  3 | В    |  3 | Д      |
|  1 | А    |  4 | Е      |
|  2 | Б    |  4 | Е      |
|  3 | В    |  4 | Е      |
+----+------+----+--------+
```

Двухтабличный запрос, представленный в листинге 5.77, называют также *перекрестным объединением*. При таком объединении каждая строка одной таблицы объединяется с каждой строкой другой таблицы, создавая тем самым все возможные комбинации строк обеих таблиц. Результирующая таблица содержит число столбцов, равное сумме столбцов в объединяемых таблицах. Таблицы `tbl1` и `tbl2` содержат по два столбца, поэтому результирующая таблица содержит 4 столбца (2 + 2 = 4). Число записей в результирующей таблице определяется произведением числа записей в таблицах, участвующих в многотабличном запросе (3 × 3 = 9). Схематично запросы из листинга 5.77 представлены на рис. 5.1.

SELECT * FROM tbl1	
id	**name**
1	А
2	Б
3	В

×

SELECT * FROM tbl2	
id	**letter**
2	Г
3	Д
4	Е

SELECT * FROM tbl1, tbl2			
tbl1.id	**tbl1.name**	**tbl2.id**	**tbl2.letter**
1	А	2	Г
2	Б	2	Г
3	В	2	Г
1	А	3	Д
2	Б	3	Д
3	В	3	Д
1	А	4	Е
2	Б	4	Е
3	В	4	Е

Рис. 5.1. Схематичное представление листинга 5.77

Точно так же, как и в обычном операторе `SELECT`, в многотабличном операторе `SELECT` можно управлять числом столбцов в результирующей таблице. Для

этого столбцы, которые должны войти в результирующую таблицу, перечисляются после ключевого слова `SELECT` через запятую (листинг 5.78).

Листинг 5.78. Управление столбцами результирующей таблицы

```
SELECT name, letter FROM tbl1, tbl2;
+------+--------+
| name | letter |
+------+--------+
| А    | Г      |
| Б    | Г      |
| В    | Г      |
| А    | Д      |
| Б    | Д      |
| В    | Д      |
| А    | Е      |
| Б    | Е      |
| В    | Е      |
+------+--------+
```

Однако если в результирующую таблицу добавить столбец `id`, который входит в состав обеих таблиц, СУБД MySQL вернет ошибку — неоднозначность поля `id` в списке полей (листинг 5.79).

Листинг 5.79. Неоднозначность поля `id`

```
SELECT id, name, letter FROM tbl1, tbl2;
ERROR 1052: Column: 'id' in field list is ambiguous
```

Для того чтобы исключить неоднозначность, т. е. определить, поле `id` какой таблицы имеется в виду в запросе из листинга 5.79, прибегают к полным именам столбцов. Полное имя включает имена таблицы и столбца, разделенные точкой (листинг 5.80).

Листинг 5.80. Использование полных имен

```
SELECT tbl1.id, tbl1.name, tbl2.letter FROM tbl1, tbl2;
+----+------+--------+
| id | name | letter |
+----+------+--------+
|  1 | А    | Г      |
|  2 | Б    | Г      |
|  3 | В    | Г      |
```

```
|  1 | A    | Д      |
|  2 | Б    | Д      |
|  3 | В    | Д      |
|  1 | A    | Е      |
|  2 | Б    | Е      |
|  3 | В    | Е      |
+----+------+--------+
```

Для символа `*`, указывающего на необходимость выборки всех столбцов таблицы, также можно использовать полное имя: `tbl1.*` и `tbl2.*` (листинг 5.81).

Листинг 5.81. Использование полного имени совместно с символом *

```
SELECT tbl1.*, tbl2.* FROM tbl1, tbl2;
+----+------+----+--------+
| id | name | id | letter |
+----+------+----+--------+
|  1 | A    |  2 | Г      |
|  2 | Б    |  2 | Г      |
|  3 | В    |  2 | Г      |
|  1 | A    |  3 | Д      |
|  2 | Б    |  3 | Д      |
|  3 | В    |  3 | Д      |
|  1 | A    |  4 | Е      |
|  2 | Б    |  4 | Е      |
|  3 | В    |  4 | Е      |
+----+------+----+--------+
SELECT tbl1.* FROM tbl1, tbl2;
+----+------+
| id | name |
+----+------+
|  1 | A    |
|  2 | Б    |
|  3 | В    |
|  1 | A    |
|  2 | Б    |
|  3 | В    |
|  1 | A    |
|  2 | Б    |
|  3 | В    |
+----+------+
```

Замечание

Полные имена можно использовать для обращения не только в пределах одной базы данных, но и для объединения таблиц из двух разных баз данных. Для

этого полное имя столбца предваряется именем базы данных. Если таблицы `tbl1` и `tbl2` расположены в базах данных `db1` и `db2`, то к столбцам `name` и `letter` можно обратиться по именам `db1.tbl1.name` и `db2.tbl2.letter` соответственно. При этом названия таблиц следует также записывать полным именем: `db1.tbl1` и `db2.tbl2`.

В реальных запросах редко требуется выводить всевозможные комбинации строк объединяемых таблиц. Чаще число строк в результирующей таблице ограничивается при помощи условия `WHERE`. В листинге 5.82 производится выборка из таблиц `tbl1` и `tbl2` при условии совпадения полей `id` этих таблиц.

Листинг 5.82. Ограничение числа строк при помощи условия `WHERE`

```
SELECT tbl1.id, tbl1.name, tbl2.letter FROM tbl1, tbl2
WHERE tbl1.id = tbl2.id;
+----+------+--------+
| id | name | letter |
+----+------+--------+
|  2 | Б    | Г      |
|  3 | В    | Д      |
+----+------+--------+
```

Еще одним эффективным способом ограничения числа столбцов является группировка результата выборки по одному из полей при помощи конструкции `GROUP BY` (листинг 5.83).

Листинг 5.83. Ограничение числа строк при помощи конструкции `GROUP BY`

```
SELECT tbl1.id, tbl1.name, tbl2.letter FROM tbl1, tbl2
GROUP BY tbl1.id;
+----+------+--------+
| id | name | letter |
+----+------+--------+
|  1 | А    | Г      |
|  2 | Б    | Г      |
|  3 | В    | Г      |
+----+------+--------+
```

При работе с результирующей таблицей в прикладных программах часто бывает неудобно работать с полными именами столбцов. Поэтому прибегают к назначению псевдонимов при помощи оператора `AS`. В листинге 5.84 полному имени `tbl1.id` назначается псевдоним `id`, имени `tbl1.name` — `name`, а `tbl2.letter` — `letter`.

Листинг 5.84. Назначение псевдонимов

```
SELECT tbl1.id AS id, tbl1.name AS name, tbl2.letter AS letter
FROM tbl1, tbl2 WHERE tbl1.id = tbl2.id;
+----+------+--------+
| id | name | letter |
+----+------+--------+
|  2 | Б    | Г      |
|  3 | В    | Д      |
+----+------+--------+
```

Оператор `AS` может использоваться не только для назначения псевдонимов столбцам, но и для назначения псевдонимов таблицам. В листинге 5.85 таблицам `tbl1` и `tbl2` назначаются псевдонимы `t1` и `t2` соответственно. Такой подход позволяет использовать в качестве имен таблиц более короткие имена, что приводит к более короткому SQL-запросу.

Листинг 5.85. Назначение псевдонимов таблицам

```
SELECT t1.id, t1.name, t2.letter
FROM tbl1 AS t1, tbl2 AS t2 WHERE t1.id = t2.id;
+----+------+--------+
| id | name | letter |
+----+------+--------+
|  2 | Б    | Г      |
|  3 | В    | Д      |
+----+------+--------+
```

Замечание

После назначения псевдонимов таблицам, участвующим в запросе, использовать исходные имена таблиц в конструкциях `WHERE`, `GROUP BY`, `ORDER BY` и списке столбцов после ключевого слова `SELECT` уже не допускается.

5.22. Самообъединение таблиц

Кроме того, такое назначение псевдонимов позволяет осуществлять многотабличные запросы к одной таблице. Такие запросы называют еще *самообъединением таблицы*. Для этого достаточно назначить одной и той же таблице разные псевдонимы (листинг 5.86).

Листинг 5.86. Многотабличный запрос к одной таблице

```
SELECT t1.id, t2.name FROM tbl1 AS t1, tbl1 AS t2
WHERE t1.id = t2.id;
```

```
+----+------+
| id | name |
+----+------+
|  1 | А    |
|  2 | Б    |
|  3 | В    |
+----+------+
```

При самообъединении нужно учитывать, что таблице следует присваивать псевдонимы при помощи оператора AS, иначе сервер MySQL возвратит ошибку.

5.23. Объединение таблиц при помощи *JOIN*

Оператор JOIN позволяет объединять таблицы и имеет многочисленные варианты использования, которые будут рассмотрены далее на протяжении данного раздела. В SQL-запросе данный оператор располагается между именами объединяемых таблиц после ключевого слова FROM. Без дополнительных ключевых слов объединение при помощи ключевого слова JOIN аналогично перекрестному объединению таблиц (листинг 5.87), рассмотренному в *разд. 5.21*.

Листинг 5.87. Использование ключевого слова JOIN

```
SELECT * FROM tbl1, tbl2 WHERE tbl1.id = tbl2.id;
+----+------+----+--------+
| id | name | id | letter |
+----+------+----+--------+
|  2 | Б    |  2 | Г      |
|  3 | В    |  3 | Д      |
+----+------+----+--------+
SELECT * FROM tbl1 JOIN tbl2 WHERE tbl1.id = tbl2.id;
+----+------+----+--------+
| id | name | id | letter |
+----+------+----+--------+
|  2 | Б    |  2 | Г      |
|  3 | В    |  3 | Д      |
+----+------+----+--------+
```

Запросы, представленные в листинге 5.87, полностью идентичны. Для формирования условия в запросах, использующих объединение JOIN, вместо ключевого слова WHERE предпочтительно указывать ключевое слово ON, как это продемонстрировано в листинге 5.88.

Листинг 5.88. Использование ключевого слова ON

```
SELECT * FROM tbl1 JOIN tbl2 ON tbl1.id = tbl2.id;
+----+------+----+--------+
| id | name | id | letter |
+----+------+----+--------+
|  2 | Б    |  2 | Г      |
|  3 | В    |  3 | Д      |
+----+------+----+--------+
```

Замечание

Ключевое слово JOIN имеет два синонима: CROSS JOIN и INNER JOIN.

Помимо перекрестного объединения таблиц, предусмотрено левое и правое объединение таблиц, которое осуществляется при помощи конструкций LEFT JOIN и RIGHT JOIN соответственно.

В листинге 5.87 продемонстрировано перекрестное объединение таблиц tbl1 и tbl2. При этом результирующая таблица содержит комбинации строк обеих таблиц, удовлетворяющих условию tbl1.id = tbl2.id. Левое объединение (LEFT JOIN) позволяет включить в результирующую таблицу строки "левой" таблицы tbl1, которым не нашлось соответствия в "правой" таблице tbl2 (листинг 5.89).

Листинг 5.89. Левое объединение таблиц tbl1 и tbl2

```
SELECT * FROM tbl1 LEFT JOIN tbl2 ON tbl1.id = tbl2.id;
+----+------+------+--------+
| id | name | id   | letter |
+----+------+------+--------+
|  1 | А    | NULL | NULL   |
|  2 | Б    |    2 | Г      |
|  3 | В    |    3 | Д      |
+----+------+------+--------+
```

Как видно из листинга 5.89, в таблице tbl1 для записи со значением id = 1 не нашлось соответствия в таблице tbl2, т. к. поле id в ней принимает значения 2, 3, 4. Тем не менее в результирующую таблицу запись включена, при этом значения полей из таблицы tbl2 принимают значение NULL. Следует заменить, что для задания условия вместо ключевого слова WHERE при левом и правом объединениях используется ключевое слово ON.

В листинге 5.90 демонстрируется "правое" объединение при помощи конструкции RIGHT JOIN.

Листинг 5.90. Правое объединение таблиц `tbl1` и `tbl2`

```
SELECT * FROM tbl1 RIGHT JOIN tbl2 ON tbl1.id = tbl2.id;
+------+------+----+--------+
| id   | name | id | letter |
+------+------+----+--------+
|    2 | Б    |  2 | Г      |
|    3 | В    |  3 | Д      |
| NULL | NULL |  4 | Е      |
+------+------+----+--------+
```

Как видно из листинга 5.90, при правом объединении возвращаются строки, удовлетворяющие условию `tbl1.id = tbl2.id` и строки "правой" таблицы `tbl2`, которым не нашлось соответствия в "левой" таблице `tbl1`.

Замечание

Ключевые слова `LEFT JOIN` и `RIGHT JOIN` имеют синонимы: `LEFT OUTER JOIN` и `RIGHT OUTER JOIN` соответственно.

Другим способом установки связи между таблицами `tbl1` и `tbl2` при правом и левом объединениях является использование ключевого слова `USING()`. В круглых скобках, следующих за этим ключевым словом, перечисляются имена столбцов, которые должны присутствовать в обеих таблицах и для которых необходимо соблюдение равенства. Данный оператор предназначен для создания более компактных SQL-запросов. Так, следующие два выражения идентичны:

```
a LEFT JOIN b USING(c1,c2,c3)
a LEFT JOIN b ON a.c1 = b.c1 AND a.c2 = b.c2 AND a.c3 = b.c3
```

Принимая во внимание синтаксис ключевого слова `USING()`, левое и правое объединения, показанные в листингах 5.89 и 5.90, можно представить так, как это показано в листинге 5.91.

Листинг 5.91. Использование ключевого слова `USING`

```
SELECT * FROM tbl1 LEFT JOIN tbl2 USING(id);
+----+------+------+--------+
| id | name | id   | letter |
+----+------+------+--------+
|  1 | А    | NULL | NULL   |
|  2 | Б    |    2 | Г      |
|  3 | В    |    3 | Д      |
+----+------+------+--------+
SELECT * FROM tbl1 RIGHT JOIN tbl2 USING(id);
```

```
+------+------+----+--------+
| id   | name | id | letter |
+------+------+----+--------+
|    2 | Б    |  2 | Г      |
|    3 | В    |  3 | Д      |
| NULL | NULL |  4 | Е      |
+------+------+----+--------+
```

Для каждой из таблиц, участвующих в объединении с использованием SQL-оператора `JOIN`, можно ввести подсказку о том, как СУБД MySQL должна использовать индексы при извлечении данных из таблицы. Указав после имени таблицы ключевое слово `USE INDEX(`*`list`*`)`, в скобках можно задать список индексов *`list`*, которые СУБД MySQL должна использовать при поиске записей в таблице. Ключевое слово `IGNORE INDEX(`*`list`*`)` предназначено для того, чтобы запретить СУБД MySQL использовать какой-то отдельный индекс. Ключевое слово `FORCE INDEX(`*`list`*`)` подобно `USE INDEX(`*`list`*`)`, но с тем отличием, что сканирование таблицы расценивается как очень дорогая операция. Это учитывается оптимизатором MySQL, и полное сканирование таблицы производится только в том случае, если нет возможности использовать индекс.

5.24. Вложенный запрос, возвращающий одно значение

Вложенный запрос — это запрос, который является составной частью другого запроса. В листинге 5.92 представлены две таблицы — `catalogs` и `products`, каждая запись таблицы `products` содержит внешний ключ `id_catalog`, который устанавливает связь между таблицами так, как это представлено на рис. 5.2.

Листинг 5.92. Таблицы `catalogs` и `products`

```
CREATE TABLE catalogs (
  id_catalog int(11) NOT NULL auto_increment,
  name tinytext NOT NULL,
  PRIMARY KEY  (id_catalog)
);

INSERT INTO catalogs VALUES (1, 'Процессоры'), (2, 'Материнские платы'),
(3, 'Видеоадаптеры'), (4, 'Жесткие диски'), (5, 'Оперативная память');

CREATE TABLE products (
  id_product int(11) NOT NULL auto_increment,
  name tinytext NOT NULL,
```

```
  id_catalog int(11) NOT NULL,
  PRIMARY KEY  (id_product),
  KEY id_catalog (id_catalog)
);

INSERT INTO products VALUES (1, 'Материнская плата N 1', 2),
(2, 'Оперативная память N 1', 5), (3, 'Процессор N 1', 1),
(4, 'Процессор N 2', 1), (5, 'Процессор N 3', 1),
(6, 'Оперативная память N 2', 5), (7, 'Материнская плата N 2', 2),
(8, 'Жесткий диск N 1', 4), (9, 'Видеоадаптер N 1', 3);
```

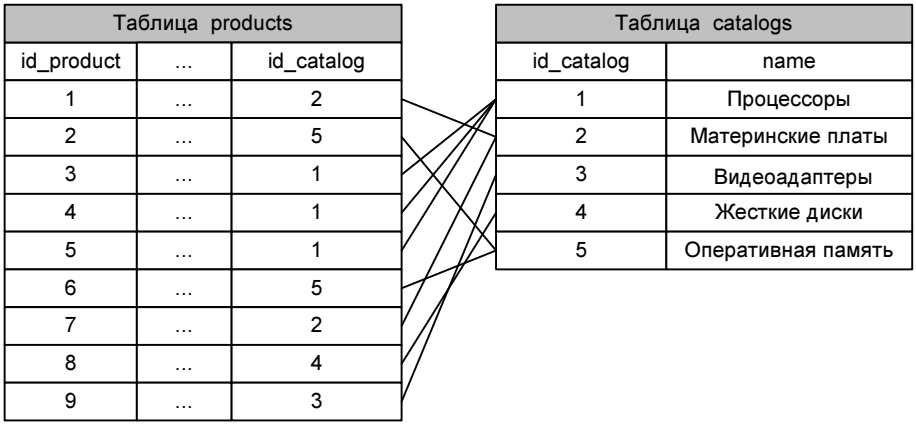


Рис. 5.2. Связанные друг с другом таблицы

Пусть необходимо вывести все позиции из таблицы `products`, которые соответствуют элементу каталога "Процессоры". Решить данную проблему можно при помощи многотабличного запроса, представленного в листинге 5.93.

Листинг 5.93. Многотабличный запрос

```
SELECT products.* FROM catalogs, products
WHERE catalogs.id_catalog = products.id_catalog AND
      catalogs.name = 'Процессоры';
+------------+---------------+------------+
| id_product | name          | id_catalog |
+------------+---------------+------------+
|          3 | Процессор N 1 |          1 |
|          4 | Процессор N 2 |          1 |
|          5 | Процессор N 3 |          1 |
+------------+---------------+------------+
```

Многотабличные запросы зачастую сложны в использовании, поэтому часто прибегают к вложенным запросам. В листинг 5.94 приводится пример SELECT-запроса, который эквивалентен запросу из листинга 5.93.

Листинг 5.94. Применение вложенного запроса

```
SELECT * FROM products
WHERE id_catalog = (SELECT id_catalog FROM catalogs
                    WHERE name = 'Процессоры');
+------------+---------------+------------+
| id_product | name          | id_catalog |
+------------+---------------+------------+
|          3 | Процессор N 1 |          1 |
|          4 | Процессор N 2 |          1 |
|          5 | Процессор N 3 |          1 |
+------------+---------------+------------+
```

Как видно из листинга 5.94, здесь в качестве внешнего запроса выступает

```
SELECT * FROM products
```

а в качестве вложенного запроса используется запрос

```
SELECT id_catalog FROM catalogs WHERE name = 'Процессоры'
```

Таким образом, результат вложенного запроса (id_catalog), минуя промежуточные переменные в MySQL или внешней программе, применяется в качестве элемента внешнего запроса. Вложенный запрос всегда помещается в круглые скобки.

В отличие от вложенного запроса, многотабличный запрос менее нагляден, хотя в ряде случаев выполняется быстрее, т. к. оптимизация многотабличных запросов происходит эффективнее вложенных. В большинстве случаев запросы можно представить как при помощи одного из объединений (перекрестного, левого, правого), так и при помощи вложенных запросов, но все же имеется ряд задач, которые решаются только при помощи вложенных запросов.

Степень вложения запросов не ограничена вторым уровнем, могут использоваться трехуровневые и многоуровневые вложенные запросы. В листинге 5.95 извлекается название каталога из таблицы catalogs, которому принадлежит последняя позиция в таблице products.

Листинг 5.95. Каталог, которому принадлежит последняя позиция

```
SELECT name FROM catalogs
WHERE id_catalog = (SELECT id_catalog FROM products
```

```
                    WHERE id_product = (SELECT MAX(id_product)
                                        FROM products));
+---------------+
| name          |
+---------------+
| Видеоадаптеры |
+---------------+
```

При помощи самого глубоко вложенного запроса сначала извлекается последний первичный ключ `id_product` из таблицы `products`:

```
SELECT MAX(id_product) FROM products
```

При помощи второго вложенного запроса извлекается вторичный ключ `id_catalog`, соответствующий полученному значению `id_product` первого вложенного запроса:

```
SELECT id_catalog FROM products
WHERE id_product = (SELECT MAX(id_product)
                    FROM products)
```

Полученное значение `id_catalog` используется для извлечения названия во внешнем запросе (см. листинг 5.95).

Замечание

На практике редко прибегают к вложенным запросам со степенью вложения больше трех, т. к. высокая степень вложенности быстро приводит к увеличению времени выполнения запроса.

Вложенный запрос, возвращающий единственное значение, просто подставляет результат на место своего выполнения (листинг 5.96).

Листинг 5.96. Использование вложенного запроса в списке столбцов

```
SELECT id_catalog, (SELECT MAX(id_product) FROM products)
FROM catalogs;
+------------+----------------------------------------+
| id_catalog | (SELECT MAX(id_product) FROM products) |
+------------+----------------------------------------+
|          1 |                                      9 |
|          2 |                                      9 |
|          3 |                                      9 |
|          4 |                                      9 |
|          5 |                                      9 |
+------------+----------------------------------------+
```

В листинге 5.96 вложенный запрос `SELECT MAX(id_product) FROM products` возвращает максимальное значение столбца `id_product` (`9`) из таблицы `products` и подставляет это значение на место запроса, поэтому запрос из листинга 5.96

можно представить, как запрос со скалярной величиной в качестве столбца (листинг 5.97).

Листинг 5.97. Запрос со скалярной величиной в качестве столбца

```
SELECT id_catalog, 9 FROM catalogs;
+------------+---+
| id_catalog | 9 |
+------------+---+
|          1 | 9 |
|          2 | 9 |
|          3 | 9 |
|          4 | 9 |
|          5 | 9 |
+------------+---+
```

Допустимы также запросы вида, представленного в листинге 5.98. Вложенный запрос возвращает название каталога с первичным ключом `id_catalog = 1`, а внешний запрос, состоящий из одного оператора `SELECT`, выводит его.

Листинг 5.98. Использование вложенного запроса как скалярной величины

```
SELECT (SELECT name FROM catalogs WHERE id_catalog = 1);
+-------------------------------------------------+
| (SELECT name FROM catalogs WHERE id_catalog = 1) |
+-------------------------------------------------+
| Процессоры                                      |
+-------------------------------------------------+
```

При использовании вложенных запросов в качестве аргументов встроенных функций MySQL следует помнить о необходимости помещения вложенного запроса в дополнительные круглые скобки. В листинге 5.99 название каталога с первичным ключом `id_catalog = 1` из таблицы `catalogs` передается в качестве аргумента функции `UPPER()`, которая переводит строку в верхний регистр. При этом используются двойные круглые скобки: одна пара для обозначения вложенного запроса, другая пара — для обозначения признака функции.

Листинг 5.99. Использование вложенного запроса в качестве аргумента функции

```
SELECT UPPER((SELECT name FROM catalogs WHERE id_catalog = 1));
+---------------------------------------------------------+
| UPPER((SELECT name FROM catalogs WHERE id_catalog = 1)) |
+---------------------------------------------------------+
| ПРОЦЕССОРЫ                                              |
+---------------------------------------------------------+
```

Наиболее часто вложенные запросы применяются в операциях сравнения в условиях, которые задаются ключевыми словами WHERE, HAVING или ON. Для этого задействуются шесть операторов сравнения (=, <>, <, <=, >, >=). Вложенный запрос, участвующий в операции сравнения, должен возвращать в качестве результата единичное значение. В листинге 5.100 приводится пример запроса, который выводит список элементов каталога, чей первичный ключ больше, чем первичный ключ каталога, которому принадлежит позиция с ключом id_product, равным пяти.

Листинг 5.100. Использование вложенных запросов в операциях сравнения

```
SELECT name FROM catalogs
WHERE id_catalog > (SELECT id_catalog FROM products
                    WHERE id_product = 5);
+--------------------+
| name               |
+--------------------+
| Материнские платы  |
| Видеоадаптеры      |
| Жесткие диски      |
| Оперативная память |
+--------------------+
SELECT name FROM catalogs
WHERE (SELECT id_catalog FROM products
       WHERE id_product = 5) < id_catalog;
+--------------------+
| name               |
+--------------------+
| Видеоадаптеры      |
| Жесткие диски      |
| Оперативная память |
+--------------------+
```

Замечание

В листинге 5.100 показано, что вложенный запрос может располагаться как справа, так и слева от оператора сравнения. Однако предпочтительнее располагать его в правой части, как это продемонстрировано в первом запросе, для совместимости с другими СУБД, не удовлетворяющими стандарту SQL2, в котором было устранено подобное ограничение.

5.25. Вложенные запросы, возвращающие несколько строк

В предыдущем разделе рассмотрены запросы, где вложенный запрос возвращает единственное значение. Если вложенный запрос возвращает несколько

строк, СУБД MySQL генерирует ошибку 1242 — "Вложенный запрос возвращает более одной строки" (листинг 5.101).

Листинг 5.101. Вложенный запрос возвращает более одной строки

```
SELECT name FROM catalogs
WHERE id_catalog = (SELECT id_catalog FROM products);
ERROR 1242: Subquery returns more than 1 row
```

Для того чтобы предотвратить такую ошибку, там, где во вложенном запросе ожидается лишь одно значение, разумно использовать конструкцию `LIMIT 1` (листинг 5.102).

Листинг 5.102. Использование конструкции `LIMIT`

```
SELECT name FROM catalogs
WHERE id_catalog = (SELECT id_catalog FROM products LIMIT 1);
+------------+
| name       |
+------------+
| Процессоры |
+------------+
```

5.25.1. Ключевое слово *IN*

Для того чтобы из таблицы `catalogs` выбрать строки, у которых первичный ключ `id_catalog` совпадает с одним из значений, возвращаемых вложенным запросом, следует воспользоваться конструкцией `IN` (листинг 5.103).

Листинг 5.103. Использование конструкции `IN`

```
SELECT name FROM catalogs
WHERE id_catalog IN (SELECT id_catalog FROM products
                     GROUP BY id_catalog)
ORDER BY name;
+--------------------+
| name               |
+--------------------+
| Видеоадаптеры      |
| Жесткие диски      |
| Материнские платы  |
| Оперативная память |
| Процессоры         |
+--------------------+
```

Запрос, представленный в листинге 5.103, аналогичен запросу, показанному в листинге 5.104.

Листинг 5.104. Использование конструкции IN совместно со списком скалярных величин

```
SELECT name FROM catalogs
WHERE id_catalog IN (1,2,3,4,5)
ORDER BY name;
+--------------------+
| name               |
+--------------------+
| Видеоадаптеры      |
| Жесткие диски      |
| Материнские платы  |
| Оперативная память |
| Процессоры         |
+--------------------+
```

Для того чтобы возвратить строки, которые отсутствуют в результирующей таблице, возвращаемой вложенным запросом, следует воспользоваться оператором NOT IN, представленным в листинге 5.105.

Листинг 5.105. Использование оператора NOT IN

```
SELECT name FROM catalogs
WHERE id_catalog NOT IN (SELECT DISTINCT id_catalog
                         FROM products WHERE id_catalog < 3)
ORDER BY name;
+--------------------+
| name               |
+--------------------+
| Видеоадаптеры      |
| Жесткие диски      |
| Оперативная память |
+--------------------+
```

5.25.2. Ключевое слово *ANY (SOME)*

Конструкция IN позволяет осуществить поиск величины в списке и выражает логику оператора "равно" (=). Однако на месте оператора = в этом запросе может стоять другой оператор сравнения (листинг 5.106).

Листинг 5.106. Вложенный запрос возвращает более одной строки

```
SELECT name FROM catalogs
WHERE id_catalog > (SELECT id_catalog FROM products);
ERROR 1242: Subquery returns more than 1 row
```

Данный запрос также завершается неудачей, но сформулировать его с использованием конструкции IN уже не получится. К счастью, язык запросов SQL обладает средствами решения подобных задач. Для этого применяется ключевое слово ANY (листинг 5.107).

Замечание

Ключевое слово ANY имеет синоним SOME, однако предпочтительнее использовать именно ANY.

Листинг 5.107. Использование ключевого слова ANY

```
SELECT id_catalog, name FROM catalogs
WHERE id_catalog > ANY (SELECT id_catalog FROM products);
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
```

Ключевое слово ANY применяется для сравнения значений с использованием одного из шести операторов сравнения (=, <>, <, <=, >, >=). Проверяемое значение id_catalog поочередно сравнивается с каждым элементом, который возвращает вложенный запрос. Если хотя бы одно из сравнений возвращает 1 (истина), строка выводится запросом. В листинге 5.107 происходит сравнение значений первичного ключа id_catalog (1, 2, 3, 4, 5), которые присутствуют в таблице catalogs, со значениями поля id_catalog (1, 2, 3, 4, 5) из таблицы products. Значение id_catalog = 1 не удовлетворяет ни одному условию:

```
1 > 1 -- 0 (ложь)
1 > 2 -- 0 (ложь)
1 > 3 -- 0 (ложь)
1 > 4 -- 0 (ложь)
1 > 5 -- 0 (ложь)
```

Поэтому в результаты эта строка не попадает, в то же время все остальные цифры удовлетворяют хотя бы одному условию и попадают в результирующую таблицу:

```
3 > 1 -- 1 (истина)
3 > 2 -- 1 (истина)
3 > 3 -- 0 (ложь)
3 > 4 -- 0 (ложь)
3 > 5 -- 0 (ложь)
```

То есть запрос вида

```
"WHERE X > ANY (SELECT Y...)"
```

можно интерпретировать как "где X больше хотя бы одного выбранного Y", а запрос вида

```
"WHERE X < ANY (SELECT Y...)"
```

следует читать "где X меньше, хотя бы одного Y...".

5.25.3. Ключевое слово *ALL*

Вместо ключевого слова ANY может быть использовано ключевое слово ALL, которое точно так же применяется совместно с одним из шести операторов сравнения (=, <>, <, <=, >, >=). В этом случае проверяемое значение также поочередно сравнивается с каждым элементом, который возвращает вложенный запрос, но строка возвращается только тогда, когда все сравнения дают 1 (истина).

Замечание

Если в выражениях с ключевым словом ANY используется логика ИЛИ, т. е. достаточно, чтобы срабатывало хотя бы одно из многих условий, то в случае ALL используется логика И — должны срабатывать все условия.

В листинге 5.108 представлен запрос, возвращающий все товарные позиции из таблицы products базы данных shop, цена которых превышает среднюю цену каждого из элементов каталога.

Листинг 5.108. Использование ключевого слова ALL

```
SELECT * FROM catalogs
WHERE id_catalog >= ALL (SELECT id_catalog FROM products
                         GROUP BY id_catalog);
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          5 | Оперативная память |
+------------+--------------------+
```

Для того чтобы понять логику запроса, представленного в листинге 5.108, полезно выполнить вложенный запрос отдельно (листинг 5.109).

Листинг 5.109. Вложенный запрос

```
SELECT id_catalog FROM products GROUP BY id_catalog;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
```

Условие ALL в листинге 5.108 эквивалентно требованию вывести все элементы каталога, чей первичный ключ больше или равен каждому значению из результирующей таблицы запроса, представленного в листинге 5.109.

5.26. Проверка результирующей таблицы на существование

Результирующая таблица, которая возвращается вложенным запросом, может быть пустой, т. е. не содержать ни одной строки. Для проверки данного факта предназначены ключевые слова EXISTS и NOT EXISTS. Данные ключевые слова не требуют левого операнда, они просто сообщают, имеются ли в результирующей таблице строки или нет. Если вложенный запрос содержит более одной строки, EXISTS возвращает 1 (истина), в противном случае ключевое слово возвращает 0 (ложь). Ключевое слово NOT EXISTS действует противоположным образом.

Замечание

Проверка на существование допустима только во вложенных запросах.

Добавим в таблицу catalogs из листинга 5.92 две позиции: "Периферия" и "Сетевое оборудование", которым не будет соответствовать ни одна позиция из таблицы products (листинг 5.110).

Листинг 5.110. Две новые позиции в таблице catalogs

```
INSERT INTO catalogs VALUES (6, 'Периферия'),
                            (7, 'Сетевое оборудование');
SELECT * FROM catalogs;
```

```
+------------+----------------------+
| id_catalog | name                 |
+------------+----------------------+
|          1 | Процессоры           |
|          2 | Материнские платы    |
|          3 | Видеоадаптеры        |
|          4 | Жесткие диски        |
|          5 | Оперативная память   |
|          6 | Периферия            |
|          7 | Сетевое оборудование |
+------------+----------------------+
```

Пусть требуется вывести названия каталогов из таблицы `catalogs`, для которых имеется хотя бы одно соответствие в таблице `products` (листинг 5.111).

Листинг 5.111. Использование ключевого слова `EXISTS`

```
SELECT * FROM catalogs
WHERE EXISTS (SELECT * FROM products
              WHERE products.id_catalog = catalogs.id_catalog);
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
```

Как видно из листинга 5.111, элементы каталога "Периферия" и "Сетевое оборудование", для которых нет ни одной позиции в таблице `products`, не попали в результирующую таблицу. Внешний запрос последовательно перебирает все строки таблицы `catalogs` и для каждого элемента каталога выполняет вложенный запрос, который проверяет, имеется ли в таблице `products` текущий товар.

В списке столбцов, следующих после ключевого слова `SELECT` вложенного запроса, вместо символа `*` может быть расположено любое допустимое имя — название столбца или просто цифра, на результатах это не отражается (листинг 5.112).

Листинг 5.112. Использование скалярного значения во вложенном запросе

```
SELECT * FROM catalogs
WHERE EXISTS (SELECT 7 FROM products
              WHERE products.id_catalog = catalogs.id_catalog);
```

```
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
```

Важно отметить, что в `WHERE`-условии вложенного запроса допускается использование столбцов внешнего запроса при обращении к ним по полному имени. В листинге 5.112 к текущей товарной позиции из внешнего запроса во вложенном запросе можно обратиться по имени `catalogs.id_catalog`.

Замечание

Обращение к текущей строке внешнего запроса из вложенного запроса называется *внешней ссылкой* и может быть использовано не только совместно с ключевым словом `EXISTS`, но и во всех других вложенных запросах. Запросы с использованием внешних ссылок называются *коррелированными запросами*.

Обратную задачу, т. е. извлечение элементов каталога, которым не соответствует ни одна позиция в таблице `products`, можно решить, воспользовавшись ключевым словом `NOT EXISTS` (листинг 5.113).

Листинг 5.113. Использование ключевого слова `NOT EXISTS`

```
SELECT * FROM catalogs
WHERE NOT EXISTS (SELECT 7 FROM products
              WHERE products.id_catalog = catalogs.id_catalog);
+------------+----------------------+
| id_catalog | name                 |
+------------+----------------------+
|          6 | Периферия            |
|          7 | Сетевое оборудование |
+------------+----------------------+
```

5.27. Вложенные запросы, возвращающие несколько столбцов

До сих пор рассматривались вложенные запросы, возвращающие единственный столбец, однако в СУБД MySQL реализованы так называемые *строчные запросы*, которые возвращают более одного столбца. Примеры таких запросов представлены в листинге 5.114.

Листинг 5.114. Пример строчных запросов

```
SELECT * FROM tbl1 WHERE (1,2) = (SELECT col1, col2 FROM tbl2);
SELECT * FROM tbl1 WHERE ROW(1,2) = (SELECT col1, col2 FROM tbl2);
```

Запрос возвращает строку, если в таблице `tbl2` присутствует запись, для которой справедливы равенства `col1 = 1` и `col2 = 2`.

Выражения `(1,2)` и `ROW(1,2)` называются *конструкторами строки*. Оба выражения являются эквивалентными, но в целях увеличения читабельности запроса предпочтительно использовать второй вариант.

Следует отметить, что конструктор строки может быть использован не только с вложенными запросами, но и в обычных запросах, как это представлено в листинге 5.115, где из таблицы `catalogs` извлекаются записи с первичным ключом `id_catalog`, равным 1, и полем `name`, содержащим строку "Процессоры".

Листинг 5.115. Применение конструктора строки

```
SELECT * FROM catalogs
WHERE ROW(1,"Процессоры") = (id_catalog,name);
+------------+------------+
| id_catalog | name       |
+------------+------------+
|          1 | Процессоры |
+------------+------------+
```

Запрос из листинга 5.115 можно переписать так, как это представлено в листинге 5.116.

Листинг 5.116. Альтернативный запрос без применения конструктора строки

```
SELECT * FROM catalogs
WHERE id_catalog = 1 AND name = 'Процессоры';
+------------+------------+
| id_catalog | name       |
+------------+------------+
|          1 | Процессоры |
+------------+------------+
```

Строчные запросы часто используются для сравнения таблиц друг с другом. Пусть таблицы `tbl1` и `tbl2` состоят из трех столбцов: `col1`, `col2` и `col3`. Тогда найти все строки таблицы `tbl1`, которые имеются также и в таблице `tbl2`, можно при помощи запроса, представленного в листинге 5.117.

Листинг 5.117. Вывод строк таблицы tbl1, присутствующих в таблице tbl2

```
SELECT col1, col2, col3 FROM tbl1
                        WHERE ROW(col1, col2, col3)
                        IN (SELECT col1, col2, col3 FROM tbl2);
```

5.28. Подзапросы в конструкции *FROM*

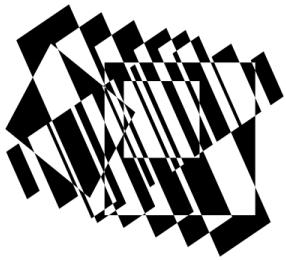
Так как вложенные запросы возвращают результирующую таблицу, которая становится предметом дальнейших запросов, стандарт SQL разрешает использование вложенных запросов везде, где допускаются ссылки на таблицы. В частности, вложенный запрос может указываться вместо имени таблицы в предложении FROM. В листинге 5.118 представлен запрос, в котором в качестве одной из таблиц используется результирующая таблица вложенного запроса.

Листинг 5.118. Использование вложенных запросов в конструкции FROM

```
SELECT prd.id_product, prd.name, cat.name
FROM products AS prd,
     (SELECT * FROM catalogs) AS cat
WHERE prd.id_catalog = cat.id_catalog;
+------------+-----------------------+--------------------+
| id_product | name                  | name               |
+------------+-----------------------+--------------------+
|          1 | Материнская плата N 1  | Материнские платы  |
|          2 | Оперативная память N 1 | Оперативная память |
|          3 | Процессор N 1          | Процессоры         |
|          4 | Процессор N 2          | Процессоры         |
|          5 | Процессор N 3          | Процессоры         |
|          6 | Оперативная память N 2 | Оперативная память |
|          7 | Материнская плата N 2  | Материнские платы  |
|          8 | Жесткий диск N 1       | Жесткие диски      |
|          9 | Видеоадаптер N 1       | Видеоадаптеры      |
+------------+-----------------------+--------------------+
```

В случае вложенного запроса, ключевое слово AS, назначающее псевдоним результирующей таблице, является обязательным, т. к. каждая таблица в конструкции FROM должна иметь имя.

ГЛАВА 6



Встроенные функции

СУБД MySQL предоставляет широкий набор встроенных функций, которые можно использовать для преобразования данных.

Замечание

Отличительной чертой MySQL является то, что при использовании функций пробелы между именем функции и круглыми скобками недопустимы, т. е. написание `NOW()` правильное, а `NOW ()` уже нет. СУБД MySQL можно заставить игнорировать пробелы между именем функции и круглыми скобками, если запустить сервер с параметром `--sql-mode=IGNORE-SPACE` или поместить в секцию `[mysqld]` директиву `sql-mode=IGNORE-SPACE`.

Глава разбита на разделы, каждый из которых посвящен отдельному виду функций. Исключения составляют функции поиска и агрегатные функции, которые рассматриваются в *главах 7* и *8* соответственно.

6.1. Математические функции

Математические функции предназначены для выполнения разнообразных математических расчетов. В табл. 6.1 представлен список математических функций, доступных в СУБД MySQL.

Таблица 6.1. *Математические функции MySQL*

Функция	Описание
`ABS(X)`	Возвращает абсолютное значение аргумента `X`
`ACOS(X)`	Возвращает арккосинус числа `X` или `NULL`, если значение `X` выходит из диапазона от –1 до 1
`ASIN(X)`	Возвращает арксинус числа `X` или `NULL`, если значение `X` выходит из диапазона от –1 до 1

Таблица 6.1 *(продолжение)*

Функция	Описание
ATAN(X)	Возвращает арктангенс числа X
ATAN(X, Y), ATAN2(X, Y)	Возвращает арктангенс частного X/Y, функции аналогичны ATAN(X/Y)
CEILING(X), CEIL(X)	Функция принимает дробное число X и возвращает наименьшее целое, не меньше чем X
COALESCE(*val*, ...)	Возвращает первый элемент из списка, который не равен NULL
COS(X)	Возвращает косинус угла X, заданного в радианах
COT(X)	Возвращает котангенс угла X, заданного в радианах
CRC32(*str*)	Возвращает значение кода циклической проверки избыточности строки *str* — 32-битное значение в диапазоне от 0 до $2^{32} - 1$. Если в качестве аргумента функции передано значение NULL, функция возвращает NULL
DEGREES(X)	Возвращает значение угла X, преобразованное из радиан в градусы
GREATEST(*val*, ...)	Возвращает максимальное значение из списка
EXP(X)	Возвращает значение степени числа X: e^X, где *e* — основание натурального логарифма
FLOOR(X)	Принимает дробное число X и возвращает наибольшее целое значение, не больше чем X
INTERVAL(N,N1,N2, ...)	Возвращает 0, если N < N1, и 1, если N < N2, и т. д., т. е. возвращается позиция, где происходит нарушение монотонного убывания значений списка. Все аргументы трактуются как целые числа
LEAST(*val*, ...)	Возвращает минимальное значение из списка
LOG(X) , LN(X)	Возвращает натуральный логарифм (с основанием *e*) числа X
LOG(B, X)	Возвращает логарифм числа X по основанию B
LOG2(X)	Возвращает логарифм числа X по основанию 2
LOG10(X)	Возвращает логарифм числа X по основанию 10
MOD(M, N), M % N, M MOD N	Возвращает остаток от деления M на N
PI()	Возвращает значение числа π
POWER(X, Y), POW(X, Y)	Возвращает значение числа X, возведенного в степень Y — X^Y

Таблица 6.1 *(окончание)*

Функция	Описание
RADIANS(X)	Возвращает значение угла X, преобразованное из градусов в радианы
RAND()	Возвращает случайное значение с плавающей точкой в диапазоне от 0,0 до 1,0. Каждый раз генерируется одна и та же последовательность случайных чисел
RAND(N)	Возвращает случайное значение с плавающей точкой в диапазоне от 0,0 до 1,0. Генератор случайных чисел инициируется числом N
ROUND(X)	Возвращает округленное до ближайшего целого значение числа X
ROUND(X, D)	Возвращает округленное значение числа X до D знаков после запятой
SIGN(X)	Позволяет определить знак числа X и возвращает –1, 0 или 1, если X отрицательно, равно 0 или положительно, соответственно
SIN(X)	Возвращает синус угла X, заданного в радианах
SQPT(X)	Возвращает квадратный корень числа X
TAN(X)	Возвращает тангенс угла X, заданного в радианах
TRUNCATE(X, D)	Возвращает число X с дробной частью, имеющей D знаков после запятой. Если количество знаков после запятой в числе X больше D, лишние разряды усекаются, если меньше, то в конец числа добавляются нули

6.1.1. Вычисление площади треугольников

Пусть имеется таблица triangle, которая состоит из трех полей, содержащих величины угла (angle) и длины двух прилежащих сторон *a* и *b* (рис. 6.1). В листинге 6.1 приведен дамп таблицы triangle.

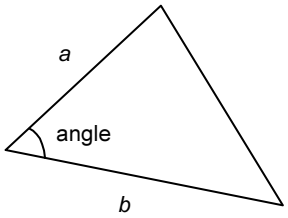


Рис. 6.1. Вычисление площади треугольника

Листинг 6.1. Дамп таблицы `triangle`

```
CREATE TABLE triangle (
  angle DOUBLE NOT NULL,
  A DOUBLE NOT NULL,
  B DOUBLE NOT NULL
);
INSERT INTO triangle VALUES (45, 1.414, 1),
                            (60, 2.707, 2.104),
                            (56, 2.088, 2.112),
                            (23, 5.014, 2.304),
                            (38, 3.482, 4.708);
SELECT * FROM triangle;
+-------+-------+-------+
| angle | A     | B     |
+-------+-------+-------+
|    45 | 1.414 |     1 |
|    60 | 2.707 | 2.104 |
|    56 | 2.088 | 2.112 |
|    23 | 5.014 | 2.304 |
|    38 | 3.482 | 4.708 |
+-------+-------+-------+
```

Вычислим площади треугольников, параметры которых содержатся в таблице `triangle` (листинг 6.1) по формуле

$$S = a \times b \times \sin(angle).$$

SQL-запрос, вычисляющий площади треугольников по представленной выше формуле, может выглядеть так, как это показано в листинге 6.2.

Листинг 6.2. Вычисление площадей треугольников

```
SELECT A*B*SIN(RADIANS(angle))/2.0 AS S FROM triangle;
+-----------------+
| S               |
+-----------------+
| 0.4999244942989 |
| 2.4662359679828 |
| 1.8279681567786 |
| 2.2569130117385 |
| 5.0463480871756 |
+-----------------+
```

Как видно из листинга, прежде чем воспользоваться функцией синуса `SIN()`, необходимо конвертировать угол из градусов в радианы при помощи функции `RADIANS()`.

6.1.2. Округление результатов вычисления

Как видно из листинга 6.2, результат вычисления имеет 13 знаков после запятой, что может быть не очень удобным для восприятия. В листинге 6.3 результат вычислений округляется до третьего знака после запятой при помощи функции `ROUND()`.

Листинг 6.3. Округление результата вычисления до третьего знака после запятой

```
SELECT ROUND(A*B*SIN(RADIANS(angle))/2.0, 3) AS S FROM triangle;
+-------+
| S     |
+-------+
| 0.500 |
| 2.466 |
| 1.828 |
| 2.257 |
| 5.046 |
+-------+
```

Функция `ROUND()` осуществляет математическое округление, т. е. до ближайшего целого числа.

До версии 5.0.3 округление чисел вида 4.5, 10.5 зависело от реализации библиотеки C. Такие значения могли быть округлены как в большую, так и в меньшую сторону. Так как, начиная с версии 5.0.3, MySQL использует собственную точную математическую библиотеку, округление производится согласно следующим правилам:

- для чисел, значение которых однозначно определяется дробной частью, округление чисел со значением дробной части `.5` и выше производится в сторону большего числа, для чисел со значением строго меньше `.5` (например, `.49999`) округление производится в сторону меньшего числа;
- для приближенных чисел (записанных в научной форме `25E-1`) результат по-прежнему зависит от реализации библиотеки C.

В листинге 6.4 показан пример использования функции `ROUND()`.

Листинг 6.4. Округление с дробной частью `.5`

```
SELECT ROUND(2.5), ROUND(25E-1);
+------------+--------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+--------------+
| 3          |            2 |
+------------+--------------+
```

Функция `ROUND()` — не единственная функция управления дробными числами. Функция `CEILING(X)` принимает дробное число `X` и возвращает наименьшее целое, не меньше чем `X`.

Функция `CEILING()` не производит округления, как может показаться на первый взгляд. Она возвращает первое целое число, которое встречает справа от значения аргумента (рис. 6.2).

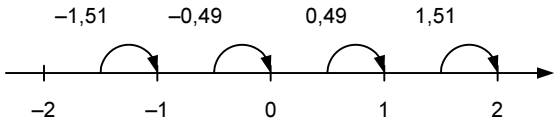


Рис. 6.2. Принцип действия функции `CEILING()`

В листинге 6.5 демонстрируется пример использования функции `CEILING()`.

Листинг 6.5. Использование функции `CEILING()`

```
SELECT CEILING(0.49), CEILING(1.51), CEIL(-0.49), CEIL(-1.51);
+---------------+---------------+-------------+-------------+
| CEILING(0.49) | CEILING(1.51) | CEIL(-0.49) | CEIL(-1.51) |
+---------------+---------------+-------------+-------------+
|             1 |             2 |           0 |          -1 |
+---------------+---------------+-------------+-------------+
```

Функция `FLOOR(X)` принимает дробное число `X` и возвращает наибольшее целое значение, не больше чем `X`. Эта функция сходна по действию с функцией `CEILING(X)`, но сдвиг происходит в обратную сторону. Пример использования функции `FLOOR()` представлен в листинге 6.6.

Листинг 6.6. Использование функции `FLOOR()`

```
SELECT FLOOR(0.49), FLOOR(1.51), FLOOR(-0.49), FLOOR(-1.51);
+-------------+-------------+--------------+--------------+
| FLOOR(0.49) | FLOOR(1.51) | FLOOR(-0.49) | FLOOR(-1.51) |
+-------------+-------------+--------------+--------------+
|           0 |           1 |           -1 |           -2 |
+-------------+-------------+--------------+--------------+
```

Последней функцией, управляющей дробными числами, является функция `TRUNCATE(X,D)`, которая возвращает число `X` с дробной частью, имеющей `D` знаков после запятой. Если количество знаков после запятой в числе `X` больше `D`,

лишние разряды усекаются, если меньше, то в конец числа добавляются нули (листинг 6.7).

Листинг 6.7. Использование функции TRUNCATE()

```
SELECT TRUNCATE(1.284,0), TRUNCATE(1.284,1), TRUNCATE(1.284,4);
+-------------------+-------------------+-------------------+
| TRUNCATE(1.284,0) | TRUNCATE(1.284,1) | TRUNCATE(1.284,4) |
+-------------------+-------------------+-------------------+
|                 1 |               1.2 |            1.2840 |
+-------------------+-------------------+-------------------+
```

6.1.3. Вывод случайной записи

Для генерации случайного числа предназначена функция RAND(), которая возвращает случайное число с плавающей точкой в диапазоне от 0,0 до 1,0. Данная функция может принимать в качестве аргумента целое число N, которым инициализируется генератор случайных чисел. Каждый раз, когда функция принимает один и тот же параметр N, она возвращает один и тот же результат. Обычно функция первый раз вызывается с параметром, а все последующие разы — без параметров. В листинге 6.8 приводится пример использования функции RAND().

Листинг 6.8. Использование функции RAND()

```
SELECT RAND(100), RAND(), RAND(100), RAND();
+-----------------+-----------------+-----------------+-----------------+
| RAND(100)       | RAND()          | RAND(100)       | RAND()          |
+-----------------+-----------------+-----------------+-----------------+
| 0.17353134804734| 0.10621183841136| 0.17353134804734| 0.65180650321004|
+-----------------+-----------------+-----------------+-----------------+
```

Задание начального значения для всех MySQL-клиентов производится независимо друг от друга, и в следующем сеансе при том же инициализирующем значении будет получен другой результат.

Функцию RAND() можно использовать после конструкции ORDER BY для выборки записей в случайном порядке. В листинге 6.9 выводится результат вычисления площадей треугольников (см. листинг 6.3) в случайном порядке.

Листинг 6.9. Вывод результата вычислений в случайном порядке

```
SELECT ROUND(A*B*SIN(RADIANS(angle))/2.0, 3) AS S FROM triangle
ORDER BY RAND();
```

```
+-------+
| S     |
+-------+
| 2.257 |
| 1.828 |
| 2.466 |
| 5.046 |
| 0.500 |
+-------+
```

Очень часто конструкцию `ORDER BY RAND()` используют совместно с конструкцией `LIMIT`. В листинге 6.10 при помощи конструкции `LIMIT 1` выводится случайная запись из таблицы `catalogs`.

Листинг 6.10. Вывод случайной записи из таблицы `catalogs`

```
SELECT * FROM catalogs ORDER BY RAND() LIMIT 1;
+------------+---------------+
| id_catalog | name          |
+------------+---------------+
|          3 | Видеоадаптеры |
+------------+---------------+
```

6.2. Функции даты и времени

Группа функций даты и времени предназначена для форматирования и преобразования календарных значений. В табл. 6.2 представлен список календарных функций доступных в СУБД MySQL.

Таблица 6.2. *Функции даты и времени MySQL*

Функция	Описание
`ADDDATE(`*`date`*`, INTERVAL `*`expr type`*`)`, `DATE_ADD(`*`date`*`, INTERVAL `*`expr type`*`)`	Возвращает время *date*, к которому прибавлен временной интервал, определяемый вторым параметром
`ADDTIME(`*`expr1`*`, `*`expr2`*`)`	Возвращает результат сложения двух календарных значений *expr1* и *expr2*
`CONVERT_TZ(`*`dt`*`, `*`from_tz`*`, `*`to_tz`*`)`	Переводит `DATETIME`-значение *dt* из часового пояса *from_tz* в *to_tz*. Если аргументы ошибочны, возвращается значение `NULL`
`CURDATE()`, `CURRENT_DATE`, `CURRENT_DATE()`	Возвращает текущую дату в формате `'YYYY-MM-DD'` или `YYYYDDMM` в зависимости от того, вызывается функция в текстовом или числовом контексте

Таблица 6.2 *(продолжение)*

Функция	**Описание**
CURTIME(), CURRENT_TIME, CURRENT_TIME()	Возвращает текущее время суток в виде 'hh-mm-ss' или hhmmss в зависимости от того, вызывается функция в текстовом или числовом контексте
DATE(*datetime*)	Извлекает из значения даты и времени суток в формате DATETIME, например '2005-04-17 00:26:08', дату, отсекая часы, минуты и секунды — '2005-04-17'
DATEDIFF(*begin*, *end*)	Вычисляет разницу в днях между датами *begin* и *end*. Аргументы функции могут иметь тип DATE или DATETIME, однако при вычислении разницы используется только DATE-часть
DATE_FORMAT(*date*, *format*)	Форматирует время *date* в соответствии со строкой *format*
DAY(*date*), DAYOFMONTH(*date*)	Принимает в качестве аргумента дату *date* и возвращает порядковый номер дня в месяце (от 1 до 31)
DAYNAME(*date*)	Принимает в качестве аргумента дату *date* и возвращает день недели в виде полного английского названия
DAYOFWEEK(*date*)	Принимает в качестве аргумента дату *date* и возвращает порядковый номер дня недели. Следует помнить, что в западных странах неделя начинается с воскресенья, для которого функция возвращает 1, для последнего дня недели, субботы, возвращается 7
DAYOFYEAR(*date*)	Принимает в качестве аргумента дату *date* и возвращает порядковый номер дня в году (от 1 до 366)
EXTRACT(*type* FROM *datetime*)	Принимает дату и время суток *datetime* и возвращает часть, определяемую параметром *type*
FROM_DAYS(*N*)	Принимает число дней *N*, прошедших с нулевого года, и возвращает дату в формате 'YYYY-MM-DD'
FROM_UNIXTIME(*unix_timestamp*), FROM_UNIXTIME(*unix_timestamp*, *format*)	Принимает число секунд, прошедших с полуночи 1 января 1970 года, и возвращает дату и время суток в виде строки 'YYYY-MM-DD hh:mm:ss' или в виде числа YYYYMMDDhhmmss в зависимости от того, вызвана функция в строковом или числовом контексте
GET_FORMAT(*date* | *time* | *datetime*, 'EUR' | 'USA' | 'JIS' | 'ISO' | 'INTERNAL')	Функция возвращает строку форматирования, соответствующую одному из пяти стандартов времени

Таблица 6.2 *(продолжение)*

Функция	Описание
HOUR(*datetime*)	Возвращает значение часа (от 0 до 23) для времени *datetime*
LAST_DAY(*datetime*)	Принимает в качестве параметра значение даты *datetime* в кратком 'YYYY-MM-DD' или расширенном формате 'YYYY-MM-DD hh:mm:ss' и возвращает дату в кратком формате 'YYYY-MM-DD', день в которой выставлен на последний день текущего месяца
MAKEDATE(*year*, *dayofyear*)	Принимает в качестве параметров год *year*, номер дня в году *dayofyear* и возвращает дату в формате 'YYYY-MM-DD'
MAKETIME(*hour*, *minute*, *second*)	Принимает три параметра: час *hour*, минуты *minute* и секунды *second*. В качестве результата функция возвращает время суток в формате 'hh:mm:ss'
MICROSECOND(*datetime*)	Извлекает из временного значения *datetime* микросекунды
MINUTE(*datetime*)	Возвращает значение минут (от 0 до 59) для времени суток *datetime*
MONTH(*datetime*)	Возвращает числовое значение месяца года (от 1 до 12) для даты *datetime*
MONTHNAME(*datetime*)	Возвращает строку с названием месяца для даты *datetime*
NOW(), CURRENT_TIME, CURRENT_TIME(), CURRENT_TIMESTAMP, CURRENT_TIMESTAMP(), LOCALTIME(), LOCALTIME, LOCALTIMESTAMP(), LOCALTIMESTAMP, SYSDATE()	Возвращает текущие дату и время в виде строки в формате 'YYYY-MM-DD hh:mm:ss' или в виде числа YYYYMMDDhhmmss, в зависимости от того, вызывается функция в строковом или числовом контексте
PERIOD_ADD(*period*, *N*)	Добавляет *N* месяцев к значению даты *period*. Аргумент *period* должен быть представлен в числовом формате YYYYMM или YYMM. Передача в качестве аргумента даты в любом другом формате приводит к непредсказуемому результату
PERIOD_DIFF(*period1*, *period2*)	Вычисляет разницу в месяцах между двумя датами *period1* и *period2*, которые представлены в числовом формате YYYYMM или YYMM

Таблица 6.2 *(продолжение)*

Функция	**Описание**
QUARTER(*datetime*)	Возвращает значение квартала года (от 1 до 4) для даты *datatime*, которая передается в формате 'YYYY-MM-DD' или 'YYYY-MM-DD hh:mm:ss'
SECOND(*time*)	Возвращает число секунд для времени суток *time*, которое задается либо в виде строки 'hh:mm:ss', либо числа hhmmss
SEC_TO_TIME(*seconds*)	Принимает число секунд *seconds*, прошедших от начала суток, и возвращает время в формате 'hh:mm:ss' или hhmmss, в зависимости от того, вызывается функция в строковом или числовом контексте
STR_TO_DATE(*str*, *format*)	Принимает дату и время суток в виде строки *str*, соответствующей формату *format*, и возвращает время в формате MySQL 'YYYY-MM-DD hh:mm:ss'
SUBDATE(*date*, INTERVAL *expr interval*), DATE_SUB(*date*, INTERVAL *expr interval*)	Возвращает дату *date*, из которой вычитается временной интервал, определяемый вторым параметром
SUBTIME(*datetime*, *time*)	Вычитает из временной величины *datetime* время суток *time* и возвращает результат
TIME(*datetime*)	Извлекает время суток 'hh:mm:ss' из даты и времени *datetime*
TIMEDIFF(*expr1*,*expr2*)	Возвращает разницу между временными значениями, заданными параметрами *expr1* и *expr2*
TIMESTAMP(*expr*)	Принимает в качестве аргумента дату и время суток в полном или кратком форматах и возвращает полный вариант в формате 'YYYY-MM-DD hh:mm:ss'
TIMESTAMPADD(*interval*, *int_expr*, *datetime_expr*)	Прибавляет к дате и времени суток *datetime_expr* в полном или кратком формате временной интервал *int_expr*, единицы измерения которого задаются параметром *interval*
TIMESTAMPDIFF(*interval*, *datetime_expr1*, *datetime_expr2*)	Возвращают разницу между двумя датами *datetime_expr1* или *datetime_expr2*, заданными в кратком 'YYYY-MM-DD' или полном 'YYYY-MM-DD hh:mm:ss' форматах. Единицы измерения, в которых функция возвращает результат, задаются параметром *interval*
TIME_FORMAT(*time*, *format*)	Форматирует время *time*, которое задается в виде 'hh:mm:ss' согласно строке форматирования *format*

Таблица 6.2 *(продолжение)*

Функция	Описание
TIME_TO_SEC(*time*)	Принимает в качестве аргумента время суток *time* в формате 'hh:mm:ss' и возвращает число секунд, прошедших с начала суток
TO_DAYS(*date*)	Принимает дату *date* в полном 'YYYY-MM-DD hh:mm:ss' или кратком 'YYYY-MM-DD' формате и возвращает число дней, прошедших с нулевого года
UNIX_TIMESTAMP(), UNIX_TIMESTAMP(*datetime*)	Вариант без параметра *datetime* возвращает число секунд, прошедших с полуночи 1 января 1970 года. Кроме этого, функция может принимать необязательный параметр *datetime*, определяющий дату в кратком 'YYYY-MM-DD' или полном 'YYYY-MM-DD hh:mm:ss' форматах. В этом случае возвращает разницу в секундах между 1 января 1970 года и указанной в *datetime* датой
UTC_DATE()	Возвращает текущую дату в виде строки 'YYYY-MM-DD' или числа YYYYMMDD в зависимости от того, в каком контексте вызывается функция. В отличие от функций NOW() или CURDATE(), возвращающих локальную дату, в UTC_DATE() время исчисляется по Гринвичу
UTC_TIME()	Возвращает текущее время суток в виде строки 'hh:mm:ss' или числа hhmmss в зависимости от того, в каком контексте вызывается функция. В отличие от функций NOW() или CURTIME(), возвращающих локальную дату, в UTC_DATE() время исчисляется по Гринвичу
UTC_TIMESTAMP()	Возвращает дату и время суток в виде строки 'YYYY-MM-DD hh:mm:ss' или в виде числа YYYYMMDDhhmmss в зависимости от того, в каком контексте вызывается функция. В отличие от функции NOW(), возвращающей локальную дату, в UTC_DATE() время исчисляется по Гринвичу
WEEK(*date*)	Возвращает номер недели (от 0 до 53) в году для даты *date*
WEEKDAY(*date*)	Принимает дату *date* в полном 'YYYY-MM-DD hh:mm:ss' или кратком 'YYYY-MM-DD' форматах и возвращает номер дня недели (0 для понедельника, 1 для вторника, ..., 6 для воскресенья)
WEEKOFYEAR(*datetime*)	Возвращает порядковый номер недели в году (от 1 до 53) для даты *datetime*
YEAR(*datetime*)	Возвращает год (от 1000 до 9999) для даты *datetime*

***Таблица 6.2** (окончание)*

Функция	Описание
YEARWEEK(*date*)	Возвращает число в формате YYYYWW, представляющее год и номер недели (от 0 до 53) в году, соответствующее дате *date*. Предполагается, что неделя начинается с воскресенья (как это принято на Западе)

6.2.1. Сложение и вычитание интервалов

Функция ADDDATE(*date*, INTERVAL *expr type*) возвращает время *date*, к которому прибавлен временной интервал, определяемый вторым параметром.

Замечание

Функция ADDDATE() имеет синоним DATE_ADD().

Второй параметр начинается с ключевого слова INTERVAL, за которым следует число и единицы измерения *type*. Значения, которые может принимать параметр *type*, приведены в первом столбце табл. 4.1.

Листинг 6.11. Использование функции ADDDATE()

```
SELECT ADDDATE('2007-03-20',INTERVAL 10 DAY);
+--------------------------------------+
| ADDDATE('2007-03-20',INTERVAL 10 DAY) |
+--------------------------------------+
| 2007-03-30                           |
+--------------------------------------+
```

В листинге 6.11 приводится пример использования функции ADDDATE(). К дате 20 марта 2007 года добавляется 10 дней, в результате чего возвращается дата 30 марта 2007 года.

Функция ADDDATE() допускает и отрицательные значения (листинг 6.12). В этом случае происходит вычитание временного интервала.

Листинг 6.12. Работа с отрицательными значениями

```
SELECT ADDDATE('2007-03-20',INTERVAL -10 DAY);
+---------------------------------------+
| ADDDATE('2007-03-20',INTERVAL -10 DAY) |
+---------------------------------------+
| 2007-03-10                            |
+---------------------------------------+
```

Выражение *expr* можно задавать как в виде числа, так и в виде строки, *expr* содержит и нечисловые значения (пробел, дефис, двоеточие). Вместо пробела, двоеточия или дефиса можно устанавливать любой разделитель символов.

Замечание

Типы временных интервалов DAY_MICROSECOND, HOUR_MICROSECOND, MINUTE_MICROSECOND, SECOND_MICROSECOND и MICROSECOND были введены в MySQL 4.1.1, а QUARTER и WEEK — начиная с 5.0.0.

Работа с типом временного интервала MICROSECOND продемонстрирована в листинге 6.13.

Листинг 6.13. Использование типа MICROSECOND

```
SELECT ADDDATE('2007-03-20 00:00:00',INTERVAL 1 MICROSECOND);
+-----------------------------------------------------+
| ADDDATE('2007-03-20 00:00:00',INTERVAL 1 MICROSECOND) |
+-----------------------------------------------------+
| 2007-03-20 00:00:00.000001                          |
+-----------------------------------------------------+
SELECT ADDDATE('2007-03-20 00:00:00',
INTERVAL 1000000 MICROSECOND);
+-------------------------------------------------------------+
| ADDDATE('2007-03-20 00:00:00',INTERVAL 1000000 MICROSECOND) |
+-------------------------------------------------------------+
| 2007-03-20 00:00:01                                         |
+-------------------------------------------------------------+
```

Как видно из листинга, если используемого формата времени недостаточно для отображения изменений, полученных в результате сложения, то MySQL автоматически расширяет тип данных. В листинге 6.14 демонстрируется использование интервала MINUTE_MICROSECOND.

Листинг 6.14. Использование типа MINUTE_MICROSECOND

```
SELECT ADDDATE('2007-03-20',
INTERVAL '59:10.20' MINUTE_MICROSECOND);
+-------------------------------------------------------------+
| ADDDATE('2007-03-20',INTERVAL '59:10.20' MINUTE_MICROSECOND) |
+-------------------------------------------------------------+
| 2007-03-20 00:59:10.200000                                  |
+-------------------------------------------------------------+
```

При использовании типа WEEK (неделя), показанного в листинге 6.15, и QUARTER (квартал), пример которого приведен в листинге 6.16, происходит увеличение даты на интервалы, кратные семи дням и трем месяцам соответственно.

Листинг 6.15. Использование типа WEEK

```
SELECT ADDDATE('2007-03-20',INTERVAL 52 WEEK);
+---------------------------------------+
| ADDDATE('2007-03-20',INTERVAL 52 WEEK) |
+---------------------------------------+
| 2008-03-19                            |
+---------------------------------------+
SELECT ADDDATE('2007-03-20',INTERVAL 1 WEEK);
+---------------------------------------+
| ADDDATE('2007-03-20',INTERVAL 1 WEEK) |
+---------------------------------------+
| 2007-03-27                            |
+---------------------------------------+
```

Листинг 6.16. Использование типа QUARTER

```
SELECT ADDDATE('2007-03-20',INTERVAL 1 QUARTER);
+-----------------------------------------+
| ADDDATE('2007-03-20',INTERVAL 1 QUARTER) |
+-----------------------------------------+
| 2007-06-20                              |
+-----------------------------------------+
SELECT ADDDATE('2007-03-20',INTERVAL 3 QUARTER);
+-----------------------------------------+
| ADDDATE('2007-03-20',INTERVAL 3 QUARTER) |
+-----------------------------------------+
| 2007-12-20                              |
+-----------------------------------------+
```

Помимо функции ADDDATE(), для сложения даты с временным интервалом допустимо использование оператора + (листинг 6.17), который действует аналогично функции ADDDATE().

Листинг 6.17. Использование оператора + с временными типами данных

```
SELECT '2007-03-20' + INTERVAL '10 12:59:59.14' DAY_MICROSECOND;
+-----------------------------------------------------------+
| '2007-03-20' + INTERVAL '10 12:59:59.14' DAY_MICROSECOND |
+-----------------------------------------------------------+
| 2007-03-30 12:59:59.140000                                |
+-----------------------------------------------------------+
```

При использовании составных форматов если значение одного из параметров выходит за границы (например, указывается 100 минут), то избыток автоматически прибавляется к старшему разряду (1 час 40 минут) (листинг 6.18).

Замечание

Наряду с оператором + можно использовать оператор – для вычитания временного интервала из даты.

Листинг 6.18. Выход за границы часа

```
SELECT '2007-03-20' + INTERVAL '10 12:100:59.14' DAY_MICROSECOND;
+-----------------------------------------------------------+
| '2007-03-20' + INTERVAL '10 12:100:59.14' DAY_MICROSECOND |
+-----------------------------------------------------------+
| 2007-03-30 13:40:59.140000                                |
+-----------------------------------------------------------+
SELECT '2005-03-20' + INTERVAL 30 MONTH;
+----------------------------------+
| '2005-03-20' + INTERVAL 30 MONTH |
+----------------------------------+
| 2007-09-20                       |
+----------------------------------+
```

Помимо развернутой формы синтаксиса, у функции `ADDDATE()` имеется краткая форма `ADDDATE(expr, days)`. Во втором параметре *days* отсутствует ключевое слово `INTERVAL`, а тип временного интервала приравнивается к `DAY` (листинг 6.19).

Замечание

Краткая форма функции `ADDDATE()` появилась в MySQL, начиная с версии 4.1.1.

Листинг 6.19. Использование краткой формы функции `ADDDATE()`

```
SELECT ADDDATE('2007-03-20',6), ADDDATE('2007-03-20',-6);
+-------------------------+-------------------------+
| ADDDATE('2007-03-20',6) | ADDDATE('2007-03-20',-6) |
+-------------------------+-------------------------+
| 2007-03-26              | 2007-03-14              |
+-------------------------+-------------------------+
```

Функция `SUBDATE(date, INTERVAL expr interval)` возвращает дату *date*, из которой вычитается временной интервал, определяемый вторым параметром. Функция полностью аналогична `ADDDATE()`, за исключением того, что `SUBDATE()` производит над аргументом *date* операцию вычитания, а не сложения (листинг 6.20).

Листинг 6.20. Использование функции SUBDATE()

```
SELECT ADDDATE('07-10-10', INTERVAL -5 DAY);
+-------------------------------------+
| ADDDATE('07-10-10', INTERVAL -5 DAY) |
+-------------------------------------+
| 2007-10-05                          |
+-------------------------------------+
SELECT SUBDATE('07-10-10', INTERVAL 5 DAY);
+------------------------------------+
| SUBDATE('07-10-10', INTERVAL 5 DAY) |
+------------------------------------+
| 2007-10-05                         |
+------------------------------------+
```

6.2.2. Заполнение календарных значений

Наиболее часто используемой функцией по праву является функция NOW() или ее синонимы, которые помещают в календарный столбец текущее значение. В листинге 6.21 создается таблица tbl, которая содержит два календарных столбца: putdatetime и putdate, имеющие типы DATETIME и DATE соответственно.

Листинг 6.21. Создание таблицы tbl

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  putdatetime DATETIME NOT NULL,
  putdate DATE NOT NULL
);
```

В листинге 6.22 приводится пример вставки в таблицу tbl новой записи, в поле putdate помещается текущая дата, а в поле putdatetime — время, превышающее текущее на 10 часов.

Листинг 6.22. Вставка календарных значений при помощи функции NOW()

```
INSERT INTO tbl VALUES (1, NOW() + INTERVAL 10 HOUR, NOW());
SELECT * FROM tbl;
+----+---------------------+------------+
| id | putdatetime         | putdate    |
+----+---------------------+------------+
|  1 | 2007-01-12 11:55:17 | 2007-01-12 |
+----+---------------------+------------+
```

Представленный в листинге 6.22 INSERT-запрос аналогичен по действию запросу из листинга 6.23.

Листинг 6.23. Вставка календарных значений без использования функции NOW()

```
INSERT INTO tbl VALUES (1, '2007-01-12 01:55:17' + INTERVAL 10 HOUR,
                        '2007-01-12');
```

6.2.3. Форматирование календарных значений

Функция DATE_FORMAT(*date*, *format*) форматирует время *date* в соответствии со строкой *format*. Более подробно ее синтаксис рассматривается в *разд. 5.9*. Здесь рассмотрим наиболее часто встречающуюся задачу преобразования календарного значения из MySQL-формата '2007-01-12 11:55:17' в привычный формат '12.01.2007 11:55'. Осуществить такое преобразование можно при помощи запроса, представленного в листинге 6.24.

Листинг 6.24. Преобразование даты при помощи функции DATE_FORMAT()

```
SELECT DATE_FORMAT(putdatetime,'%d.%m.%Y %H:%i') AS putdatetime FROM tbl;
+------------------+
| putdatetime      |
+------------------+
| 12.01.2007 11:55 |
+------------------+
```

6.2.4. Вычисление возраста человека

Пусть имеется таблица работников tbl, состоящая из трех столбцов:

- id — первичный ключ таблицы;
- fio — фамилия, имя, отчество работника;
- putdate — дата рождения.

В листинге 6.25 представлен запрос CREATE TABLE, создающий таблицу tbl.

Листинг 6.25. Создание таблицы tbl

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  fio TINYTEXT NOT NULL,
  putdate DATE NOT NULL
);
```

```
INSERT INTO tbl VALUES (1, 'Тимирязев В.К.', '1972-12-02');
INSERT INTO tbl VALUES (2, 'Малышев Ю.Г.', '1976-06-24');
INSERT INTO tbl VALUES (3, 'Абрамов К.Т.', '1968-10-24');
INSERT INTO tbl VALUES (4, 'Ганюшкин В.В.', '1986-12-07');
```

Пусть стоит задача вычисления текущего возраста работника. Один из вариантов состоит в преобразовании даты рождения и текущей даты в дни при помощи функции `TO_DAYS()` и делению на число 365,25 (листинг 6.26). В году 365 дней, дробное число 0,25 призвано компенсировать високосные года, которые случаются раз в четыре года.

Листинг 6.26. Вычисление возраста работников

```
SELECT fio, (TO_DAYS(NOW()) — TO_DAYS(putdate))/365.25 AS putdate
FROM tbl;
+----------------+---------+
| fio            | putdate |
+----------------+---------+
| Тимирязев В.К. | 34.1109 |
| Малышев Ю.Г.   | 30.5517 |
| Абрамов К.Т.   | 38.2177 |
| Ганюшкин В.В.  | 20.0986 |
+----------------+---------+
```

Для того чтобы избавиться от дробной части, можно воспользоваться функцией `FLOOR()` (листинг 6.27).

Листинг 6.27. Отбрасывание дробной части

```
SELECT fio, FLOOR((TO_DAYS(NOW()) — TO_DAYS(putdate))/365.25) AS putdate
FROM tbl;
+----------------+---------+
| fio            | putdate |
+----------------+---------+
| Тимирязев В.К. |      34 |
| Малышев Ю.Г.   |      30 |
| Абрамов К.Т.   |      38 |
| Ганюшкин В.В.  |      20 |
+----------------+---------+
```

Представленный в листинге 6.27 подход является приблизительным, более точный результат можно получить при помощи специальной функции `TIMESTAMPDIFF()`, введенной в MySQL в версии 5.0.0.

Листинг 6.28. Использование функции `TIMESTAMPDIFF()`

```
SELECT fio, TIMESTAMPDIFF(YEAR, putdate, NOW()) AS putdate FROM tbl;
+----------------+---------+
| fio            | putdate |
+----------------+---------+
| Тимирязев В.К. |      34 |
| Малышев Ю.Г.   |      30 |
| Абрамов К.Т.   |      38 |
| Ганюшкин В.В.  |      20 |
+----------------+---------+
```

6.2.5. Преобразование даты в UNIXSTAMP-формат

Формат хранения времени UNIXSTAMP является, пожалуй, самым распространенным форматом хранения времени. Его популярность обусловлена удобством операций сложения и вычитания временных интервалов. UNIXSTAMP-формат — это количество секунд, прошедших с полуночи 1 января 1970 года. Для преобразования MySQL-даты в UNIXSTAMP-формат используется функция `UNIX_TIMESTAMP([datetime])`. Функция может принимать необязательный параметр *datetime*, определяющий дату в кратком `'YYYY-MM-DD'` или полном `'YYYY-MM-DD hh:mm:ss'` форматах. В этом случае возвращает разницу в секундах между 1 января 1970 года и указанной в *datetime* датой (листинг 6.29).

Листинг 6.29. Использование функции `UNIX_TIMESTAMP()`

```
SELECT UNIX_TIMESTAMP(),UNIX_TIMESTAMP('1997-10-04 22:23:00');
+------------------+--------------------------------------+
| UNIX_TIMESTAMP() | UNIX_TIMESTAMP('1997-10-04 22:23:00') |
+------------------+--------------------------------------+
|       1113999060 |                            875989380 |
+------------------+--------------------------------------+
```

Функция `FROM_UNIXTIME(unix_timestamp)` решает обратную задачу и принимает число секунд, прошедших с полуночи 1 января 1970 года, и возвращает дату и время суток в виде строки `'YYYY-MM-DD hh:mm:ss'` или в виде числа `YYYYMMDDhhmmss` в зависимости от того, вызвана функция в строковом или числовом контексте (листинг 6.30).

Листинг 6.30. Использование функции FROM_UNIXTIME()

```
SELECT FROM_UNIXTIME(1113837695), FROM_UNIXTIME(0);
+---------------------------+---------------------+
| FROM_UNIXTIME(1113837695) | FROM_UNIXTIME(0)    |
+---------------------------+---------------------+
| 2005-04-18 19:21:35       | 1970-01-01 03:00:00 |
+---------------------------+---------------------+
```

Как видно из листинга 6.30, передача функции значения 0 в качестве аргумента приводит к выводу вместо полуночи 3 часов ночи. Это связано с тем, что время на машине, где расположен сервер MySQL, настроено для третьей временной зоны.

Функция FROM_UNIXTIME(*unix_timestamp*, *format*) может принимать второй параметр *format*, который представляет собой строку, содержащую определители из табл. 5.6. Использование определителей позволяет отформатировать строку, например, так, как это представлено в листинге 6.31.

Листинг 6.31. Форматирование результата в функции FROM_UNIX()

```
SELECT FROM_UNIXTIME(1286481600,'В %Y году контракт заканчивается');
+-------------------------------------------------------------+
| FROM_UNIXTIME(1286481600,'В %Y году контракт заканчивается') |
+-------------------------------------------------------------+
| В 2010 году контракт заканчивается                          |
+-------------------------------------------------------------+
```

6.3. Строковые функции

Строковые функции позволяют осуществлять преобразование строк и строковых столбцов. В табл. 6.3 представлен список строковых функций, доступных в СУБД MySQL.

При использовании строковых функций следует помнить, что позиция в строке начинается с 1, а не с 0, как, например, в С-подобных языках программирования.

Замечание

Если строковой функции передается в качестве одного из аргументов значение NULL, она также возвращает NULL.

Таблица 6.3. *Строковые функции MySQL*

Функция	Описание
ASCII(*str*)	Возвращает значение ASCII-кода первого символа строки *str*
BIN(*N*)	Принимает десятичное число *N* и возвращает его двоичное представление
BIT_LENGTH(*str*)	Принимает строку *str* и возвращает ее длину в битах
CHAR(*N1*,*N2*,...)	Принимает последовательность из ASCII-кодов и возвращает строку, созданную путем объединения соответствующих им символов
CHAR_LENGTH(*str*), CHARACTER_LENGTH(*str*)	Возвращает число символов в строке *str*. В отличие от функции LENGTH() расчет ведется не по байтам, а по символам, поэтому функция корректно работает с многобайтовыми кодировками
CHARSET(*str*)	Возвращает имя кодировки, в которой представлена строка, передаваемая функции в качестве единственного аргумента
COLLATION(*str*)	Возвращает порядок сортировки, установленный для кодировки аргумента *str*
COMPRESS(*string_to_compress*)	Сжимает строку *string_to_compress* при помощи библиотеки zlib
CONCAT(*str1*, *str2*, ...)	Возвращает строку, созданную путем объединения всех аргументов, количество которых не ограничено
CONCAT_WS(*separator*, *str1*, *str2*, ...)	Объединяет аргументы *str1*, *str2* и т. д., помещая между ними разделитель *separator*
CONV(*N*, *from_base*, *to_base*)	Преобразует число *N* из одной системы счисления *from_base* в другую *to_base*. Параметры *from_base* и to_base могут принимать значения от 2 до 36
CONVERT(*expr*, *type*), CONVERT(*expr* USING *charset*)	Преобразует выражения *expr* в тип *type*. Вторая форма функции CONVERT() (с использованием USING) предназначена для преобразования текста из одной кодировки в другую
ELT(*N*, *str1*, *str2*, *str3*, ...)	Возвращает *N*-ю строку из списка аргументов *str1*, *str2*, *str3* и т. д. Так для *N* = 1 возвращается *str1*, для *N* = 2 — *str2* и т. д. Если строка с номером *N* отсутствует или равна NULL, возвращается NULL

Таблица 6.3 *(продолжение)*

Функция	**Описание**
EXPORT_SET(*bits*, *on*, *off*[, *separator*[, *number_of_bits*]])	Возвращает число *bits* в двоичном представлении, 1 в котором заменяется на параметр *on*, а 0 — на параметр *off*. Необязательный параметр *separator* задает разделитель, которым по умолчанию является запятая. Параметр *number_of_bits* позволяет ограничить число возвращаемых функцией символов (по умолчанию равно 64). Если данный параметр не задан, будут выведены все 64 символа, недостающие символы будут установлены в 0 (*off*)
FIELD(*str*, *str1*, *str2*, ...)	Находит строку *str* в списке строк *str1*, *str2*, ... и возвращает номер строки в этом списке (нумерация начинается с 1)
FIND_IN_SET(*str*, *str_list*)	Ищет вхождение строки *str* в список *str_list* и возвращает номер строки в этом списке (нумерация начинается с 1). Если вхождение не найдено, возвращается 0. Параметр *str_list* представляет собой набор строк, разделенных запятыми
FORMAT(*X*, *D*)	Преобразует число *X* в строку с *D* знаками после запятой. Каждые три разряда разделяются запятой
HEX(*N_or_S*)	Возвращает значение аргумента *N_or_S* в виде шестнадцатеричного числа. Аргумент *N_or_S* может быть как числом, так и строкой. В первом случае возвращается обычное шестнадцатеричное число. Когда в качестве аргумента выступает строка, то функция переводит в шестнадцатеричное представление каждый символ строки и объединяет результат
INSERT(*str*, *pos*, *len*, *new_str*)	Возвращает строку *str*, в которой подстрока, начинающаяся с позиции *pos* и имеющая длину *len* символов, заменена подстрокой *new_str*. Функция возвращает строку *str* без изменений, если значение *pos* находится за пределами строки
INSTR(*str*, *substr*)	Возвращает позицию первого вхождения подстроки *substr* в строку *str*
LEFT(*str*, *len*)	Возвращает *len* крайних левых символов строки *str*. Если аргумент *len* превышает число символов в строке *str*, она возвращается без изменений. Если *len* равно NULL или меньше 0, возвращается пустая строка

Таблица 6.3 *(продолжение)*

Функция	Описание
LENGTH(*str*), OCTET_LENGTH(*str*)	Возвращает длину строки *str*. Данная функция может работать некорректно с многобайтными кодировками, т. к. фактически возвращается число байтов в строке
LOAD_FILE(*file_name*)	Читает файл *file_name* и возвращает его содержимое в виде строки. Файл должен быть расположен на сервере, а в параметре *file_name* необходимо указывать абсолютный путь к файлу. Если файл не существует или не может быть прочитан из-за того, что MySQL не обладает достаточными правами доступа, то возвращается NULL
LOCATE(*substr*, *str* [, *pos*]), POSITION(*substr* IN *str*)	Возвращает позицию первого вхождения подстроки *substr* в строку *str*. Если ни одно вхождение не найдено, возвращается 0. Если хотя бы один из аргументов функции равен NULL, функция возвращает NULL. При наличии необязательного аргумента *pos* поиск начинается с позиции, указанной в этом аргументе
LOWER(*str*), LCASE()	Возвращает строку *str*, в которой все символы записаны строчными символами
LPAD(*str*, *len*, *padstr*)	Возвращает строку *str* дополненную слева строкой *padstr* до длины *len*
LTRIM(*str*)	Возвращает строку *str*, в которой удалены все начальные пробелы
MAKE_SET(*N*, *bit0_str*, *bit1_str*, ...)	Представляет число *N* в двоичной форме, создавая список для типа SET, заменяя 1 значениями из списка *bit0_str*, *bit1_str*, ..., при этом для первого разряда используется *bit0_str*, для второго — *bit1_str* и т. д.
MID(*str*, *pos* [, *len*]), SUBSTRING(*str*, *pos*, *len*)	Возвращает подстроку строки *str*, которая начинается с позиции *pos* и имеет длину *len* символов. Если третий параметр *len* не указывается, то подстрока возвращается, начиная с позиции *pos* и до конца строки *str*
OCT(*N*)	Возвращает число *N* в восьмеричной системе счисления
ORD(*str*)	Возвращает ASCII-код крайнего левого символа строки *str*. Аналогична функции ASCII(), но в отличие от последней корректно работает с многобайтными кодировками

Таблица 6.3 *(продолжение)*

Функция	**Описание**
QUOTE(*str*)	Экранирует строку *str* с тем, чтобы получить корректное значение для SQL-выражения. Строка заключается в одинарные кавычки, и каждое вхождение одинарной кавычки ('), обратного слэша (\), значения ASCII-символов NUL и <Ctrl>+<Z> экранируются обратным слэшем
REPEAT(*str*, *count*)	Возвращает строку, полученную из *count* повторений строки *str*. Если аргумент *count* имеет отрицательное значение или ноль, возвращается пустая строка
REPLACE(*str*, *from_str*, *to_str*)	Возвращает строку *str*, в которой все подстроки *from_str* заменены *to_str*
REVERSE(*str*)	Возвращает строку *str*, записанную в обратном порядке
RIGHT(*str*, *len*)	Возвращает *len* крайних правых символов строки *str* или всю строку *str*, если ее длина короче *len*. Функция возвращает пустую строку, если аргумент *len* равен NULL или меньше 1. Если строка *str* равна NULL, функция RIGHT() также возвращает NULL
RPAD(*str*, *len*, *padstr*)	Возвращает строку *str*, дополненную справа строкой *padstr* до длины *len*. Если строка *str* содержит более *len* символов, она усекается до *len*
RTRIM(*str*)	Возвращает строку *str*, в которой удалены все конечные пробелы
SOUNDEX(*str*)	Возвращает произношение строки *str* по алгоритму, описанному Д. Кнутом. Все неалфавитные символы в строке игнорируются, а национальные символы рассматриваются как гласные
SPACE(*N*)	Возвращает строку, состоящую из *N* пробелов, или пустую строку, если *N* имеет отрицательное значение
SUBSTRING_INDEX(*str*, *delim*, *N*)	Возвращает подстроку строки *str*. Если параметр *N* имеет положительное значение, то функция SUBSTRING_INDEX() находит *N*-е вхождение (отсчет слева) подстроки *delim* в строку *str* и возвращает всю часть строки, расположенную слева от подстроки *delim*. Если *N* имеет отрицательное значение, то находится *N*-е вхождение (отсчет справа) подстроки *delim* в строку *str* и возвращается часть строки, расположенная справа от подстроки *delim*

Таблица 6.3 *(окончание)*

Функция	Описание
TRIM([[BOTH | LEADING | TRAILING] [*remstr*] FROM] *str*)	Удаляет из строки *str* расположенные в начале (в конце) символы, указанные в строке *remstr*. Если указано ключевое слово LEADING, удаляются расположенные в начале символы, если TRAILING — в конце, если BOTH — и в начале, и в конце. Если ни одно из ключевых слов не задано, то по умолчанию устанавливается BOTH. Если строка *remstr* не задана, в качестве удаляемых символов выступают пробелы
UNCOMPRESS(*string_to_uncompress*)	Разархивирует строку *string_to_uncompress*, сжатую при помощи функции COMPRESS(). Если в качестве аргумента функции передается обычная, не сжатая строка, возвращается NULL
UNCOMPRESSED_LENGTH(*compressed_string*)	Возвращает длину строки *compressed_string* сжатой функцией COMPRESS() до применения данной функции, т. е. при помощи функции UNCOMPRESSED_LENGTH() можно выяснить, какой длины будет строка после разархивирования ее функцией UNCOMPRESS()
UNHEX(*str*)	Является обратной функции HEX() и интерпретирует каждую пару символов строки *str* как шестнадцатеричный код, который необходимо преобразовать в символ
UPPER(*str*), UCASE(*str*)	Переводит все символы строки *str* в верхний регистр

6.3.1. Изменение кодировки строки

Функция CONVERT(*expr* USING *charset*) преобразует текст из одной кодировки в другую (листинг 6.32).

Листинг 6.32. Преобразование кодировки

```
SELECT CONVERT('База данных MySQL' USING koi8r);
+----------------------------------------+
| CONVERT('База данных MySQL' USING koi8r) |
+----------------------------------------+
| аЮГЮ ДЮММШУ MySQL                       |
+----------------------------------------+
```

Иногда преобразовывать строку не требуется, а необходимо просто указать ее кодировку, это осуществляется при помощи так называемого *представителя* (листинг 6.33).

Листинг 6.33. Альтернативное преобразование кодировки

```
SELECT _koi8r'База данных MySQL';
+-------------------+
| аЮГЮ ДЮММШУ MySQL |
+-------------------+
| аЮГЮ ДЮММШУ MySQL |
+-------------------+
```

Имя кодировки, которое предваряется знаком подчеркивания, помещается перед преобразуемой строкой — результатом является строка в требуемой кодировке. Для того чтобы лучше разобраться в этом способе управления кодировкой, необходимо более подробно рассмотреть поддержку кодировок в СУБД MySQL.

6.3.2. Первые несколько символов строки

Очень часто требуется выбрать из таблицы не весь текст, а лишь несколько первых символов. Пусть используется таблица работников `tbl` (см. листинг 6.25), из которой требуется извлечь первые 5 символов поля `fio`. Эту задачу удобно решать при помощи функции `SUBSTRING()` (листинг 6.34).

Листинг 6.34. Извлечение первых пяти символов столбца

```
SELECT SUBSTRING(fio,1,5) FROM tbl;
+--------------------+
| SUBSTRING(fio,1,5) |
+--------------------+
| Тимир              |
| Малыш              |
| Абрам              |
| Ганюш              |
+--------------------+
```

6.3.3. Извлечение инициалов

Пусть имеется таблица работников `tbl`, которая содержит полные фамилию, имя и отчество в трех разных столбцах (листинг 6.35).

Листинг 6.35. Таблица tbl

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  family TINYTEXT NOT NULL,
  name TINYTEXT NOT NULL,
  patronymic TINYTEXT NOT NULL
);
INSERT INTO tbl VALUES (1, 'Тимирязев', 'Владимир', 'Константинович');
INSERT INTO tbl VALUES (2, 'Малышев', 'Юрий', 'Гаврилович');
INSERT INTO tbl VALUES (3, 'Абрамов', 'Константин', 'Тимурович');
INSERT INTO tbl VALUES (4, 'Ганюшкин', 'Валерий', 'Владимирович');
```

Пусть требуется извлечь только фамилию и инициалы, т. е. вместо "Тимирязев Владимир Константинович" извлечь строку "Тимирязев В. К.". Решить данную задачу можно при помощи запроса, представленного в листинге 6.36.

Листинг 6.36. Извлечение инициалов работников из таблицы tbl

```
SELECT CONCAT(family, ' ',
              SUBSTRING(name,1,1), '.',
              SUBSTRING(patronymic,1,1), '.') AS fio
FROM tbl;
+----------------+
| fio            |
+----------------+
| Тимирязев В.К. |
| Малышев Ю.Г.   |
| Абрамов К.Т.   |
| Ганюшкин В.В.  |
+----------------+
```

Как видно из листинга 6.36, инициалы извлекаются при помощи функции SUBSTRING(), а полученный результат объединяется в одну строку при помощи функции CONCAT().

6.3.4. Изменение регистра строки

Пусть требуется перевести все строки таблицы tbl (см. листинг 6.35) в верхний регистр. Для этого можно воспользоваться запросом, представленным в листинге 6.37.

Листинг 6.37. Преобразование строк в верхний регистр

```
UPDATE tbl SET family = UPPER(family),
               name = UPPER(name),
               patronymic = UPPER(patronymic);
```

```
SELECT * FROM tbl;
+----+-----------+------------+----------------+
| id | family    | name       | patronymic     |
+----+-----------+------------+----------------+
|  1 | ТИМИРЯЗЕВ | ВЛАДИМИР   | КОНСТАНТИНОВИЧ |
|  2 | МАЛЫШЕВ   | ЮРИЙ       | ГАВРИЛОВИЧ     |
|  3 | АБРАМОВ   | КОНСТАНТИН | ТИМУРОВИЧ      |
|  4 | ГАНЮШКИН  | ВАЛЕРИЙ    | ВЛАДИМИРОВИЧ   |
+----+-----------+------------+----------------+
```

Как видно из листинга 6.37, преобразование строки в верхний регистр осуществляется при помощи функции `UPPER()`, обратную задачу можно решить при помощи функции `LOWER()` (листинг 6.38).

Листинг 6.38. Преобразование строк в нижний регистр

```
UPDATE tbl SET family = LOWER(family),
               name = LOWER(name),
               patronymic = LOWER(patronymic);
SELECT * FROM tbl;
+----+-----------+------------+----------------+
| id | family    | name       | patronymic     |
+----+-----------+------------+----------------+
|  1 | тимирязев | владимир   | константинович |
|  2 | малышев   | юрий       | гаврилович     |
|  3 | абрамов   | константин | тимурович      |
|  4 | ганюшкин  | валерий    | владимирович   |
+----+-----------+------------+----------------+
```

В листинге 6.39 в верхний регистр переводится лишь первая буква каждой строки в таблице `tbl`.

Листинг 6.39. Перевод в верхний регистр первого символа

```
UPDATE tbl SET
family = CONCAT(UPPER(SUBSTRING(family,1,1)),SUBSTRING(family,2)),
name = CONCAT(UPPER(SUBSTRING(name,1,1)),SUBSTRING(name,2)),
patronymic = CONCAT(UPPER(SUBSTRING(patronymic,1,1)),
                    SUBSTRING(patronymic,2));
SELECT * FROM tbl;
+----+-----------+------------+----------------+
| id | family    | name       | patronymic     |
+----+-----------+------------+----------------+
|  1 | Тимирязев | Владимир   | Константинович |
|  2 | Малышев   | Юрий       | Гаврилович     |
```

```
|  3 | Абрамов    | Константин | Тимурович      |
|  4 | Ганюшкин   | Валерий    | Владимирович   |
+----+-----------+------------+----------------+
```

6.3.5. Сжатие строки

Строку в таблице можно подвергнуть сжатию при помощи функции COMPRESS(), а при выборке осуществить распаковку посредством функции UNCOMPRESS(). В листинге 6.40 приводится пример использования функции COMPRESS().

Листинг 6.40. Использование функции COMPRESS()

```
SELECT LENGTH(COMPRESS(REPEAT('a',1000))) AS a1000,
LENGTH(COMPRESS(REPEAT('a',16))) AS a16,
LENGTH(COMPRESS('')) AS a0;
+-------+-----+----+
| a1000 | a16 | a0 |
+-------+-----+----+
|    21 |  15 |  0 |
+-------+-----+----+
```

Пустая строка не подвергается сжатию и возвращается без изменений. Если функция заканчивается пробельными символами, необходимо добавить завершающую точку (.), в противном случае они будут отброшены.

Так как сжатые данные представляют собой бинарный код, их рекомендуется хранить в столбцах типа BLOB, а не TEXT, CHAR или VARCHAR, где они могут быть повреждены кодировкой.

Замечание

Следует отметить, что эффект сжатия заметен только для длинных текстовых строк с большим числом повторяющихся элементов. Применение функции COMPRESS() к коротким текстовым строкам приводит только к увеличению их длины. Это связано с тем, что при сжатии данных добавляются заголовки, позволяющие произвести разархивирование. Для коротких строк длина заголовков превышает эффект от сжатия.

В листинге 6.41 приводится пример использования функции UNCOMPRESS().

Листинг 6.41. Использование функции UNCOMPRESS()

```
SELECT UNCOMPRESS(COMPRESS('any string'));
+-----------------------------------+
| UNCOMPRESS(COMPRESS('any string')) |
```

```
+-----------------------------------+
| any string                        |
+-----------------------------------+
SELECT UNCOMPRESS('any string');
+--------------------------+
| UNCOMPRESS('any string') |
+--------------------------+
| NULL                     |
+--------------------------+
```

Как видно из листинга 6.41, если в качестве аргумента функции передается обычная, не сжатая строка, возвращается `NULL`.

6.3.6. Загрузка строки из файла

При помощи функции `LOAD_FILE(file_name)` можно прочитать содержимое файла *file_name* и вернуть его в виде строки. В листинге 6.42 при помощи функции `LOAD_FILE()` извлекается содержимое файла C:\my.ini.

Листинг 6.42. Извлечение содержимого файла C:\my.ini

```
SELECT LOAD_FILE("C:/my.ini")\G
*************************** 1. row ***************************
LOAD_FILE("C:/my.ini"): [mysqld]

# The TCP/IP Port the MySQL Server will listen on
port=3306

# Path to installation directory. All paths are usually resolved relative
# to this.
basedir="C:/mysql/"

#Path to the database root
datadir="C:/mysql/Data/"

log-bin='elf-bin'
# The default character set that will be used when a new schema or table
# is created and no character set is defined
default-character-set=cp1251
server-id=1

key_buffer_size = 128M
max_allowed_packet = 8M
table_cache = 512
sort_buffer_size = 16M
```

```
read_buffer_size = 1M
join_buffer_size = 1M
query_cache_size = 64M
query_cache_limit = 2M
myisam_sort_buffer_size = 128M
thread_cache = 8
tmp_table_size = 64M
max_connections = 200

[myisamchk]
[mysql]
user = root
```

В листинге 6.43 приводится другой пример: содержимое файла C:\mysql5\my.ini передается функции `LENGTH()`, которая подсчитывает количество символов в строке.

Листинг 6.43. Подсчет количества символов в файле C:\mysql5\my.ini

```
SELECT LENGTH(LOAD_FILE("C:/mysql5/my.ini"));
+--------------------------------------+
| LENGTH(LOAD_FILE("C:/mysql5/my.ini")) |
+--------------------------------------+
|                                 9046 |
+--------------------------------------+
```

Как видно из листинга 6.43, файл C:\mysql5\my.ini занимает 9046 байт. Создадим таблицу `putfile` с единственным полем `content` типа `LONGBLOB` для хранения содержимого файлов (листинг 6.44).

Листинг 6.44. Таблица `putfile`

```
CREATE TABLE putfile (
  content LONGBLOB NOT NULL
);
```

Вставка содержимого файла C:\mysql5\my.ini может выглядеть так, как это представлено в листинге 6.45.

Листинг 6.45. Вставка содержимого файла в таблицу `putfile`

```
INSERT INTO putfile VALUES(LOAD_FILE("C:/mysql5/my.ini"));
```

6.3.7. Извлечение строк, содержащих искомую подстроку в поле типа *SET*

Функция `FIND_IN_SET(str, str_list)` ищет вхождение строки *str* в список *str_list* и возвращает номер строки из этого списка (нумерация начинается с 1). Если вхождение не найдено, возвращается 0. Параметр *str_list* представляет собой набор строк, разделенных запятыми (листинг 6.46).

Замечание

Если один из аргументов функции `FIND_IN_SET()` равен `NULL`, функция возвращает `NULL`.

Листинг 6.46. Использование функции `FIND_IN_SET()`

```
SELECT FIND_IN_SET('show','hide,lock,show');
+-------------------------------------+
| FIND_IN_SET('show','hide,lock,show') |
+-------------------------------------+
|                                   3 |
+-------------------------------------+
SELECT FIND_IN_SET('active','hide,lock,show');
+----------------------------------------+
| FIND_IN_SET('active','hide,lock,show') |
+----------------------------------------+
|                                      0 |
+----------------------------------------+
```

Функция `FIND_IN_SET()` идеально подходит для определения, входит ли значение в строковый набор поля `SET`. Создадим таблицу клиентов `tbl`, каждый из которых может быть подписан на новости о пяти каталогах (листинг 6.47).

Листинг 6.47. Создание таблицы `tbl`

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  family TINYTEXT NOT NULL,
  name TINYTEXT NOT NULL,
  patronymic TINYTEXT NOT NULL,
  subscribe SET('cat1','cat2','cat3','cat4','cat5') NOT NULL
);
INSERT INTO tbl
VALUES (1, 'Тимирязев', 'Владимир', 'Константинович', 'cat1,cat3,cat5');
INSERT INTO tbl
```

```
VALUES (2, 'Малышев', 'Юрий', 'Гаврилович', 'cat1');
INSERT INTO tbl
VALUES (3, 'Абрамов', 'Константин', 'Тимурович',
       'cat1,cat2,cat3,cat4,cat5');
INSERT INTO tbl
VALUES (4, 'Ганюшкин', 'Валерий', 'Владимирович', '');
```

Если клиент подписан на все новости, поле `subscribe` принимает значение `'cat1,cat2,cat3,cat4,cat5'`, если же клиент не подписан на новости ни одного из каталогов, поле `subscribe` его записи в таблице принимает пустую строку `''`. В листинге 6.48 выводятся клиенты, подписанные на новости первого (`cat1`) и третьего (`cat3`) каталогов.

Листинг 6.48. Вывод подписчиков на первый и третий каталоги

```
SELECT * FROM tbl WHERE FIND_IN_SET('cat1',subscribe) AND
                        FIND_IN_SET('cat3',subscribe);
+----+-----------+------------+----------------+-----------------------+
| id | family    | name       | patronymic     | subscribe             |
+----+-----------+------------+----------------+-----------------------+
|  1 | Тимирязев | Владимир   | Константинович |cat1,cat3,cat5         |
|  3 | Абрамов   | Константин | Тимурович      |cat1,cat2,cat3,cat4,cat5|
+----+-----------+------------+----------------+-----------------------+
```

6.4. Функции шифрования

Различают два вида шифрования: обратимое и необратимое. При обратимом шифровании зашифрованный текст может быть подвергнут дешифровке. При необратимом шифровании результатом является хэш, имея который невозможно восстановить исходный текст. В табл. 6.5 представлены функции шифрования, доступные в СУБД MySQL.

Замечание

Если функции передается в качестве одного из аргументов значение `NULL`, она также возвращает `NULL`.

Таблица 6.4. *Функции шифрования*

Функция	Описание
`AES_ENCRYPT(`*`str`*`, `*`key_str`*`)`	Принимает в качестве аргумента *`str`* строку, которую необходимо подвергнуть шифрованию, и секретный ключ *`key_str`*. Входные аргументы могут быть любой длины

Таблица 6.4 *(продолжение)*

Функция	**Описание**
AES_DECRYPT(*crypt_str*, *key_str*)	Принимает в качестве первого аргумента зашифрованную при помощи AES_ENCRYPT() строку. Ключ *key_str* при этом должен совпадать как в первой, так и во второй строках. Если функция AES_DECRYPT() обнаруживает некорректные данные или некорректное заполнение строки, должно возвращаться значение NULL. Однако AES_DECRYPT() вполне может вернуть величину, отличную от NULL, или, возможно, просто "мусор"
ENCODE(*str*, *pass_str*)	Шифрует строку *str*, используя аргумент *pass_str* как секретный ключ
DECODE(*crypt_str*, *pass_str*)	Дешифрует строку *crypt_str*, зашифрованную функцией AES_DECRYPT(). Аргумент *pass_str* используется как секретный ключ
DES_ENCRYPT(*str* [, (*key_number* | *key_string*)])	Шифрует строку *str*, используя аргумент *pass_str* как секретный ключ. В качестве второго необязательного параметра может выступать строка *key_string*, задающая секретный ключ. В случае использования секретного ключа необходимо приводить его в качестве второго параметра и в функции дешифровки DES_DECRYPT(). Вместо секретного ключа можно указать номер *key_number*, принимающий значения от 0 до 9. Номер указывает на запись в ключевом DES-файле сервера, местоположение которого можно задать при старте сервера MySQL в параметре --des-key-file
DES_DECRYPT(*str* [, *key_string*])	Дешифрует строку *str*, зашифрованную при помощи функции DES_ENCRYPT(). Если при шифровании в качестве второго параметра функции DES_ENCRYPT() было передано число или второй параметр был опущен, то параметр *key_string* функции DES_DECRYPT() указывать уже не требуется, т. к. он прописывается в зашифрованную строку. Такой подход, когда секретный ключ хранится на сервере и не передается через сетевое соединение, значительно безопаснее, т. к. значение ключа невозможно извлечь из сетевого трафика. Если второй параметр не указывается, то предполагается, что используется первая строка DES-файла
ENCRYPT(*str*[, *salt*])	Подвергает строку *str* необратимому шифрованию, используя вызов системной функции crypt() UNIX. Если второй необязательный параметр *salt* не указывается, то результат каждый раз получается новым

Таблица 6.4 *(окончание)*

Функция	Описание
MD5(*str*)	Принимает строковый параметр *str* и возвращает 128-битовую контрольную сумму, вычисленную по алгоритму MD5. Возвращаемая величина представляет собой 32-разрядное шестнадцатеричное число, которое уникально для строки, т. е. для строк, отличающихся хотя бы одним символом, результат функции MD5() будет разный. В то же время, для двух одинаковых строк всегда возвращается одинаковый результат
PASSWORD(*str*)	Подвергает необратимому шифрованию данные *str*. Именно эта функция используется для шифрования паролей в MySQL
OLD_PASSWORD(*str*)	Эмулирует работу функции PASSWORD() старых версий MySQL, предшествующих MySQL 4.1
SHA1(*str*)	Вычисляет 160-битовую контрольную сумму по алгоритму SHA1 (Secure Hash Algorithm) для строки *str*

6.4.1. Обратимое шифрование

При обратимом шифровании, как правило, предоставляются две функции для шифровки и расшифровки данных. Наиболее серьезную защиту обеспечивают функции AES_ENCRYPT(*str*, *key_str*) и AES_DECRYPT(*crypt_str*, *key_str*), которые используют официальный алгоритм AES (Advanced Encryption Standard). В нем применяется кодирование с 128-битовым ключом, однако при помощи добавки (патча, patch) к исходному коду СУБД MySQL длину ключа можно увеличить до 256 битов.

Функция AES_ENCRYPT() принимает в качестве аргумента *str* строку, которую необходимо подвергнуть шифрованию, и секретный ключ *key_str*. Входные аргументы могут быть любой длины. Если один из аргументов равен NULL, то результат этой функции также будет иметь значение NULL.

Функция AES_DECRYPT() принимает в качестве первого аргумента зашифрованную при помощи AES_ENCRYPT() строку. Ключ *key_str* при этом должен совпадать как в первой, так и во второй строках. Если функция AES_DECRYPT() обнаруживает некорректные данные или некорректное заполнение строки, должно возвращаться значение NULL. Однако AES_DECRYPT() вполне может вернуть величину, отличную от NULL, или, возможно, просто "мусор".

В листинге 6.49 приводится пример использования функций AES_ENCRYPT() и AES_DECRYPT().

Листинг 6.49. Использование функций `AES_ENCRYPT()` и `AES_DECRYPT()`

```
SELECT AES_ENCRYPT('MySQL','секретный ключ');
+--------------------------------------+
| AES_ENCRYPT('MySQL','секретный ключ') |
+--------------------------------------+
| ...ЗАьZЭj9к•-еП™Mh                    |
+--------------------------------------+
SELECT AES_DECRYPT('...ЗАьZЭj9к•-еП™Mh','секретный ключ');
+-------------------------------------------------+
| AES_DECRYPT('...ЗАьZЭj9к•-еП™Mh','секретный ключ') |
+-------------------------------------------------+
| MySQL                                           |
+-------------------------------------------------+
```

6.4.2. Необратимое шифрование

Функция `MD5()` (листинг 6.50) осуществляет необратимое шифрование данных по алгоритму MD5 (Message-Digest Algorithm) и имеет следующий синтаксис:

`MD5(`*`str`*`)`

Функция принимает строковый параметр *`str`* и возвращает 128-битовую контрольную сумму, вычисленную по алгоритму MD5. Возвращаемая величина представляет собой 32-разрядное шестнадцатеричное число, которое уникально для строки, т. е. для строк, отличающихся хотя бы одним символом, результат функции `MD5()` будет разный. В то же время, для двух одинаковых строк всегда возвращается одинаковый результат (листинг 6.50).

Листинг 6.50. Использование функции `MD5()`

```
SELECT MD5('MySQL'), MD5('MySQL');
+----------------------------------+----------------------------------+
| MD5('MySQL')                     | MD5('MySQL')                     |
+----------------------------------+----------------------------------+
| 62a004b95946bb97541afa471dcca73a | 62a004b95946bb97541afa471dcca73a |
+----------------------------------+----------------------------------+
SELECT MD5('MySQL1'), MD5('MySQL');
+----------------------------------+----------------------------------+
| MD5('MySQL1')                    | MD5('MySQL')                     |
+----------------------------------+----------------------------------+
| fc3dd4ddcb132de1f9552818344e1b09 | 62a004b95946bb97541afa471dcca73a |
+----------------------------------+----------------------------------+
```

Алгоритм MD5 также часто применяется для создания уникального хэш-кода объемных файлов, которые передаются по сети. Загрузив файл, всегда можно

проверить его целостность, вычислив код по алгоритму MD5 и сравнив полученный результат с хэш-кодом, предоставляемым распространителем. Это позволяет отследить повреждения файла, вызванные передачей через сеть, а также предотвращает фальсификацию дистрибутива.

Необратимое шифрование может применяться также для шифрования паролей — в базе данных хранятся не сами пароли, а их MD5-хэши. Во время аутентификации пароль, введенный пользователем, также подвергается обработке функцией `MD5()`. Сравнению подвергаются лишь MD5-хэши. Такая схема позволяет хранить пароли в защищенном виде, и никто, даже администратор базы данных, не получает возможности узнать пользовательские пароли. В последнее время такой вид аутентификации приобретает все большую популярность. Средства подбора пароля по сети практически вышли из употребления из-за применения паузы после ввода некорректного пароля (время подбора пароля в этом случае многократно возрастает) — злоумышленники стараются получить всю базу с паролями. В этом случае злоумышленнику придется восстанавливать методом перебора каждый из зашифрованных паролей в базе данных.

Необратимое шифрование используется и в СУБД MySQL для хранения паролей MySQL-пользователей. Для этого используется специальная функция `PASSWORD()` (листинг 6.51).

Листинг 6.51. Использование функции `PASSWORD()`

```
SELECT PASSWORD('MySQL');
+-------------------------------------------+
| PASSWORD('MySQL')                         |
+-------------------------------------------+
| *1799AB5202FE2E9958365F9B3ECBBF53657254C7 |
+-------------------------------------------+
```

До версии MySQL 4.1 использовался менее защищенный алгоритм шифрования, результат работы которого представлен в листинге 6.52.

Листинг 6.52. Результат работы функции `PASSWORD()` в старых версиях MySQL

```
SELECT PASSWORD('MySQL');
+-------------------+
| PASSWORD('MySQL') |
+-------------------+
| 7c651ebc23eebf89  |
+-------------------+
```

Так как в настоящее время используется большое число баз данных, поля в которых зашифрованы при помощи старой версии функции PASSWORD(), в MySQL 4.1 была введена функция OLD_PASSWORD(), которая эмулирует работу старой версии (листинг 6.53).

Листинг 6.53. Использование функции OLD_PASSWORD()

```
SELECT PASSWORD('MySQL'), OLD_PASSWORD('MySQL');
+-------------------------------------------+-----------------------+
| PASSWORD('MySQL')                         | OLD_PASSWORD('MySQL') |
+-------------------------------------------+-----------------------+
| *1799AB5202FE2E9958365F9B3ECBBF53657254C7 | 7c651ebc23eebf89      |
+-------------------------------------------+-----------------------+
```

Функция PASSWORD() применяется в системе аутентификации MySQL-сервера, поэтому ее не следует использовать в приложениях — лучше воспользоваться функциями MD5() и SHA1().

6.5. Функции управления потоком

Функции управления потоком предназначены для принятия решений в ходе выполнения SQL-инструкций. В табл. 6.5 представлены функции управления потоком, доступные в СУБД MySQL.

Таблица 6.5. *Функции управления потоком*

Функция	Описание
CASE()	Позволяет выбрать значение среди множественного набора данных
IF(*expr1*, *expr2*, *expr3*)	Если выражение *expr1* истинно (не равно 0 или NULL), функция возвращает *expr2*, в противном случае — *expr3*
IFNULL(*expr1*, *expr2*)	Если выражение *expr1* не равно NULL, функция возвращает *expr1*, в противном случае возвращается *expr2*
NULLIF(*expr1*, *expr2*)	Возвращает NULL, если значение *expr1* совпадает с *expr2*, в противном случае возвращается значение *expr1*

6.5.1. Статус записи

Пусть имеется таблица catalogs, состоящая из трех полей: id_catalog — первичный ключ, name — название раздела и total — количество товарных позиций на складе (листинг 6.54).

Листинг 6.54. Таблица catalogs

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL auto_increment,
  name TINYTEXT NOT NULL,
  total Int(11) NOT NULL,
  PRIMARY KEY (id_catalog)
);
INSERT INTO catalogs VALUES (1, 'Процессоры', 15);
INSERT INTO catalogs VALUES (2, 'Материнские платы', 4);
INSERT INTO catalogs VALUES (3, 'Видеоадаптеры', 7);
INSERT INTO catalogs VALUES (4, 'Жесткие диски', 18);
INSERT INTO catalogs VALUES (5, 'Оперативная память', 9);
```

При этом считается, что товарные позиции заканчиваются на складе, если их количество меньше 10 штук. В листинге 6.55 в результирующую таблицу SELECT-запроса добавляется дополнительное поле статуса, в которое выводится строка "Достаточно", если товарных позиций больше 10, и строка "Заканчивается" в противном случае.

Листинг 6.55. Оценка запасов товарных позиций

```
SELECT id_catalog,
       name,
       IF(total > 10, 'Достаточно', 'Заканчивается') AS status
FROM catalogs;
+------------+--------------------+---------------+
| id_catalog | name               | status        |
+------------+--------------------+---------------+
|          1 | Процессоры         | Достаточно    |
|          2 | Материнские платы  | Заканчивается |
|          3 | Видеоадаптеры      | Заканчивается |
|          4 | Жесткие диски      | Достаточно    |
|          5 | Оперативная память | Заканчивается |
+------------+--------------------+---------------+
```

В листинге 6.56 приведенный запрос изменяется таким образом, что при достижении полем total величины менее 5 товарных позиций выводится статус "Критически мало".

Листинг 6.56. Вывод трех статусов

```
SELECT id_catalog,
       name,
```

```
       IF(total > 10,
          'Достаточно',
          IF(total > 5, 'Заканчивается', 'Критически мало')) AS status
FROM catalogs;
+------------+--------------------+-----------------+
| id_catalog | name               | status          |
+------------+--------------------+-----------------+
|          1 | Процессоры         | Достаточно      |
|          2 | Материнские платы  | Критически мало |
|          3 | Видеоадаптеры      | Заканчивается   |
|          4 | Жесткие диски      | Достаточно      |
|          5 | Оперативная память | Заканчивается   |
+------------+--------------------+-----------------+
```

Как видно из листинга 6.56, вывод трех статусов "Достаточно", "Заканчивается" и "Критически мало" достигается путем использования вложенных функций `IF()`. Степень вложения функций не ограничена.

6.5.2. Замена *NULL* на эквиваленты

Функция `IFNULL()`, так же как `IF()`, предназначена для логического выбора и имеет следующий синтаксис:

`IFNULL(`*`expr1`*`, `*`expr2`*`)`

Если выражение *expr1* не равно `NULL`, функция возвращает *expr1*, в противном случае возвращается *expr2*. Функция `IFNULL()` возвращает число или строку в зависимости от контекста, в котором вызывается. В листинге 6.57 приводятся примеры использования функции: везде, где в качестве первого аргумента передается `NULL` или выражение, приводящее к `NULL`, функция возвращает значение второго аргумента.

Листинг 6.57. Использование функции `IFNULL()`

```
SELECT IFNULL(0,1), IFNULL(NULL,1);
+-------------+----------------+
| IFNULL(0,1) | IFNULL(NULL,1) |
+-------------+----------------+
|           0 | 1              |
+-------------+----------------+
SELECT IFNULL(1/0,1), IFNULL(0/1,1);
+---------------+---------------+
| IFNULL(1/0,1) | IFNULL(0/1,1) |
+---------------+---------------+
| 1             | 0.00000       |
+---------------+---------------+
```

Добавим в таблицу `catalogs` (листинг 6.54) поле `url`, в котором будет указываться Web-страница, соответствующая элементу каталога. Если информации о наличии страницы нет, в поле будет помещаться значение `NULL` (листинг 6.58).

Листинг 6.58. Таблица `catalogs`

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  total INT(11) NOT NULL,
  url TINYTEXT,
  PRIMARY KEY (id_catalog)
);
INSERT INTO catalogs VALUES (1, 'Процессоры', 15,
'http://www.site.ru/processor/');
INSERT INTO catalogs VALUES (2, 'Материнские платы', 4, NULL);
INSERT INTO catalogs VALUES (3, 'Видеоадаптеры', 7, NULL);
INSERT INTO catalogs VALUES (4, 'Жесткие диски', 18, NULL);
INSERT INTO catalogs VALUES (5, 'Оперативная память', 9,
'http://www.site.ru/memory/');
```

Для непосвященных пользователей ключевое слово `NULL` ни о чем не говорит, заменим его в результирующей таблице более информативным словом "отсутствует" (листинг 6.59).

Листинг 6.59. Замена ключевого слова `NULL` на слово "отсутствует"

```
SELECT id_catalog, name, IFNULL(url,'отсутствует') AS url FROM catalogs;
+------------+--------------------+------------------------------+
| id_catalog | name               | url                          |
+------------+--------------------+------------------------------+
|          1 | Процессоры         | http://www.site.ru/processor/ |
|          2 | Материнские платы  | отсутствует                  |
|          3 | Видеоадаптеры      | отсутствует                  |
|          4 | Жесткие диски      | отсутствует                  |
|          5 | Оперативная память | http://www.site.ru/memory/   |
+------------+--------------------+------------------------------+
```

6.6. Информационные функции

Информационные функции возвращают различную служебную и системную информацию. В табл. 6.6 представлены информационные функции, доступные в СУБД MySQL.

Таблица 6.6. *Информационные функции MySQL*

Функция	Описание
BENCHMARK(*count*, *expr*)	Повторяет выполнение выражения *expr* заданное количество раз, указанное в аргументе *count*. Функция может использоваться для определения того, насколько быстро MySQL обрабатывает данное выражение. Значение результата, возвращаемого функцией, всегда равно 0. Функция предназначена для использования в клиенте mysql, который сообщает о времени выполнения запроса
CONNECTION_ID()	Возвращает идентификатор (thread_id) для текущего соединения
CURRENT_USER(), CURRENT_USER	Возвращает текущее имя, под которым пользователь аутентифицировался в текущей сессии, и имя хоста, с которого он обращается к серверу
DATABASE(), SCHEMA()	Возвращает имя текущей базы данных
FOUND_ROWS()	Возвращает фактическое количество строк в таблице, если выборка была ограничена конструкцией LIMIT
LAST_INSERT_ID(), LAST_INSERT_ID(*expr*)	Возвращает последнее автоматически сгенерированное значение для столбца, снабженного атрибутом AUTO_INCREMENT. Функция LAST_INSERT_ID() может принимать параметр *expr*, который возвращается функцией и запоминается как следующее значение, которое функция LAST_INSERT_ID() вернет при повторном вызове
ROW_COUNT()	Возвращает число записей, которые подверглись изменению в последнем SQL-запросе. Под изменениями здесь подразумевается обновление, вставка или удаление
USER(), SESSION_USER(), SYSTEM_USER()	Возвращает имя текущего пользователя и имя хоста, с которого он обращается к СУБД MySQL
VERSION()	Возвращает строку, содержащую информацию о версии сервера MySQL

6.6.1. Статусная информация о текущем сеансе

Информационные функции возвращают различную статусную информацию, главным образом о текущем сеансе. В листинге 6.60 возвращается информация о текущем пользователе и выбранной базе данных.

Листинг 6.60. Имя текущего пользователя и выбранной базы данных

```
SELECT USER(), DATABASE();
+----------------+------------+
| USER()         | DATABASE() |
+----------------+------------+
| root@localhost | test       |
+----------------+------------+
```

Если текущая база данных не выбрана, функция `DATABASE()` возвращает `NULL` (листинг 6.61).

Листинг 6.61. Если не выбрана текущая база данных, `DATABASE()` возвращает `NULL`

```
SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| NULL       |
+------------+
```

6.6.2. Информация о сервере

Среди информационных функций особняком стоит функция `VERSION()`, возвращающая версию MySQL-сервера и `BENCHMARK(`*`count, expr`*`)`, которая позволяет измерить производительность MySQL-сервера.

Функция `VERSION()` возвращает строку, содержащую информацию о версии сервера MySQL. Если текущая версия находится в стадии альфа-, бета- или гамма-тестирования, это будет отмечено в отчете функции (листинг 6.62).

Листинг 6.62. Использование функции `VERSION()` в Windows

```
SELECT VERSION();
+---------------------+
| VERSION()           |
+---------------------+
| 5.0.26-community-nt |
+---------------------+
```

Суффикс `nt` характерен для версий, адаптированных для Windows, в операционной системе Linux результат работы функции `VERSION()` может выглядеть так, как это представлено в листинге 6.63.

Листинг 6.63. Использование функции VERSION() в Linux

```
SELECT VERSION();
+-----------------+
| VERSION()       |
+-----------------+
| 5.0.18-standard |
+-----------------+
```

Суффиксы nt и standard указывают на то, что используется обычная версия СУБД MySQL, а не max-вариант сервера, настроенного на максимальную производительность. В зависимости от режима работы сервера, к версии могут прибавляться различные суффиксы, например, суффикс log говорит о том, что сервер работает в режиме записи в бинарный журнал.

6.6.3. Фактическое количество строк в таблице

Оператор SELECT может включать инструкцию LIMIT, которая ограничивает число записей, возвращаемых СУБД MySQL. Тем не менее часто, особенно при построении постраничной навигации, требуется определить число строк, которое бы возвратил оператор SELECT в отсутствие инструкции LIMIT. Для решения этой задачи предназначена функция FOUND_ROWS(). Чтобы получить корректное значение, в состав оператора SELECT необходимо включить параметр SQL_CALC_FOUND_ROWS. После этого можно вызывать функцию FOUND_ROWS() (листинг 6.64).

Листинг 6.64. Использование функции FOUND_ROWS()

```
SELECT SQL_CALC_FOUND_ROWS name FROM catalogs LIMIT 2;
+-------------------+
| name              |
+-------------------+
| Процессоры        |
| Материнские платы |
+-------------------+
SELECT FOUND_ROWS();
+--------------+
| FOUND_ROWS() |
+--------------+
|            5 |
+--------------+
```

Если параметр SQL_CALC_FOUND_ROWS в операторе SELECT будет пропущен, функция вернет количество строк, подсчитанное оператором SELECT. Применительно к листингу 6.64 это число равно 2.

Замечание

Следует отметить, что когда используется опция SQL_CALC_FOUND_ROWS, то СУБД MySQL приходится выполнять дополнительные вычисления для того, чтобы подсчитать число строк в полном результирующем наборе (в отсутствие инструкции LIMIT).

6.7. Разное

В англоязычной литературе этот вид функций обозначается как Miscellaneous-функции, что переводится как смешанные функции, т. е. функции, которые не поддаются классификации. Список функций данной категории представлен в табл. 6.7.

Таблица 6.7. *Miscellaneous-функции MySQL*

Функция	Описание
DEFAULT(*col_name*)	Возвращает значение по умолчанию для столбца *col_name*, которое назначается при помощи ключевого слова DEFAULT
GET_LOCK(*str, timeout*)	Предназначена для получения блокировки для имени *str* со временем ожидания ответа сервера *timeout* секунд и возвращает 1 (истина) в случае успешной установки блокировки на имя *str* и 0 (ложь), если время ожидания ответа превысило величину *timeout*
INET_ATON(*address*)	Принимает IP-адрес *address* и представляет его в виде целого числа
INET_NTOA(*address*)	Принимает IP-адрес в виде числа (результат выполнения функции INET_ATON()) и возвращает адрес в виде строки, состоящей из четырех чисел, разделенных точкой
IS_FREE_LOCK(*str*)	Проверяет, свободна ли блокировка с именем *str*, которая устанавливается при помощи функции GET_LOCK(), рассмотренной ранее. Функция возвращает 1 (истина), если блокировка свободна (никем не используется), и 0 (ложь), если занята
IS_USED_LOCK(*str*)	Проверяет, установлена ли блокировка с именем *str*. В случае, если блокировка установлена, функция возвращает идентификатор соединения клиента, который удерживает блокировку. Если блокировка не установлена, возвращается NULL
NAME_CONST(*name, value*)	Возвращает значение параметра *value*, назначая в качестве имени столбца строку *name*

Таблица 6.7 *(окончание)*

Функция	Описание
RELEASE_LOCK(*str*)	Снимает блокировку с именем *str*, которая была установлена функцией GET_LOCK(). Функция возвращает 1 (истина), если блокировка успешно снята, 0 (ложь), если блокировка установлена другим потоком и не может быть снята, и NULL, если блокировка с таким именем не существует
SLEEP(*duration*)	Останавливает работу на число секунд, указанное в параметре *duration*, которое может принимать в том числе и дробное значение, точность учитывается вплоть до микросекунд. В случае успешного выполнения функция возвращает значение 0, если работа функции прерывается, возвращается 1
UUID()	Возвращает универсальный уникальный идентификатор (Universal Unique Identifier, UUID). Идентификатор UUID реализован в виде числа, которое является глобально уникальным во времени и пространстве. Два вызова функции UUID() вернут два разных значения, если они производятся одновременно на двух разных компьютерах или на одном и том же компьютере в разное время

6.7.1. Преобразование IP-адреса

Функция INET_ATON(*address*) принимает IP-адрес *address* и представляет его в виде целого числа (листинг 6.65). Для числа XXX.YYY.ZZZ.WWW результат функции вычисляется по формуле:

$$\text{XXX} \times 256^3 + \text{YYY} \times 256^2 + \text{ZZZ} \times 256 + \text{WWW}.$$

Замечание

Для того чтобы можно было поместить в целочисленное поле весь диапазон IP-адресов, следует использовать тип BIGINT.

Листинг 6.65. Использование функции INET_ATON()

```
SELECT INET_ATON('62.145.69.10'), INET_ATON('127.0.0.1');
+---------------------------+-----------------------+
| INET_ATON('62.145.69.10') | INET_ATON('127.0.0.1') |
+---------------------------+-----------------------+
|                1049707786 |             2130706433 |
+---------------------------+-----------------------+
```

Начиная с версии 4.1.2, функция INET_ATON() способна принимать IP-адреса в сокращенной форме, как показано в листинге 6.66.

Листинг 6.66. Работа с сокращенной формой IP-адреса

```
SELECT INET_ATON('127.0.01'), INET_ATON('127.1');
+-----------------------+--------------------+
| INET_ATON('127.0.01') | INET_ATON('127.1') |
+-----------------------+--------------------+
|            2130706433 |         2130706433 |
+-----------------------+--------------------+
```

Функция `INET_NTOA(address)` принимает IP-адрес в виде числа (результат выполнения функции `INET_ATON()`) и возвращает адрес в виде строки, состоящей из четырех чисел, разделенных точкой (листинг 6.67).

Листинг 6.67. Работа с функцией `INET_NTOA()`

```
SELECT INET_NTOA(1049707786), INET_NTOA(2130706433);
+-----------------------+-----------------------+
| INET_NTOA(1049707786) | INET_NTOA(2130706433) |
+-----------------------+-----------------------+
| 62.145.69.10          | 127.0.0.1             |
+-----------------------+-----------------------+
```

6.7.2. Блокировка

Установить блокировку с именем *str* со временем ожидания ответа сервера *timeout* секунд можно при помощи функции `GET_LOCK(str, timeout)`. Функция возвращает 1 (истина) в случае успешной установки блокировки на имя *str* и 0 (ложь), если время ожидания ответа превысило величину *timeout*.

Замечание

Функция может вернуть значение `NULL`, если произошла ошибка, такая как переполнение памяти или уничтожение потока.

Снятие блокировки осуществляется при помощи функции `RELEASE_LOCK()`, которая рассматривается далее, повторным вызовом функции `GET_LOCK()`, а также в результате разрыва соединения с сервером.

Блокировка, устанавливаемая функцией `GET_LOCK()`, предназначена для эмуляции блокировок записей. Имена блокируются в глобальном контексте сервера. Если имя блокировано одним клиентом, `GET_LOCK()` блокирует любой запрос другого клиента на получение блокировки с тем же именем. Это позволяет клиентам согласовать попытки доступа к общим ресурсам. После того как клиент закончил работу с ресурсом, он может снять блокировку при помощи функции `RELEASE_LOCK()`. В листинге 6.68 приводится пример работы с блокировками.

Замечание

Имена блокировок *str* могут быть любыми и вовсе не должны совпадать с именами баз данных и таблиц, хотя это не запрещается.

Листинг 6.68. Использование функции GET_LOCK()

```
SELECT GET_LOCK('lock1',10), GET_LOCK('lock2',10);
+----------------------+----------------------+
| GET_LOCK('lock1',10) | GET_LOCK('lock2',10) |
+----------------------+----------------------+
|                    1 |                    1 |
+----------------------+----------------------+
SELECT RELEASE_LOCK('lock1'), RELEASE_LOCK('lock2');
+-----------------------+-----------------------+
| RELEASE_LOCK('lock1') | RELEASE_LOCK('lock2') |
+-----------------------+-----------------------+
|                  NULL |                     1 |
+-----------------------+-----------------------+
```

Следует отметить, что вызов функции `RELEASE_LOCK('lock1')` возвращает `NULL`, т. к. блокировка `'lock1'` была автоматически снята установкой новой блокировки `GET_LOCK('lock2',10)`. Каждый клиент может иметь лишь одну блокировку.

Функция `IS_FREE_LOCK(str)` (листинг 6.69) проверяет, свободна ли блокировка с именем *str*, которая устанавливается при помощи функции `GET_LOCK()`. Функция возвращает 1 (истина), если блокировка свободна (никем не используется), и 0 (ложь), если занята.

Листинг 6.69. Использование функции IS_FREE_LOCK()

```
SELECT GET_LOCK('lock1',10), GET_LOCK('lock2',10);
+----------------------+----------------------+
| GET_LOCK('lock1',10) | GET_LOCK('lock2',10) |
+----------------------+----------------------+
|                    1 |                    1 |
+----------------------+----------------------+
SELECT IS_FREE_LOCK('lock1'), IS_FREE_LOCK('lock2');
+-----------------------+-----------------------+
| IS_FREE_LOCK('lock1') | IS_FREE_LOCK('lock2') |
+-----------------------+-----------------------+
|                     1 |                     0 |
+-----------------------+-----------------------+
```

Функция `IS_USED_LOCK(str)` проверяет, установлена ли блокировка с именем *str* (листинг 6.70). В случае, если блокировка установлена, функция возвращает идентификатор соединения клиента, который удерживает блокировку. Если блокировка не установлена, возвращается `NULL`.

Листинг 6.70. Использование функции `IS_USED_LOCK()`

```
SELECT GET_LOCK('lock1',10), GET_LOCK('lock2',10);
+----------------------+----------------------+
| GET_LOCK('lock1',10) | GET_LOCK('lock2',10) |
+----------------------+----------------------+
|                    1 |                    1 |
+----------------------+----------------------+
SELECT IS_USED_LOCK('lock1'), IS_USED_LOCK('lock2');
+-----------------------+-----------------------+
| IS_USED_LOCK('lock1') | IS_USED_LOCK('lock2') |
+-----------------------+-----------------------+
|                  NULL |                     2 |
+-----------------------+-----------------------+
```

6.7.3. Универсальный уникальный идентификатор компьютера

Функция `UUID()` (листинг 6.71) возвращает *универсальный уникальный идентификатор* (Universal Unique Identifier, UUID), сгенерированный в соответствии со спецификацией "DCE 1.1: удаленный вызов процедур", опубликованной Open Group в 1987 году.

Идентификатор UUID реализован в виде числа, которое является глобально уникальным во времени и пространстве. Два вызова функции `UUID()` вернут два разных значения, если они производятся одновременно на двух разных компьютерах или на одном и том же компьютере в разное время.

Замечание

Функция `UUID()` введена в СУБД MySQL, начиная с версии 4.1.2.

Листинг 6.71. Использование функции `UUID()`

```
SELECT UUID();
+--------------------------------------+
| UUID()                               |
+--------------------------------------+
| 6627c9e4-2fec-1028-9007-551f9bad91cf |
```

```
+--------------------------------------+
SELECT UUID();
+--------------------------------------+
| UUID()                               |
+--------------------------------------+
| 6975f814-2fec-1028-9007-551f9bad91cf |
+--------------------------------------+
```

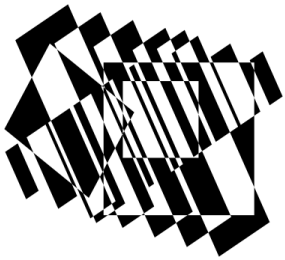
Идентификатор UUID — это 128-разрядное число, представленное в виде строки, состоящей из пяти шестнадцатеричных чисел в формате aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee.

- Первые три числа генерируются на основе текущего времени.
- Четвертое число предохраняет вре́менную уникальность в случае, если значение временной метки теряет монотонность (постоянное увеличение со временем), например, из-за перехода на летнее время и обратно.
- Пятое число — это номер узла IEEE 802, т. е. аппаратный MAC-адрес платы Ethernet, который уникален для каждой платы. Если хост не имеет сетевой платы или отсутствует возможность его получения, подставляется случайное число.

Замечание

В настоящее время MAC-адрес интерфейса принимается во внимание только в средах FreeBSD и Linux. В других операционных системах MySQL использует случайно сгенерированное 48-разрядное число.

ГЛАВА 7



Поиск

По сравнению с текстовыми файлами, СУБД обладает неоспоримым преимуществом — высокой эффективностью операций поиска. В *главе 5* уже затрагивались вопросы поиска, в частности был рассмотрен оператор `LIKE`, позволяющий осуществлять простейшие операции поиска. В данной главе рассматриваются более сложные вопросы поиска: регулярные выражения и полнотекстовый поиск.

7.1. Регулярные выражения

Поиск в соответствии с регулярными выражениями позволяет осуществлять оператор `RLIKE` (или его синоним `REGEXP`). Такой поиск предоставляет значительно более гибкие средства для поиска по сравнению с оператором `LIKE`. Обратной стороной медали является более медленное выполнение операции поиска с использованием регулярных выражений по сравнению с оператором `LIKE`. Операторы `RLIKE` и `REGEXP` имеют следующий синтаксис:

expr REGEXP *pat*
expr RLIKE *pat*

Данные операторы часто используются в выражении `WHERE` и возвращают 1, если выражение *expr* соответствует выражению *pat*, и 0 в противном случае.

Замечание

Помимо операторов `REGEXP` и `RLIKE` существуют обратные формы: `NOT REGEXP` и `NOT RLIKE`, которые возвращают 0, если выражение *expr* соответствует выражению *pat*, и 1 в противном случае.

Регулярное выражение — это шаблон, применяемый к заданному тексту слева направо. Большая часть символов сохраняет свое значение в шаблоне и

означает совпадение с соответствующим символом. Так, регулярное выражение, содержащее обычный текст, например, "грам", соответствует строке, содержащей подстроку "грам". Например, этому регулярному выражению будут соответствовать строки "программирование", "грамм", "грампластинка" и т. п.

Замечание

В настоящий момент существует несколько диалектов регулярных выражений. В СУБД MySQL реализована расширенная версия предложенной Генри Спенсером реализации регулярных выражений, которая ориентирована на соответствие стандарту POSIX.

Пример использования оператора `RLIKE` приведен в листинге 7.1.

Листинг 7.1. Использование оператора `RLIKE`

```
SELECT 'грамм' RLIKE 'грам', 'грампластинка' RLIKE 'грам';
+----------------------+-----------------------------+
| 'грамм' RLIKE 'грам' | 'грампластинка' RLIKE 'грам' |
+----------------------+-----------------------------+
|                    1 |                           1 |
+----------------------+-----------------------------+
SELECT 'программирование' RLIKE 'грам';
+--------------------------------+
| 'программирование' RLIKE 'грам' |
+--------------------------------+
|                              1 |
+--------------------------------+
```

Как видно из листинга 7.1, регулярное выражение `грам` осуществляет поиск по всему тексту, независимо от того, находится ли подстрока "грам" в начале, середине или конце слова.

Часто необходимо привязать регулярное выражение к началу слова, т. е. чтобы регулярное выражение `грам` соответствовало строке, начинающейся с подстроки "грам", например, "грампластинка", но не соответствовало слову "про**грам**мирование". Для этого используется символ `^`, соответствующий началу строки (листинг 7.2).

Листинг 7.2. Использование спецсимвола ^

```
SELECT 'программирование' RLIKE '^грам',
'грампластинка' RLIKE '^грам';
```

```
+----------------------------------+-------------------------------+
| 'программирование' RLIKE '^грам' | 'грампластинка' RLIKE '^грам' |
+----------------------------------+-------------------------------+
|                                0 |                             1 |
+----------------------------------+-------------------------------+
```

Специальный символ `$` помогает привязать регулярное выражение к концу строки (листинг 7.3), т. е. применение символов `^` и `$` позволяет указать, что регулярное выражение должно в точности соответствовать всей строке поиска от начала до конца. Регулярное выражение `^грампластинка$` соответствует строке "грампластинка", но не соответствует строке "грампластинка — это вам не программирование", т. к. после искомого слова идет текст.

Листинг 7.3. Использование спецсимвола $

```
SELECT 'грампластинка' RLIKE '^грампластинка$';
+----------------------------------------+
| 'грампластинка' RLIKE '^грампластинка$' |
+----------------------------------------+
|                                      1 |
+----------------------------------------+
SELECT 'грампластинка — это вам не программирование'
RLIKE '^грампластинка$';
+----------------------------------------------------------------------+
| 'грампластинка — это вам не программирование' RLIKE '^грампластинка$' |
+----------------------------------------------------------------------+
|                                                                    0 |
+----------------------------------------------------------------------+
```

Замечание

Регулярное выражение `^$` соответствует пустой строке, но не соответствует `NULL`. Для проверки записей на равенство `NULL` следует использовать операторы `IS NULL` и `IS NOT NULL`.

В листинге 7.4 представлен дамп таблицы `tbl`, в которой хранятся полные фамилии, имена и отчества.

Листинг 7.4. Создание таблицы `tbl`

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  family TINYTEXT NOT NULL,
  name TINYTEXT NOT NULL,
  patronymic TINYTEXT NOT NULL
);
```

```
INSERT INTO tbl VALUES (1, 'Тимирязев', 'Владимир', 'Константинович');
INSERT INTO tbl VALUES (2, 'Малышев', 'Юрий', 'Гаврилович');
INSERT INTO tbl VALUES (3, 'Абрамов', 'Константин', 'Тимурович');
INSERT INTO tbl VALUES (4, 'Ганюшкин', 'Валерий', 'Владимирович');
```

В листинге 7.5 представлен запрос, извлекающий из таблицы `tbl` записи, поле `family` в которых заканчивается символом "в".

Листинг 7.5. Извлечение всех записей, заканчивающихся символом "в"

```
SELECT * FROM tbl WHERE family RLIKE 'в$';
+----+-----------+------------+----------------+
| id | family    | name       | patronymic     |
+----+-----------+------------+----------------+
|  1 | Тимирязев | Владимир   | Константинович |
|  2 | Малышев   | Юрий       | Гаврилович     |
|  3 | Абрамов   | Константин | Тимурович      |
+----+-----------+------------+----------------+
```

Символы `^` и `$` соответствуют началу и концу строки, но гораздо чаще встает задача привязки регулярного выражения к началу и концу слова. Для этого предназначены последовательности `[[:<:]]` и `[[:>:]]`, соответствующие началу и концу слова соответственно (листинг 7.6).

Листинг 7.6. Использование последовательностей `[[:<:]]` и `[[:>:]]`

```
SELECT 'a word a' REGEXP '[[:<:]]word[[:>:]]';
+----------------------------------------+
| 'a word a' REGEXP '[[:<:]]word[[:>:]]' |
+----------------------------------------+
|                                      1 |
+----------------------------------------+
SELECT 'a xword a' REGEXP '[[:<:]]word[[:>:]]';
+----------------------------------------+
| 'a xword a' REGEXP '[[:<:]]word[[:>:]]' |
+----------------------------------------+
|                                       0 |
+----------------------------------------+
```

Символ вертикальной черты `|` (листинг 7.7) применяется в регулярном выражении для задания альтернативных масок:

```
'abc|абв'
```

Этому регулярному выражению соответствует любая строка, содержащая подстроки "abc" или "абв".

Листинг 7.7. Использование специального символа |

```
SELECT 'abc' RLIKE 'abc|абв', 'абв' RLIKE 'abc|абв';
+----------------------+----------------------+
| 'abc' RLIKE 'abc|абв' | 'абв' RLIKE 'abc|абв' |
+----------------------+----------------------+
|                    1 |                    1 |
+----------------------+----------------------+
```

Специальный символ | можно интерпретировать как оператор ИЛИ: при его использовании удовлетворяющими шаблону признаются записи, соответствующие либо правой, либо левой частям.

Если шаблон должен включать символ | или любой другой специальный символ, например, рассмотренные выше ^ и $, то их необходимо экранировать при помощи двойного обратного слэша \\ — в этом случае они теряют свое специальное значение и рассматриваются как обычные символы.

Для задания класса символов применяются квадратные скобки (листинг 7.8). Они ограничивают поиск теми символами, которые в них заключены, например, [abc].

Листинг 7.8. Использование класса символов

```
SELECT 'a' RLIKE '[abc]' AS a,
       'b' RLIKE '[abc]' AS b,
       'c' RLIKE '[abc]' AS c;
+---+---+---+
| a | b | c |
+---+---+---+
| 1 | 1 | 1 |
+---+---+---+
```

Регулярному выражению [abc] соответствует подстрока, содержащая один символ: либо a, либо b, либо c.

Так, для создания регулярного выражения, соответствующего всем буквам русского алфавита, можно, конечно, перечислить все буквы в квадратных скобках. Это допустимо, но утомительно и неэлегантно. Более кратко такое регулярное выражение можно записать следующим образом:

```
'[а-я]'
```

Это выражение соответствует всем буквам русского алфавита, поскольку любые два символа, разделяемые дефисом, задают соответствие диапазону символов, находящихся между ними.

Замечание

При построении регулярного выражения, соответствующего всем буквам русского алфавита, следует учитывать кодировку, в которой набираются символы. Так, если используется кодировка cp1251, следует вместо `[а-я]` писать `[а-яе]`, иначе буква `е` не попадет в диапазон.

Точно таким же образом задается регулярное выражение, соответствующее любому числу:

```
'[0-9]'
```

Это выражение эквивалентно

```
'[0123456789]'
```

В листинге 7.9 приводится пример сравнения русской буквы Л и английской z с регулярным выражением `[а-я]`.

Листинг 7.9. Использование диапазонов в классах символов

```
SELECT 'Л' RLIKE '[а-яе]' AS a;
+---+
| a |
+---+
| 1 |
+---+
SELECT 'z' RLIKE '[а-яе]' AS a;
+---+
| a |
+---+
| 0 |
+---+
```

Выражение вида

```
'[а-яе0-9 ]'
```

соответствует либо букве русского алфавита, либо цифре, либо пробелу.

Если в начале класса помещается символ `^`, то смысл выражения инвертируется (листинг 7.10). Так, выражение

```
'[^0-9]'
```

соответствует любому символу, кроме цифры.

Листинг 7.10. Отрицание в классах символов

```
SELECT '7' RLIKE '[^0-9]' AS number, 'a' RLIKE '[^0-9]' AS str;
+--------+-----+
| number | str |
+--------+-----+
|      0 |   1 |
+--------+-----+
```

Для определения специальных последовательностей внутри строк в СУБД MySQL используется С-нотация, в которой экранирование обычных символов приводит к их специальной интерпретации:

- `\t` — символ табуляции;
- `\f` — конец файла;
- `\n` — символ перевода строки;
- `\r` — символ возврата каретки;
- `\\` — символ обратного слэша `\`.

Кроме классов, которые могут создать разработчики, путем комбинирования отдельных символов и их диапазонов в стандарте POSIX предусмотрены специальные конструкции классов, представленные в табл. 7.1.

Таблица 7.1. *Классы символов POSIX регулярных выражений*

Класс	Описание
`[:alnum:]`	Алфавитно-цифровые символы
`[:alpha:]`	Алфавитные символы
`[:blank:]`	Символы пробела или табуляции
`[:cntrl:]`	Управляющие символы
`[:digit:]`	Десятичные цифры (0—9)
`[:graph:]`	Графические (видимые) символы
`[:lower:]`	Символы алфавита в нижнем регистре
`[:print:]`	Графические или невидимые символы
`[:punct:]`	Знаки препинания
`[:space:]`	Символы пробела, табуляции, новой строки или возврата каретки
`[:upper:]`	Символы алфавита в верхнем регистре
`[:xdigit:]`	Шестнадцатеричные цифры

Классы в табл. 7.1 определяют диапазоны символов, так, например, регулярное выражение `[0-9]` эквивалентно `[[:digit:]]`. В листинге 7.11 демонстрируется применение классов символов `[:digit:]`, `[:alpha:]` и `[:alnum:]` с цифрой 1 и символом "a". Важно отметить, что алфавитные символы удовлетворяют всем алфавитным символам, включая русские и английские.

Листинг 7.11. Использование классов символов

```
SELECT '1' RLIKE '[[:digit:]]', 'a' RLIKE '[[:digit:]]';
+-------------------------+-------------------------+
| '1' RLIKE '[[:digit:]]' | 'a' RLIKE '[[:digit:]]' |
+-------------------------+-------------------------+
|                       1 |                       0 |
+-------------------------+-------------------------+
SELECT '1' RLIKE '[[:alpha:]]', 'a' RLIKE '[[:alpha:]]';
+-------------------------+-------------------------+
| '1' RLIKE '[[:alpha:]]' | 'a' RLIKE '[[:alpha:]]' |
+-------------------------+-------------------------+
|                       0 |                       1 |
+-------------------------+-------------------------+
SELECT '1' RLIKE '[[:alnum:]]', 'a' RLIKE '[[:alnum:]]';
+-------------------------+-------------------------+
| '1' RLIKE '[[:alnum:]]' | 'a' RLIKE '[[:alnum:]]' |
+-------------------------+-------------------------+
|                       1 |                       1 |
+-------------------------+-------------------------+
```

Выражение в квадратных скобках соответствует только одному символу и часто применяется совместно с так называемыми *квантификаторами*. Это символы `?`, `+` и `*`, которые следуют сразу за символом и изменяют число вхождений этого символа в строку:

- `?` — символ либо входит в строку один раз, либо вообще в нее не входит;
- `*` — любое число вхождений символа в строку, в том числе и 0;
- `+` — одно или более число вхождений символа в строку.

Символ `?` позволяет сократить выражения вида (листинг 7.12)

```
'^СУБД|СУБД MySQL$'
```

до

```
'^СУБД( MySQL)?$'
```

Листинг 7.12. Использование квантификатора `?`

```
SELECT 'СУБД MySQL' RLIKE '^СУБД( MySQL)?$',
'СУБД' RLIKE '^СУБД( MySQL)?$';
```

```
+-------------------------------------+-------------------------------+
| 'СУБД MySQL' RLIKE '^СУБД( MySQL)?$' | 'СУБД' RLIKE '^СУБД( MySQL)?$' |
+-------------------------------------+-------------------------------+
|                                   1 |                             1 |
+-------------------------------------+-------------------------------+
```

Символ + обозначает один или несколько экземпляров элемента непосредственно предшествующего элемента. Так, если необходимо найти подстроку, содержащую одну цифру или более, можно воспользоваться выражением вида

```
'[[:digit:]]+'
```

В листинге 7.13 демонстрируется использование квантификатора +, который удовлетворяет последовательности из одного и большего числа символов.

Листинг 7.13. Использование квантификатора +

```
SELECT '1' RLIKE '^[[:digit:]]+$',
'453455234' RLIKE '^[[:digit:]]+$';
+----------------------------+-----------------------------------+
| '1' RLIKE '^[[:digit:]]+$' | '453455234' RLIKE '^[[:digit:]]+$' |
+----------------------------+-----------------------------------+
|                          1 |                                 1 |
+----------------------------+-----------------------------------+
SELECT '' RLIKE '^[[:digit:]]+$',
'45.3455234' RLIKE '^[[:digit:]]+$';
+---------------------------+-----------------------------------+
| '' RLIKE '^[[:digit:]]+$' | '45.3455234' RLIKE '^[[:digit:]]+$' |
+---------------------------+-----------------------------------+
|                         0 |                                 0 |
+---------------------------+-----------------------------------+
```

Символ * используется для любого числа вхождений строки в подстроку, в том числе и нулевого, т. е. регулярное выражение (листинг 7.14)

```
'^[[:digit:]]*$'
```

соответствует либо пустой строке, либо строке, содержащей только цифры, причем их количество не ограничено.

Листинг 7.14. Использование квантификатора *

```
SELECT '1' RLIKE '^[[:digit:]]*$',
       '453455234' RLIKE '^[[:digit:]]*$';
```

```
+---------------------------+-----------------------------------+
| '1' RLIKE '^[[:digit:]]*$' | '453455234' RLIKE '^[[:digit:]]*$' |
+---------------------------+-----------------------------------+
|                         1 |                                 1 |
+---------------------------+-----------------------------------+
SELECT '' RLIKE '^[[:digit:]]*$',
       '45.3455234' RLIKE '^[[:digit:]]*$';
+--------------------------+-------------------------------------+
| '' RLIKE '^[[:digit:]]*$' | '45.3455234' RLIKE '^[[:digit:]]*$' |
+--------------------------+-------------------------------------+
|                        1 |                                   0 |
+--------------------------+-------------------------------------+
```

Помимо круглых и квадратных скобок, в регулярных выражениях также применяются фигурные скобки. Они предназначены для указания числа или диапазона повторения элемента:

- `xy{2}` соответствует строке, в которой за `x` следуют два символа `y`, т. е. `xyy`;
- `xy{2,}` соответствует строке, в которой за `x` следует не менее двух `y` (но может быть и больше);
- `xy{2,6}` соответствует строке, в которой за `x` следует от двух до шести `y`.

Для указания количества вхождений не одного символа, а их последовательности, используются круглые скобки:

- `x(yz){2,6}` соответствует строке, в которой за `x` следует от двух до шести последовательностей `yz`;
- `x(yz)*` соответствует строке, в которой за `x` следует ноль и более последовательностей `yz`.

В листинге 7.15 демонстрируется регулярное выражение, удовлетворяющее числу формата `###.##`. Целая часть может состоять из любого количества цифр, а дробная часть всегда состоит из двух.

Листинг 7.15. Поиск чисел формата ###.##

```
SELECT '123.90' RLIKE '^[[:digit:]]*\\.[[:digit:]]{2}$';
+-------------------------------------------------+
| '123.90' RLIKE '^[[:digit:]]*\\.[[:digit:]]{2}$' |
+-------------------------------------------------+
|                                               1 |
+-------------------------------------------------+
SELECT '123' RLIKE '^[[:digit:]]*\\.[[:digit:]]{2}$';
+-----------------------------------------------+
| '123' RLIKE '^[[:digit:]]*\\.[[:digit:]]{2}$' |
+-----------------------------------------------+
|                                             0 |
+-----------------------------------------------+
```

Как видно из листинга 7.15, если числу `123.90` удовлетворяет представленное регулярное выражение, то обычное целое число `123` данным запросом отбрасывается. Для того чтобы регулярное выражение удовлетворяло сразу обоим числам, необходимо видоизменить запрос так, как это показано в листинге 7.16.

Листинг 7.16. Поиск чисел в форматах ### и ###.##

```
SELECT '123' RLIKE '^[[:digit:]]*(\\.[[:digit:]]{2})?$';
+-------------------------------------------------+
| '123' RLIKE '^[[:digit:]]*(\\.[[:digit:]]{2})?$' |
+-------------------------------------------------+
|                                               1 |
+-------------------------------------------------+
SELECT '123.90' RLIKE '^[[:digit:]]*(\\.[[:digit:]]{2})?$';
+--------------------------------------------------+
| '123.90' RLIKE '^[[:digit:]]*(\\.[[:digit:]]{2})?$' |
+--------------------------------------------------+
|                                                1 |
+--------------------------------------------------+
```

Как видно из листинга 7.16, элемент регулярного выражения, ответственного за дробную часть числа `\\.[[:digit:]]{2}`, группируется при помощи круглых скобок, за которыми следует спецсимвол `?`, требующий вхождения подстроки 0 или 1 раз.

7.2. Полнотекстовый поиск

Помимо рассмотренного в предыдущем разделе поиска по регулярным выражениям, в СУБД MySQL предусмотрен так называемый *режим полнотекстового поиска*, который не требует применения шаблонов. Данный режим предоставляет широкие возможности поиска в тексте и выполняется гораздо быстрее поиска с использованием регулярных выражений и оператора `LIKE` благодаря специальному индексу `FULLTEXT`. Следует помнить, что индексация столбца, в том числе и индексом `FULLTEXT`, требует дополнительного объема памяти для хранения индексов, иногда превышающего объем основных данных в несколько раз, и приводит к замедлению операций вставки и удаления при помощи операторов `INSERT` и `DELETE`.

Полнотекстовый поиск в СУБД MySQL на сегодняшний день поддерживается только для таблиц типа MyISAM и столбцов `CHAR`, `VARCHAR` и `TEXT`.

7.2.1. Индекс *FULLTEXT*

Для использования возможностей полнотекстового поиска необходимо проиндексировать текстовые столбцы таблицы при помощи индекса FULLTEXT. На необходимость индексирования текстового столбца можно указать при создании таблицы в операторе CREATE TABLE.

Определение индекса начинается с ключевого слова FULLTEXT, после которого следует необязательное ключевое слово INDEX. Далее указывается имя индекса (которое может совпадать с именем столбца) и имя индексируемого столбца.

В листинге 7.17 демонстрируется создание таблицы catalogs, в которой столбец name индексируется индексом FULLTEXT.

Замечание

Если индекс создается только по одному столбцу и его имя совпадает с именем столбца, то имя индекса, размещаемое после ключевого слова FULLTEXT, можно опустить.

Листинг 7.17. Индексирование столбца name индексом FULLTEXT

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  PRIMARY KEY (id_catalog),
  FULLTEXT INDEX name (name)
) ENGINE=MyISAM;
DESCRIBE catalogs;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| id_catalog | int(11)  | NO   | PRI | NULL    | auto_increment |
| name       | tinytext | NO   | MUL |         |                |
+------------+----------+------+-----+---------+----------------+
```

Индекс FULLTEXT можно создать сразу по нескольким столбцам. Так, если поиск следует осуществлять одновременно по нескольким столбцам, то в скобках после имени индекса следует перечислить имена индексируемых столбцов через запятую. В листинге 7.18 приводится оператор CREATE TABLE, создающий таблицу catalogs, в которой проиндексированы столбцы name и description.

Листинг 7.18. Индексирование нескольких столбцов

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  description TEXT,
  PRIMARY KEY (id_catalog),
  FULLTEXT INDEX search (name, description)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Имя индекса может быть любым, в листинге 7.18 в качестве такого имени выбрано `search`.

Часто задача индексирования столбцов возникает после того, как данные в таблицу уже добавлены. Это может быть вызвано необходимостью добавления поисковых возможностей в уже готовое приложение или переносом таблицы при помощи текстового дампа с ускоренным заполнением таблицы, после которого требуется повторное построение индекса.

Для добавления индекса `FULLTEXT` в уже готовую таблицу предназначен оператор `ALTER TABLE`. Индекс `FULLTEXT`, как и любой другой индекс, создается при помощи спецификации `ADD`, за которой следует определение индекса. В листинге 7.19 приводятся операторы `ALTER TABLE` для создания ранее рассмотренных индексов в листингах 7.17 и 7.18.

Листинг 7.19. Создание индекса `FULLTEXT` при помощи `ALTER TABLE`

```
ALTER TABLE catalogs ADD FULLTEXT name (name);
ALTER TABLE catalogs ADD FULLTEXT search (name, description);
```

В таблице не накладывается ограничений на число индексов, допустимо создание нескольких индексов, как показано в листинге 7.20.

Листинг 7.20. Создание нескольких индексов

```
ALTER TABLE catalogs ADD FULLTEXT (name),
                     ADD FULLTEXT (description),
                     ADD FULLTEXT search (name, description);
```

Помимо оператора `ALTER TABLE`, для создания индекса `FULLTEXT` в уже существующей таблице можно использовать оператор `CREATE INDEX`, показанный в листинге 7.21.

Листинг 7.21. Использование оператора `CREATE INDEX`

```
CREATE FULLTEXT INDEX name ON catalogs (name);
CREATE FULLTEXT INDEX search ON catalogs (name, description);
```

7.2.2. Конструкция *MATCH (...) AGAINST (...)*

Полнотекстовый поиск выполняется с помощью конструкции `MATCH (...) AGAINST (...)`, которая помещается в условие `WHERE` оператора `SELECT`. В круглых скобках после ключевого слова `MATCH` указываются имена столбцов, по которым производится поиск, а в скобках после `AGAINST` задается фраза, которую необходимо найти.

Замечание

Полнотекстовый поиск в СУБД MySQL не чувствителен к регистру. Кроме того, при поиске игнорируются так называемые "общеупотребительные" слова. К ним относятся слишком короткие слова (по умолчанию состоящие меньше чем из четырех символов), а также слова, встречающиеся, по крайней мере, в половине записей таблицы. Так, если в таблице имеются только две записи, то полнотекстовый поиск не даст результатов, т. к. каждое слово будет присутствовать, как минимум, в половине записей таблицы.

Для того чтобы осуществлять поиск по столбцу или по комбинации столбцов, столбец и каждая из комбинаций должны быть проиндексированы. То есть создание индекса `FULLTEXT` по двум столбцам `name` и `description` (см. листинг 7.18) позволит выполнять поиск одновременно по этим двум столбцам (листинг 7.22).

Листинг 7.22. Использование конструкции `MATCH (...) AGAINST (...)`

```
SELECT * FROM products
         WEHRE MATCH (name, description) AGAINST ('Celeron')
```

На запросы по отдельным столбцам будет возвращена ошибка 1191 — "Отсутствие проиндексированного столбца". Для осуществления таких запросов необходимо создать дополнительные индексы по столбцам `name` и `description`, помимо уже существующего индекса по обоим столбцам.

В листинге 7.23 представлен дамп таблицы `catalogs`, содержащей два столбца: `name` и `description`. Дальнейшая демонстрация возможностей полнотекстового поиска будет проводиться на примере этой таблицы.

Листинг 7.23. Таблица `catalogs`

```
CREATE TABLE catalogs (
  id_catalog int(11) NOT NULL auto_increment,
  name tinytext NOT NULL,
  description text NOT NULL,
  PRIMARY KEY  (id_catalog),
  FULLTEXT KEY search (name,description)
);
```

```
INSERT INTO catalogs VALUES (1, 'Процессоры', 'Процессор — это центральное
устройство компьютера. Он  обрабатывает информацию и управляет работой других
устройств компьютера. В каталоге представлены два вида процессоров,
выпускаемых компаниями Intel и AMD.');
INSERT INTO catalogs VALUES (2, 'Материнские платы', 'Материнская плата — это
комплекс различных устройств, поддерживающий работу системы в целом.
Обязательными атрибутами материнской платы являются базовый процессор,
оперативная память, системный BIOS, контроллер клавиатуры, разъемы расширения.
В каталоге представлены материнские платы нескольких производителей: ASUS,
Gigabyte, ABIT и т. д.');
INSERT INTO catalogs VALUES (3, 'Видеоадаптеры', 'Видеоадаптер — это
устройство, через которое сигналы изображения выводятся на монитор (или другое
устройство). Как правило, имеют собственные видеопроцессоры, чипсет,
внутренние шины, специальные BIOS и буферные элементы памяти. При этом
видеопамять имеет значительный размер — 64, 128 и 256 Мбайт. Кроме того,
память, устанавливаемая на видеоадаптеры работает на больших частотах, нежели
ОЗУ компьютера. В каталоге представлены видеоадаптеры нескольких
производителей: ASUS, Gigabyte и т. д.');
INSERT INTO catalogs VALUES (4, 'Жесткие диски', 'Жесткий диск — это один или
несколько дисков из твердого материала с магнитным покрытием. Диски вращаются
на общей оси. Кроме собственно дисков, устройство включает в себя магнитные
головки для чтения и записи и необходимую электронику. Жесткий диск
используется для хранения данных. В каталоге представлены жесткие диски
нескольких производителей: Seagate, Maxtor, Western Digital и т. д.');
INSERT INTO catalogs VALUES (5, 'Оперативная память', 'Модуль оперативной
памяти представляет собой плату для реализации в компьютере памяти с
произвольным доступом. Оперативная память отличается более высокой
производительностью по сравнению с жестким диском.');
```

В листинге 7.24 демонстрируется поиск слова "Процессор" в таблице `catalogs`, причем из столбца `description` выводятся только первые 20 символов.

Листинг 7.24. Использование конструкции поиска

```
SELECT name, CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description) AGAINST ('Процессор Gigabyte');
+-------------------+------------------------+
| name              | description            |
+-------------------+------------------------+
| Процессоры        | Процессор — это цент... |
| Материнские платы | Материнская плата — ... |
+-------------------+------------------------+
```

Следует отметить, что ищется целое слово "Процессор" (без учета регистра), слово "видеопроцессоры" из записи с первичным ключом 3 уже не удовлетворяет поиску.

Для каждой строки конструкция `MATCH (...) AGAINST (...)` возвращает коэффициент релевантности, т. е. степень сходства между искомой строкой и текстом в таблице. Когда конструкция `MATCH (...) AGAINST (...)` используется в выражении `WHERE`, возвращенные строки столбцов автоматически сортируются с помещением в начало списка наиболее релевантных записей. Величина релевантности представляет собой неотрицательное число с плавающей точкой. Релевантность вычисляется на основе количества слов в данной строке столбца, количества уникальных слов в этой строке, общего количества слов в тексте и числа строк, содержащих отдельное слово.

Если хоть одно слово из списка `AGAINST()` присутствует в индексированных столбцах, запись появляется в результирующей таблице (листинг 7.25).

Листинг 7.25. Использование конструкции `MATCH (...) AGAINST (...)`

```
SELECT name, CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description) AGAINST ('Процессор Gigabyte');
+-------------------+------------------------+
| name              | description            |
+-------------------+------------------------+
| Материнские платы | Материнская плата — ... |
| Процессоры        | Процессор — это цент... |
| Видеоадаптеры     | Видеоадаптер — это у... |
+-------------------+------------------------+
```

Слова "Процессор" и "Gigabyte" встречаются вместе в записи, посвященной материнским платам, и по отдельности в записях, посвященных процессорам и видеоадаптерам. Поэтому запись "Материнские платы" получает больший коэффициент релевантности и располагается самой первой в результирующей таблице.

Если поместить конструкцию `MATCH()` в список столбцов, следующих за ключевым словом `SELECT`, можно получить численные значения коэффициента релевантности (листинг 7.26). Это не приводит к дополнительной нагрузке, поскольку оптимизатор MySQL определяет, что два вызова `MATCH()` идентичны, и выполняет только один поиск.

Листинг 7.26. Численные значения коэффициента релевантности

```
SELECT name,
       CONCAT(SUBSTRING(description,1,20),'...') AS description,
       MATCH(name, description)
       AGAINST ('Процессор Gigabyte') AS coefficient
```

```
FROM catalogs
WHERE MATCH (name, description) AGAINST ('Процессор Gigabyte');
+------------------+-----------------------+-----------------+
| name             | description           | coefficient     |
+------------------+-----------------------+-----------------+
| Материнские платы | Материнская плата — ... | 0.56135364417286 |
| Процессоры       | Процессор — это цент... | 0.32347170307520 |
| Видеоадаптеры    | Видеоадаптер — это у... | 0.26045211456341 |
+------------------+-----------------------+-----------------+
```

Как видно из приведенных выше примеров, в результат полнотекстового поиска входят записи, содержащие хотя бы одно слово из строки `AGAINST`, и коэффициент релевантности для которых отличен от нуля.

Для разбивки текста на слова СУБД MySQL использует очень простой синтаксический анализатор. Словом является любая последовательность символов, состоящая из букв, цифр, одинарных кавычек `'` и символов подчеркивания _. По умолчанию любое слово меньше 4-х символов игнорируется. Если требуется осуществлять поиск по словам, состоящим из трех символов, необходимо изменить значение системной переменной `ft_min_word_len` в конфигурационном файле my.ini или my.cnf (листинг 7.27) или передать параметр `--ft_min_word_len=3` при старте сервера `mysqld`.

Замечание

Если изменение системной переменной `ft_min_word_len` производится после того, как созданы таблицы с индексом `FULLTEXT`, необходимо перестроить индекс (удалить его из таблицы и создать вновь), т. к. структура MYI-файла индекса зависит от значения системной переменной `ft_min_word_len`.

Листинг 7.27. Изменение минимального числа символов в искомой строке

```
[mysqld]
ft_min_word_len=3
```

Каждое слово в искомой фразе оценивается в соответствии с его важностью в этом запросе. Таким образом, слово, присутствующее во многих документах, будет иметь меньший вес (и даже, возможно, нулевой, если слово присутствует в более чем половине записей таблицы), как имеющее более низкое смысловое значение в данном конкретном наборе текстов. С другой стороны, редко встречающееся слово получит более высокий вес. Затем полученные значения весов слов объединяются для вычисления релевантности данной строки столбца. Полнотекстовый поиск создавался и настраивался для поиска в большом объеме текста, поэтому он плохо работает с небольшими таблицами, где более эффективно работают регулярные выражения *(см. разд. 7.1)*.

Можно составлять достаточно сложные SQL-запросы, прибегая к комбинации нескольких конструкций `MATCH (...) AGAINST (...)` при помощи логических операторов `AND`, `OR` и `NOT`. В листинге 7.28 приводится пример запроса, в котором слова "Процессор" и "Gigabyte" ищутся при помощи двух отдельных конструкций `MATCH (...) AGAINST (...)`, объединенных логическим оператором `AND`.

Листинг 7.28. Использование логических операторов при полнотекстовом поиске

```
SELECT name,
       CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description) AGAINST ('Процессор') AND
      MATCH (name, description) AGAINST ('Gigabyte');
+-------------------+------------------------+
| name              | description            |
+-------------------+------------------------+
| Материнские платы | Материнская плата — ... |
+-------------------+------------------------+
```

Как видно из листинга 7.28, возвращается только единственная запись, посвященная материнским платам, которая содержит и слово "Процессор", и слово "Gigabyte".

При использовании комбинации нескольких конструкций `MATCH (...) AGAINST (...)`, объединенных логическими операторами, теряется преимущество сортировки результатов по коэффициенту релевантности, однако в большинстве случаев такая сортировка не нужна, и требуется просто отсортировать результат либо по алфавиту, либо по дате.

7.2.3. Модификаторы полнотекстового поиска

Конструкция `MATCH (...) AGAINST (...)` позволяет использовать поисковые модификаторы в конструкции `AGAINST`:

`MATCH (`*`col1`*`, `*`col2`*`, ...) AGAINST ('`*`search`*`' [`*`search_modifier`*`])`

В качестве необязательного модификатора *`search_modifier`* могут выступать следующие ключевые слова:

- `IN NATURAL LANGUAGE MODE` — естественный режим поиска, данное ключевое слово обозначает обычный режим, рассмотренный в текущем разделе. Ключевое слово `IN NATURAL LANGUAGE MODE` было введено для симметрии, начиная с версии 5.1.7;

- `IN BOOLEAN MODE` — логический режим поиска;
- `WITH QUERY EXPANSION` — поиск с расширением запроса, введенный, начиная с версии 4.1.1;
- `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION` — данное ключевое слово является синонимом для `WITH QUERY EXPANSION` и введено, начиная с версии 5.1.7.

7.2.4. Логический режим

Логический режим позволяет более гибко управлять системой поиска, для его активизации в конструкции `AGAINST` после строки с ключевыми словами следует поместить конструкцию `IN BOOLEAN MODE`, как это продемонстрировано в листинге 7.29.

Листинг 7.29. Поиск в логическом режиме

```
SELECT name, CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description)
      AGAINST ('+Процессор Gigabyte' IN BOOLEAN MODE);
+-------------------+------------------------+
| name              | description            |
+-------------------+------------------------+
| Процессоры        | Процессор — это цент... |
| Материнские платы | Материнская плата — ... |
+-------------------+------------------------+
```

По сравнению с листингом 7.25, запрос в листинге 7.29 выдал только две записи, это связано с тем, что ключевому слову "Процессор" предшествует знак `+`, который требует, чтобы слово входило в каждую запись. Поэтому записи, содержащие в поле `description` только слово "Gigabyte", но не содержащие в индексированных столбцах слово "Процессор", отбрасываются.

Замечание

В отличие от естественного режима поиска, в логическом режиме отсутствует сортировка по релевантности запросов и ограничение на вхождение слова в более чем 50% записей таблицы.

Если слову предшествует знак "минус" `-`, то СУБД MySQL вернет только те записи, которые не содержат данного слова. В листинге 7.30 представлен запрос, который возвращает все записи, в состав которых входит слово "Gigabyte", но из выборки исключаются записи, содержащие слово "Процессор".

Листинг 7.30. Исключение слов из выборки

```
SELECT name, CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description)
      AGAINST ('-Процессор Gigabyte' IN BOOLEAN MODE);
+---------------+------------------------+
| name          | description            |
+---------------+------------------------+
| Видеоадаптеры | Видеоадаптер — это у... |
+---------------+------------------------+
```

Важно отметить, что запрос, состоящий из одних слов, которые предваряет минус, вернет пустую выборку (листинг 7.31).

Листинг 7.31. Использование символа "минус" –

```
SELECT name, CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description)
      AGAINST ('-Процессор -Gigabyte' IN BOOLEAN MODE);
Empty set (0.00 sec)
```

По умолчанию (если ни плюс, ни минус не указаны) данное слово является не обязательным, но содержащие его строки будут оцениваться более высоко. Это имитирует поведение команды `MATCH (...) AGAINST(...)` без модификатора `IN BOOLEAN MODE`. Так, в листинге 7.32 использовать ключевое слово `IN BOOLEAN MODE` не обязательно.

Листинг 7.32. Естественный режим поиска

```
SELECT name, CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description)
      AGAINST ('Процессор Gigabyte' IN BOOLEAN MODE);
+------------------+------------------------+
| name             | description            |
+------------------+------------------------+
| Процессоры       | Процессор — это цент... |
| Материнские платы | Материнская плата — ... |
| Видеоадаптеры    | Видеоадаптер — это у... |
+------------------+------------------------+
```

В листинге 7.32 выборка содержит записи, в которых встречается как слово "Процессор", так и слово "Gigabyte".

Помимо + и –, в логическом режиме предусмотрено еще несколько специальных символов, описание которых приведено в табл. 7.2.

Таблица 7.2. *Специальные символы логического режима*

Символ	Описание
+	Предшествующий слову знак + показывает, что это слово должно присутствовать в каждой возвращенной строке
–	Предшествующий слову знак – показывает, что это слово не должно присутствовать в какой-либо возвращенной строке
<	Этот оператор применяется для того, чтобы уменьшить вклад слова в величину релевантности, которая приписывается строке
>	Этот оператор используются для того, чтобы увеличить вклад слова в величину релевантности, которая приписывается строке
( )	Круглые скобки группируют слова в подвыражения
~	Предшествующий слову знак ~ воздействует как оператор отрицания, обуславливая негативный вклад данного слова в релевантность строки. Им отмечают нежелательные слова. Строка, содержащая такое слово, будет оценена ниже других, но не будет исключена совершенно, как в случае символа –
*	Звездочка является оператором усечения. В отличие от остальных символов, она должна добавляться в конце слова, а не в начале
"	Фраза, заключенная в двойные кавычки, соответствует только строкам, содержащим эту фразу, написанную буквально

7.2.5. Режим расширения запроса

Режим расширения запроса применяется, как правило, когда поисковая фраза подразумевает более широкую трактовку запроса. Например, пользователь, выполняя поиск по слову "database" может иметь в виду, что записи, в состав которых входят фразы "MySQL", "Oracle", "DB2" и "MSSQL" тоже должны быть включены в результирующую таблицу. Для включения режима расширения запроса в конструкции `AGAINST` после строки с ключевыми словами следует поместить конструкцию `WITH QUERY EXPANSION` или `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION`, как это продемонстрировано в листинге 7.33.

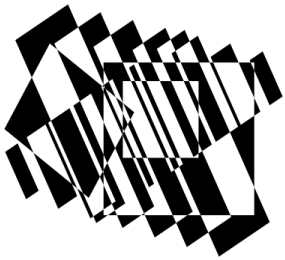
Листинг 7.33. Поиск в режиме расширения запроса

```
SELECT name, CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description)
      AGAINST ('Процессор');
```

```
+-------------------+------------------------+
| name              | description            |
+-------------------+------------------------+
| Процессоры        | Процессор — это цент... |
| Материнские платы | Материнская плата — ... |
+-------------------+------------------------+
SELECT name, CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description)
      AGAINST ('Процессор' WITH QUERY EXPANSION);
+--------------------+------------------------+
| name               | description            |
+--------------------+------------------------+
| Материнские платы  | Материнская плата — ... |
| Процессоры         | Процессор — это цент... |
| Видеоадаптеры      | Видеоадаптер — это у... |
| Оперативная память | Модуль оперативной п... |
+--------------------+------------------------+
```

В режиме расширения запроса происходит двойной поиск: сначала ищется искомое слово, а затем слова из результатов поиска. Поэтому при поиске в режиме `WITH QUERY EXPANSION` в листинге 7.33, помимо записей, содержащих слово "Процессор", найдены также дополнительные записи, в которых это слово не встречается. Следует выбирать как можно более короткие начальные поисковые фразы, т. к. при повторном поиске используются все слова из найденных записей, больше трех символов и встречающиеся в менее чем 50% записей.

ГЛАВА 8



Функции, применяемые вместе с конструкцией *GROUP BY*

Функции, применяемые совместно с конструкцией `GROUP BY`, часто называют *агрегатными* или *суммирующими функциями*. Они вычисляют одно значение для каждой группы, которые создает конструкция `GROUP BY`. Функции данной группы позволяют узнать число строк, входящих в группу, подсчитать среднее значение или получить сумму значений столбцов. Результирующее значение рассчитывается только для значений, не равных `NULL` (исключение составляет лишь функция `COUNT(*)`, которая подсчитывает число всех строк). Несмотря на то, что функции позиционируются как функции, применяемые совместно с конструкцией `GROUP BY`, вполне допустимо их использование в запросах без группировки. В этом случае вся выборка выступает как одна большая группа. В табл. 8.1 представлен список агрегатных функций, доступных в MySQL.

Таблица 8.1. *Агрегатные функции MySQL*

Функция	**Описание**
`AVG(`*expr*`)`	Возвращает среднее значение аргумента *expr*
`BIT_AND(`*expr*`)`	Возвращает побитовое И для всех битов в *expr*
`BIT_OR(`*expr*`)`	Возвращает побитовое ИЛИ для всех битов в *expr*
`BIT_XOR(`*expr*`)`	Возвращает исключающее побитовое ИЛИ для всех битов в *expr*
`COUNT(`*expr*`)`, `COUNT(*)`, `COUNT(DISTINCT` *expr1*, *expr2*, ...`)`	Подсчитывает число записей в *expr*
`GROUP_CONCAT(`*expr*`)`	Объединяет значения отдельных групп, полученных в результате применения конструкции `GROUP BY`, в одну строку

Таблица 8.1 *(окончание)*

Функция	Описание
MIN([DISTINCT] *expr*)	Возвращает минимальное значение среди всех непустых значений выбранных строк в столбце *expr*
MAX([DISTINCT] *expr*)	Возвращает максимальное значение среди всех непустых значений выбранных строк в столбце *expr*
STD(*expr*), STDDEV(*expr*), STDDEV_POP(*expr*)	Возвращает стандартное среднеквадратичное отклонение значения в аргументе *expr*
STDDEV_SAMP()	Возвращает выборочное среднеквадратическое отклонение *expr*
SUM([DISTINCT] *expr*)	Возвращает сумму величин в столбце *expr*
VAR_POP(*expr*), VARIANCE(*expr*)	Возвращает стандартное отклонение значения в столбце *expr*
VAR_SAMP(*expr*)	Возвращает выборочное отклонение значения в аргументе *expr*

8.1. Среднее значение

Функция AVG() возвращает среднее значение аргумента *expr* и имеет следующий синтаксис

AVG([DISTINCT] *expr*)

В качестве аргумента *expr* обычно выступает имя столбца. Необязательное ключевое слово DISTINCT позволяет дать указание СУБД MySQL обрабатывать только уникальные значения столбца *expr*.

Замечание

Ключевое слово DISTINCT в функции AVG() добавлено, начиная с версии 5.0.3.

В листинге 8.1 представлена таблица catalogs, состоящая из трех полей: id_catalog — первичный ключ, name — название раздела и total — количество товарных позиций на складе.

Листинг 8.1. Таблица catalogs

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL auto_increment,
  name TINYTEXT NOT NULL,
```

```
  total INT(11) NOT NULL,
  PRIMARY KEY (id_catalog)
);
INSERT INTO catalogs VALUES (1, 'Процессоры', 15);
INSERT INTO catalogs VALUES (2, 'Материнские платы', 4);
INSERT INTO catalogs VALUES (3, 'Видеоадаптеры', 7);
INSERT INTO catalogs VALUES (4, 'Жесткие диски', 18);
INSERT INTO catalogs VALUES (5, 'Оперативная память', 9);
```

В листинге 8.2 приводится запрос к таблице `catalogs`, который возвращает среднее количество товаров на складе.

Листинг 8.2. Среднее количество товарных позиций на складе

```
SELECT AVG(total) FROM catalogs;
+------------+
| AVG(total) |
+------------+
|    10.6000 |
+------------+
```

При использовании конструкции `GROUP BY` функция `AVG()` вычисляет среднее значение для каждой из группы. Пусть имеется таблица `products` (листинг 8.3), в которой хранится информация о товарных позициях. Таблица имеет следующие поля:

- `id_product` — первичный ключ;
- `name` — название товарной позиции;
- `price` — цена товарной позиции;
- `id_catalog` — ссылка на элемент каталога, которому принадлежит товарная позиция.

Листинг 8.3. Таблица `products`

```
CREATE TABLE products (
  id_product INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  price decimal(7,2) NOT NULL,
  id_catalog int(11) NOT NULL,
  PRIMARY KEY (id_product),
  KEY id_catalog (id_catalog)
);
INSERT INTO products VALUES (1, 'Материнская плата N 1', '2050.00', 2);
INSERT INTO products VALUES (2, 'Оперативная память N 1', '1500.00', 5);
```

```
INSERT INTO products VALUES (3, 'Процессор N 1', '6000.00', 1);
INSERT INTO products VALUES (4, 'Процессор N 2', '3523.00', 1);
INSERT INTO products VALUES (5, 'Процессор N 3', '4856.00', 1);
INSERT INTO products VALUES (6, 'Оперативная память N 2', '1028.00', 5);
INSERT INTO products VALUES (7, 'Материнская плата N 2', '1845.00', 2);
INSERT INTO products VALUES (8, 'Жесткий диск N 1', '4210.00', 4);
INSERT INTO products VALUES (9, 'Видеоадаптер N 1', '3670.00', 3);
```

Если сгруппировать записи таблицы `products` по полю `id_catalog`, при помощи встроенной функции `AVG()`, можно узнать среднюю цену по каждому из пяти элементов каталога. В листинге 8.4 выводится результирующая таблица с уникальными номерами `id_catalog` и соответствующие каждому элементу каталога средние цены.

Листинг 8.4. Использование функции `AVG()` совместно с `GROUP BY`

```
SELECT id_catalog, AVG(price) FROM products GROUP BY id_catalog;
+------------+-------------+
| id_catalog | AVG(price)  |
+------------+-------------+
|          1 | 4793.000000 |
|          2 | 1947.500000 |
|          3 | 3670.000000 |
|          4 | 4210.000000 |
|          5 | 1264.000000 |
+------------+-------------+
```

Средние значения, полученные при помощи функции `AVG()`, могут использоваться в вычисляемых столбцах. Например, для того чтобы увеличить средние значения для каждого элемента каталога на 20%, достаточно умножить либо столбец `price`, либо функцию `AVG()` на `1.2` (листинг 8.5).

Листинг 8.5. Использование функции `AVG()` в выражениях

```
SELECT id_catalog, AVG(price*1.2) FROM products
GROUP BY id_catalog;
+------------+----------------+
| id_catalog | AVG(price*1.2) |
+------------+----------------+
|          1 |   5751.6000000 |
|          2 |   2337.0000000 |
|          3 |   4404.0000000 |
|          4 |   5052.0000000 |
|          5 |   1516.8000000 |
+------------+----------------+
```

```
5 rows in set (0.02 sec)
SELECT id_catalog, AVG(price)*1.2 FROM products
GROUP BY id_catalog;
+------------+----------------+
| id_catalog | AVG(price*1.2) |
+------------+----------------+
|          1 |   5751.6000000 |
|          2 |   2337.0000000 |
|          3 |   4404.0000000 |
|          4 |   5052.0000000 |
|          5 |   1516.8000000 |
+------------+----------------+
5 rows in set (0.00 sec)
```

В первом случае сначала каждое из значений столбца `price` умножается на `1.2`, а затем производится группировка и вычисление результата при помощи функции `AVG()`. Во втором случае сначала осуществляется группировка значений и вычисление средних значений, и лишь после этого результат умножается на `1.2`. Как следствие, первый запрос выполняется 0,02 секунды, а второй 0,00 секунды.

8.2. Сортировка агрегатных значений

По полученным в результате выполнения агрегатных функций значениям может выполняться сортировка результирующей таблицы. Для этого столбцу назначается псевдоним при помощи оператора `AS`, который передается конструкции `ORDER BY` (листинг 8.6).

Листинг 8.6. Сортировка результирующей таблицы

```
SELECT id_catalog, AVG(price) AS price FROM products
GROUP BY id_catalog ORDER BY price DESC;
+------------+-------------+
| id_catalog | price       |
+------------+-------------+
|          1 | 4793.000000 |
|          4 | 4210.000000 |
|          3 | 3670.000000 |
|          2 | 1947.500000 |
|          5 | 1264.000000 |
+------------+-------------+
```

Если необходимо использовать в условии вычисляемые столбцы, следует указывать конструкцию `HAVING`, а не `WHERE` (листинг 8.7).

Листинг 8.7. Использование условия HAVING

```
SELECT id_catalog, AVG(price) AS price FROM products
GROUP BY id_catalog HAVING price > 3000 ORDER BY price DESC;
+------------+-------------+
| id_catalog | price       |
+------------+-------------+
|          1 | 4793.000000 |
|          4 | 4210.000000 |
|          3 | 3670.000000 |
+------------+-------------+
```

8.3. Подсчет количества записей в таблице

Подсчет количества записей в таблице осуществляется при помощи функции COUNT(), имеющей несколько форм, синтаксис которых представлен ниже:

```
COUNT(expr)
COUNT(*)
COUNT(DISTINCT expr1, expr2, ...)
```

Первая форма возвращает число записей в таблице, поле *expr* для которых не равно NULL. В листинге 8.8 представлен дамп таблицы tbl.

Листинг 8.8. Дамп таблицы tbl

```
CREATE TABLE tbl (
  id int(11) NOT NULL,
  value int(11) default NULL
);
INSERT INTO tbl VALUES (1, 230);
INSERT INTO tbl VALUES (2, NULL);
INSERT INTO tbl VALUES (3, 405);
INSERT INTO tbl VALUES (4, NULL);
```

В листинге 8.8 представлен запрос, возвращающий число записей в таблице tbl.

Листинг 8.9. Использование функции COUNT()

```
SELECT * FROM tbl;
+----+-------+
| id | value |
+----+-------+
|  1 |   230 |
|  2 |  NULL |
```

```
|  3 |   405 |
|  4 |  NULL |
+----+-------+
SELECT COUNT(id), COUNT(value) FROM tbl;
+-----------+--------------+
| COUNT(id) | COUNT(value) |
+-----------+--------------+
|         4 |            2 |
+-----------+--------------+
```

Как видно из листинга 8.9, для полей `id` и `value` возвращаются различные значения. Это связано с тем, что количество `NULL`-полей в столбцах различается.

Форма функции `COUNT(*)` возвращает общее число строк в таблице, независимо от того, принимает какое-либо поле значение `NULL` или нет (листинг 8.10). Запись таблицы учитывается в результате, даже если все поля записи равны `NULL`.

Листинг 8.10. Использование функции `COUNT(*)`

```
SELECT COUNT(*) FROM tbl;
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
```

Функция `COUNT(*)` оптимизирована для очень быстрого возврата результата при условии, что команда `SELECT` извлекает данные из одной таблицы, никакие другие столбцы не обрабатываются и запрос не содержит условия `WHERE`.

Функция `COUNT()` может быть использована не только для подсчета общего числа записей в таблице, но и для подсчета числа строк в выборке с условием `WHERE` (листинг 8.11).

Листинг 8.11. Использование функции `COUNT()` совместно с `WHERE`

```
SELECT COUNT(*) FROM tbl WHERE value < 300;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
```

В листинге 8.11 представлен запрос, извлекающий из таблицы `tbl` записи, чье поле `value` меньше 300. Следует обратить внимание, что поля, содержащие `NULL`, не удовлетворяют этому условию, т. к. `NULL` обозначается отсутствие данных (поле может быть как больше, так и меньше 300 — информация об этом не известна).

Разумеется, как агрегатная функция, `COUNT()` может быть использована для вычисления числа записей в каждой из групп, полученных в результате применения группировки результата с помощью конструкции `GROUP BY`.

В листинге 8.12 представлен запрос к таблице `products` (листинг 8.3), сообщающий о количестве товарных позиций по каждому элементу каталога.

Листинг 8.12. Подсчет количества товарных позиций для каждого элемента каталога

```
SELECT id_catalog, COUNT(*) AS total FROM products
GROUP BY id_catalog ORDER BY total DESC;
+------------+-------+
| id_catalog | total |
+------------+-------+
|          1 |     3 |
|          5 |     2 |
|          2 |     2 |
|          4 |     1 |
|          3 |     1 |
+------------+-------+
```

Сортировка строк в результирующей таблице производится по количеству товарных позиций в каталоге. Для этого столбцу `COUNT(*)` при помощи ключевого слова `AS` назначается псевдоним `total`, который затем используется в выражении `ORDER BY`.

Третий вариант функции `COUNT()` позволяет использовать ключевое слово `DISTINCT`, которое обеспечивает подсчет только уникальных значений столбца (листинг 8.13).

Листинг 8.13. Использование ключевого слова `DISTINCT` совместно с `COUNT()`

```
SELECT COUNT(id_catalog), COUNT(DISTINCT id_catalog) FROM products;
+-------------------+----------------------------+
| COUNT(id_catalog) | COUNT(DISTINCT id_catalog) |
+-------------------+----------------------------+
|                 9 |                          5 |
+-------------------+----------------------------+
```

8.4. Объединение значений группы

Для объединения значения группы предназначена функция GROUP_CONCAT(), которая имеет следующий синтаксис:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
             [ASC | DESC] [,col_name ...]]
             [SEPARATOR str_val])
```

В простейшем случае функция принимает имя столбца *expr* и возвращает строку со значениями столбца, разделенными запятыми. В листинге 8.14 приводится пример использования функции GROUP_CONCAT() для поля id_catalog таблицы products (листинг 8.3).

Замечание

Функция появилась в MySQL, начиная с версии 4.1.0.

Листинг 8.14. Использование функции GROUP_CONCAT()

```
SELECT GROUP_CONCAT(id_catalog) FROM products;
+--------------------------+
| GROUP_CONCAT(id_catalog) |
+--------------------------+
| 1,1,1,2,2,3,4,5,5        |
+--------------------------+
```

При передаче в качестве аргумента имени числового столбца значения автоматически преобразуются к текстовому типу.

Ключевое слово DISTINCT требует вернуть только уникальные значения столбца, а ключевое слово SEPARATOR позволяет задать в качестве разделителя значений произвольный символ. В листинге 8.15 представлен запрос, извлекающий уникальные значения столбца id_catalog таблицы products с использованием в качестве разделителя символа -.

Листинг 8.15. Использование ключевых слов DISTINCT и SEPARATOR

```
SELECT GROUP_CONCAT(DISTINCT id_catalog SEPARATOR '-')
FROM catalogs;
+-------------------------------------------------+
| GROUP_CONCAT(DISTINCT id_catalog SEPARATOR '-') |
+-------------------------------------------------+
| 1-2-3-4-5                                       |
+-------------------------------------------------+
```

Ключевое слово ORDER BY позволяет отсортировать значения в рамках возвращаемой строки. Отсортируем значения, представленные в листинге 8.15, в обратном порядке (листинг 8.16).

Листинг 8.16. Использование ключевого слова ORDER BY

```
SELECT GROUP_CONCAT(DISTINCT id_catalog
                    ORDER BY id_catalog
                    DESC SEPARATOR '-') AS str
FROM catalogs;
+-----------+
| str       |
+-----------+
| 5-4-3-2-1 |
+-----------+
```

В листинге 8.17 возвращается список цен (в порядке убывания) на товарные позиции каждого из каталогов.

Листинг 8.17. Список цен товарных позиций в каталогах

```
SELECT GROUP_CONCAT(price ORDER BY price DESC)
FROM products GROUP BY id_catalog;
+----------------------------------------+
| GROUP_CONCAT(price ORDER BY price DESC) |
+----------------------------------------+
| 6000.00,4856.00,3523.00                |
| 2050.00,1845.00                        |
| 3670.00                                |
| 4210.00                                |
| 1500.00,1028.00                        |
+----------------------------------------+
```

8.5. Поиск минимального и максимального значений

Для поиска минимального значения в столбце *expr* предназначена функция MIN(), которая имеет следующий синтаксис:

```
MIN([DISTINCT] expr)
```

В качестве аргумента *expr* обычно выступает имя столбца. Необязательное ключевое слово DISTINCT позволяет дать указание СУБД MySQL обрабатывать только уникальные значения столбца *expr*.

Замечание

Ключевое слово `DISTINCT` в функции `MIN()` добавлено, начиная с версии 5.0.0.

В листинге 8.18 приводится пример запроса, который ищет в таблице `products` (листинг 8.3) минимальную цену.

Листинг 8.18. Использование функции `MIN()`

```
SELECT MIN(price) FROM products;
+------------+
| MIN(price) |
+------------+
|    1028.00 |
+------------+
```

Использование конструкции `GROUP BY id_catalog` позволяет найти минимальную цену для каждого из элементов каталога (листинг 8.19).

Листинг 8.19. Использование функции `MIN()` совместно с конструкцией `GROUP BY`

```
SELECT id_catalog, MIN(price) FROM products GROUP BY id_catalog;
+------------+------------+
| id_catalog | MIN(price) |
+------------+------------+
|          1 |    3523.00 |
|          2 |    1845.00 |
|          3 |    3670.00 |
|          4 |    4210.00 |
|          5 |    1028.00 |
+------------+------------+
```

Как видно из результатов запроса, товарная позиция с минимальной для всей таблицы ценой находится в пятом элементе каталога.

Замечание

Функцию `MIN()` можно также использовать со строковыми столбцами. В этом случае возвращается минимальное лексикографическое значение.

Для поиска максимального значения в столбце *expr* предназначена функция `MAX()`, которая имеет следующий синтаксис:

```
MAX([DISTINCT] expr)
```

В качестве аргумента *expr* обычно выступает имя столбца. Необязательное ключевое слово DISTINCT позволяет дать указание СУБД MySQL обрабатывать только уникальные значения столбца *expr*.

Замечание

Ключевое слово DISTINCT в функции MAX() добавлено, начиная с версии 5.0.0.

В листинге 8.20 демонстрируется использование функции MAX().

Листинг 8.20. Использование функции MAX()

```
SELECT MAX(price) FROM products;
+------------+
| MAX(price) |
+------------+
|    6000.00 |
+------------+
SELECT id_catalog, MAX(price) FROM products GROUP BY id_catalog;
+------------+------------+
| id_catalog | MAX(price) |
+------------+------------+
|          1 |    6000.00 |
|          2 |    2050.00 |
|          3 |    3670.00 |
|          4 |    4210.00 |
|          5 |    1500.00 |
+------------+------------+
```

8.6. Сумма столбца

Сумму столбца *expr* позволяет подсчитать функция SUM(), которая имеет следующий синтаксис:

```
SUM([DISTINCT] expr)
```

Если возвращаемый набор данных не содержит ни одной строки, то функция возвращает NULL. Необязательное ключевое слово DISTINCT позволяет указать СУБД MySQL обрабатывать только уникальные значения столбца *expr*. В листинге 8.21 приводится пример использования функции SUM().

Замечание

Ключевое слово DISTINCT в функции SUM() добавлено, начиная с версии 5.0.0.

Листинг 8.21. Использование функции SUM()

```
SELECT SUM(price) FROM products;
+------------+
| SUM(price) |
+------------+
|   28682.00 |
+------------+
SELECT id_catalog, SUM(price) FROM products GROUP BY id_catalog;
+------------+------------+
| id_catalog | SUM(price) |
+------------+------------+
|          1 |   14379.00 |
|          2 |    3895.00 |
|          3 |    3670.00 |
|          4 |    4210.00 |
|          5 |    2528.00 |
+------------+------------+
```

8.7. Конструкция *WITH ROLLUP*

Конструкция WITH ROLLUP позволяет добавить еще одну строку с суммой значений всех предыдущих строк (листинг 8.22).

Листинг 8.22. Использование конструкции WITH ROLLUP

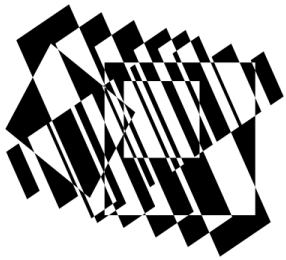
```
SELECT id_catalog, SUM(price) FROM products
GROUP BY id_catalog WITH ROLLUP;
+------------+------------+
| id_catalog | SUM(price) |
+------------+------------+
|          1 |   14379.00 |
|          2 |    3895.00 |
|          3 |    3670.00 |
|          4 |    4210.00 |
|          5 |    2528.00 |
|       NULL |   28682.00 |
+------------+------------+
```

Как видно из листинга 8.22, в результирующую таблицу добавлена дополнительная строка с суммой значений столбца SUM(price) и значением NULL для столбца id_catalog. Значение NULL присваивается всем столбцам, кроме сгенерированного агрегатной функцией (в данном случае функцией SUM()).

Замечание

При использовании конструкции WITH ROLLUP нельзя применять конструкцию ORDER BY.

ГЛАВА 9



Переменные

Часто результаты запроса необходимо использовать в последующих запросах. Для этого полученные данные следует сохранить во временных структурах. Эту задачу решают переменные SQL и динамические запросы.

9.1. Переменные SQL

Пусть имеется таблица `products` (листинг 9.1), в которой хранится информация о товарных позициях. Таблица имеет следующие поля:

- `id_product` — первичный ключ;
- `name` — название товарной позиции;
- `price` — цена товарной позиции;
- `id_catalog` — ссылка на элемент каталога, которому принадлежит товарная позиция.

Листинг 9.1. Таблица `products`

```
CREATE TABLE products (
  id_product INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  price decimal(7,2) NOT NULL,
  id_catalog int(11) NOT NULL,
  PRIMARY KEY  (id_product),
  KEY id_catalog (id_catalog)
);
INSERT INTO products VALUES (1, 'Материнская плата N 1', '2050.00', 2);
INSERT INTO products VALUES (2, 'Оперативная память N 1', '1500.00', 5);
INSERT INTO products VALUES (3, 'Процессор N 1', '6000.00', 1);
INSERT INTO products VALUES (4, 'Процессор N 2', '3523.00', 1);
```

```
INSERT INTO products VALUES (5, 'Процессор N 3', '4856.00', 1);
INSERT INTO products VALUES (6, 'Оперативная память N 2', '1028.00', 5);
INSERT INTO products VALUES (7, 'Материнская плата N 2', '1845.00', 2);
INSERT INTO products VALUES (8, 'Жесткий диск N 1', '4210.00', 4);
INSERT INTO products VALUES (9, 'Видеоадаптер N 1', '3670.00', 3);
```

СУБД MySQL предоставляет возможность сохранения результатов текущего запроса для использования в следующих запросах в переменных SQL. Объявление переменной начинается с символа `@`, за которым следует имя переменной. Значения переменным присваиваются посредством оператора `SELECT` с использованием оператора присваивания `:=` (листинг 9.2).

Листинг 9.2. Объявление и использование переменных SQL

```
SELECT @total := COUNT(*) FROM products;
+--------------------+
| @total := COUNT(*) |
+--------------------+
|                  9 |
+--------------------+
SELECT @total;
+--------+
| @total |
+--------+
|      9 |
+--------+
```

В листинге 9.2 объявляется переменная `@total`, которой присваивается число записей в таблице `products` учебной базы данных `shop`. Затем в рамках текущего сеанса в последующих запросах появляется возможность использования данной переменной.

Замечание

Переменная действует только в рамках одного сеанса соединения с сервером MySQL и прекращает свое существование после разрыва соединения.

В листинге 9.3 из таблицы `products` извлекается информация о товарной позиции, обладающей самой высокой ценой.

Листинг 9.3. Извлечение товарной позиции с самой высокой ценой

```
SELECT @price := MAX(price) FROM products;
+----------------------+
| @price := MAX(price) |
+----------------------+
|              6000.00 |
+----------------------+
```

```
SELECT * FROM products WHERE price = @price;
+------------+---------------+---------+------------+
| id_product | name          | price   | id_catalog |
+------------+---------------+---------+------------+
|          3 | Процессор N 1 | 6000.00 |          1 |
+------------+---------------+---------+------------+
```

Если в качестве значения переменной передается имя столбца, содержащего множество значений, то переменная получит последнее значение (листинг 9.4).

Листинг 9.4. Передача в качестве значения переменной имени столбца

```
SELECT @id := id_product FROM products;
+-------------------+
| @id := id_product |
+-------------------+
|                 1 |
|                 2 |
|                 3 |
|                 4 |
|                 5 |
|                 6 |
|                 7 |
|                 8 |
|                 9 |
+-------------------+
SELECT @id;
+------+
| @id  |
+------+
| 9    |
+------+
SELECT * FROM products WHERE id_product = @id;
+------------+------------------+---------+------------+
| id_product | name             | price   | id_catalog |
+------------+------------------+---------+------------+
|          9 | Видеоадаптер N 1 | 3670.00 |          3 |
+------------+------------------+---------+------------+
```

В качестве значения переменной необязательно должны выступать результаты выполнения функций и значения столбцов таблицы, можно использовать произвольные числовые и строковые значения (листинг 9.5).

Листинг 9.5. Использование для инициализации переменной числового значения

```
SELECT * FROM products WHERE id_product = @id;
+------------+---------------+---------+------------+
| id_product | name          | price   | id_catalog |
+------------+---------------+---------+------------+
|          3 | Процессор N 1 | 6000.00 |          1 |
+------------+---------------+---------+------------+
```

Следует помнить, что имена переменных чувствительны к регистру (листинг 9.6).

Листинг 9.6. Имена переменных чувствительны к регистру

```
SELECT @id := 5, @ID := 3;
+----------+----------+
| @id := 5 | @ID := 3 |
+----------+----------+
|        5 |        3 |
+----------+----------+
SELECT @id, @ID;
+------+------+
| @id  | @ID  |
+------+------+
| 3    | 3    |
+------+------+
```

Переменные также могут объявляться при помощи оператора `SET`, как это продемонстрировано в листинге 9.7.

Замечание

При использовании оператора `SET` в качестве оператора присваивания может выступать как `:=`, так и обычный знак равенства `=`.

Листинг 9.7. Использования оператора `SET` для объявления переменной

```
SET @last = CURDATE() - INTERVAL 7 DAY;
SELECT CURDATE(), @last;
+------------+------------+
| CURDATE()  | @last      |
+------------+------------+
| 2007-01-15 | 2007-01-08 |
+------------+------------+
```

Удобство применения оператора SET заключается в том, что в отличие от оператора SELECT он не возвращает результирующую таблицу.

9.2. Нумерация записей

Рассмотрим, как можно использовать переменные для нумерации записей в таблице. Пусть имеется таблица tbl, не снабженная первичным ключом (листинг 9.8).

Листинг 9.8. Таблица tbl

```
CREATE TABLE tbl (
  value tinytext
);
INSERT INTO tbl VALUES ('first');
INSERT INTO tbl VALUES ('second');
INSERT INTO tbl VALUES ('third');
INSERT INTO tbl VALUES ('fourth');
```

Пусть требуется при выводе содержимого таблицы tbl пронумеровать строки от 1 до *N*, где *N* — количество записей в таблице tbl. Решить данную задачу можно при помощи запроса, представленного в листинге 9.9.

Листинг 9.9. Нумерация записей при помощи пользовательских переменных

```
SET @start := 0;
SELECT @start := @start + 1 AS id, value FROM tbl;
+------+--------+
| id   | value  |
+------+--------+
|    1 | first  |
|    2 | second |
|    3 | third  |
|    4 | fourth |
+------+--------+
```

9.3. Системные переменные сервера

Помимо пользовательских переменных сервер MySQL поддерживает большое число системных переменных, с помощью которых можно осуществлять его тонкую настройку и масштабирование. Каждая системная переменная имеет значение по умолчанию. Получить их полный список можно при по-

мощи оператора SHOW VARIABLES, частичный вывод которого представлен в листинге 9.10.

Замечание

Более подробно системные переменные рассматриваются в *приложении 1*.

Листинг 9.10. Использование оператора SHOW VARIABLES

```
SHOW VARIABLES;
+--------------------------------+---------------------------+
| Variable_name                  | Value                     |
+--------------------------------+---------------------------+
| auto_increment_increment       | 1                         |
| auto_increment_offset          | 1                         |
| automatic_sp_privileges        | ON                        |
...
| max_allowed_packet             | 1048576                   |
| max_binlog_cache_size          | 4294967295                |
| max_binlog_size                | 1073741824                |
...
| version_compile_machine        | ia32                      |
| version_compile_os             | Win32                     |
| wait_timeout                   | 28800                     |
+--------------------------------+---------------------------+
```

Значение некоторых системных переменных можно изменить. Для этого необходимо установить значение переменной в конфигурационном файле my.ini или my.cnf (листинг 9.11).

Листинг 9.11. Изменение значений системных переменных в конфигурационном файле

```
[mysqld]
sort_buffer_size = 16M
read_buffer_size = 1M
join_buffer_size = 1M
query_cache_size = 64M
query_cache_limit = 2M
```

Помимо этого, можно переопределить значение системной переменной динамически во время работы сервера. Для этого необходимо воспользоваться оператором SET. Однако изменить можно не все системные переменные, а лишь некоторые.

Сервер MySQL поддерживает два типа переменных: *глобальные*, которые влияют на сервер, и *сеансовые*, которые влияют на текущее соединение клиента с сервером.

При старте сервера происходит инициализация глобальных переменных значениями по умолчанию. Можно влиять на значения глобальных переменных либо параметрами командной строки, либо установкой соответствующих директив в конфигурационных файлах my.ini и my.cnf. Оператор `SET` позволяет изменять значения глобальных переменных уже после старта, для этого используется синтаксис `SET GLOBAL` *`name=var`*, где *`name`* — имя переменной, *`var`* — ее значение (листинг 9.12).

Листинг 9.12. Установка глобальной переменной

```
SET GLOBAL read_buffer_size = 2097152;
```

В листинге 9.12 значение глобальной переменной `read_buffer_size`, несущей ответственность за размер буфера чтения, устанавливается равным двум мегабайтам. Для установки глобальных переменных имеется альтернативный синтаксис, основанный на прибавлении к имени переменной префикса `@@global`, представленный в листинге 9.13.

Замечание

При установке системных переменных, обозначающих объем памяти, можно использовать суффиксы `K`, `M` и `G` для обозначения килобайт, мегабайт и гигибайт соответственно. Регистр букв при этом не имеет значения.

Листинг 9.13. Альтернативный способ установки глобальных переменных

```
SET @@global.read_buffer_size = 2097152;
```

Запросы в листингах 9.12 и 9.13 являются эквивалентными.

Помимо глобальных переменных, сервер MySQL поддерживает набор *сеансовых переменных* для каждого соединения клиента с сервером. При установке соединения с сервером сеансовые переменные получают значения, заданные для глобальных переменных. Однако для тех сеансовых переменных, которые являются динамическими, клиент при помощи оператора `SET` может выставлять новые значения, что осуществляется при помощи следующего синтаксиса: `SET SESSION` *`name=var`*, где *`name`* — имя переменной, *`var`* — ее значение (листинг 9.14).

Замечание

Установка сеансовых переменных не требует специальных привилегий.

Листинг 9.14. Установка сеансовой переменной

```
SET SESSION read_buffer_size = 2097152;
```

Сеансовые переменные действуют независимо от глобальных, т. е. если клиент устанавливает глобальную переменную, значение его текущей сеансовой переменной не меняется — изменения затронут только будущие соединения с сервером.

Замечание

MySQL генерирует ошибку, если выполняется оператор `SET GLOBAL` с переменной, которую можно использовать только с `SET SESSION`.

Помимо представленного в листинге 9.14 синтаксиса, оператор `SET` имеет несколько альтернативных форм для определения сеансовой переменной (листинг 9.15).

Замечание

Для того чтобы пользователи не могли установить очень большое значение системной переменной при помощи оператора `SET`, при запуске сервера можно ограничить максимальный размер, который способна принимать переменная при помощи параметра `--maximum-`*имя_переменной*.

Листинг 9.15. Установка сеансовой переменной

```
SET SESSION read_buffer_size = 2097152;
SET LOCAL read_buffer_size = 2097152;
SET read_buffer_size = 2097152;
SET @@session.read_buffer_size = 2097152;
```

Представленные в листинге 9.15 запросы являются эквивалентными. Ключевое слово `LOCAL` является синонимом для `SESSION`. Кроме того, так же как и в случае глобальных переменных, сеансовая переменная может быть снабжена префиксом `@@session` (или `@@local`) для того, чтобы явно указать область ее действия. Если ни одно из ключевых слов (`GLOBAL`, `LOCAL` или `SESSION`) не указывается, то по умолчанию переменная считается сессионной (сеансовой).

Чтобы установить локальной переменной значение глобальной, достаточно присвоить локальной переменной ключевое слово `DEFAULT` или осуществить присвоение явно (листинг 9.16).

Листинг 9.16. Использование ключевого слова `DEFAULT`

```
SET read_buffer_size = DEFAULT;
SET @@session.read_buffer_size = @@global.read_buffer_size;
```

Для того чтобы узнать текущее значение глобальной переменной, требуется либо обратиться к ней с добавлением суффикса `@@global`, либо использовать оператор `SHOW` (листинг 9.17).

Листинг 9.17. Извлечение значения глобальной переменной

```
SELECT @@global.read_buffer_size;
+---------------------------+
| @@global.read_buffer_size |
+---------------------------+
|                   2097152 |
+---------------------------+
SHOW GLOBAL VARIABLES LIKE 'read_buffer_size';
+------------------+---------+
| Variable_name    | Value   |
+------------------+---------+
| read_buffer_size | 2097152 |
+------------------+---------+
```

Получить значение сеансовой переменной можно одним из способов, представленных в листинге (листинг 9.18).

Листинг 9.18. Извлечение значения сеансовой переменной

```
SELECT @@read_buffer_size;
+--------------------+
| @@read_buffer_size |
+--------------------+
|            2097152 |
+--------------------+
SELECT @@session.read_buffer_size;
+----------------------------+
| @@session.read_buffer_size |
+----------------------------+
|                    2097152 |
+----------------------------+
SHOW SESSION VARIABLES LIKE 'read_buffer_size';
+------------------+---------+
| Variable_name    | Value   |
+------------------+---------+
| read_buffer_size | 2097152 |
+------------------+---------+
```

9.4. Динамические запросы

Начиная с версии 5.1, в MySQL доступны *динамические запросы*. Динамическими называют запросы, которым подобно пользовательским переменным могут быть сохранены под конкретным именем и вызваны позже в течение сессии. Для объявления динамического запроса используется оператор PREPARE, который имеет следующий синтаксис:

```
PREPARE stmt_name FROM preparable_stmt
```

Оператор объявляет динамический запрос, который формируется на основании строки *preparable_stmt*. В листинге 9.19 приводится пример использования динамического запроса.

Листинг 9.19. Создание динамического запроса ver

```
PREPARE ver FROM 'SELECT VERSION()';
EXECUTE ver;
+-------------------------+
| VERSION()               |
+-------------------------+
| 5.1.12-beta-community-nt |
+-------------------------+
```

В листинге 9.19 создается динамический запрос ver, в основе которого лежит SELECT-запрос 'SELECT VERSION()'. Выполнить динамический запрос можно, используя оператор EXECUTE.

Замечание

Динамические запросы имеют время жизни только в течение текущего сеанса. То есть после того, как соединение с сервером закрыто, динамический запрос перестает существовать. Если необходимо, чтобы сохраненный ранее запрос существовал более длительное время, нужно прибегнуть к представлениям, которые рассматриваются более подробно в *главе 14*.

Оператор EXECUTE предназначен для выполнения динамических запросов и имеет следующий синтаксис:

```
EXECUTE stmt_name [USING @var_name [, @var_name] ...]
```

Ключевое слово USING используется в том случае, если динамический запрос имеет параметры. Обратимся к таблице products, рассмотренной в листинге 9.1, и создадим динамический запрос, который выбирает товарные позиции, принадлежащие конкретному элементу каталога (т. е. записи, имеющие одинаковое значение id_catalog). При этом значение поля id_catalog оформим в качестве параметра (листинг 9.20).

Листинг 9.20. Использование параметра в динамическом запросе

```
PREPARE prd FROM 'SELECT * FROM products WHERE id_catalog = ?';
SET @id_catalog = 1;
EXECUTE prd USING @id_catalog;
+------------+---------------+---------+------------+
| id_product | name          | price   | id_catalog |
+------------+---------------+---------+------------+
|          3 | Процессор N 1 | 6000.00 |          1 |
|          4 | Процессор N 2 | 3523.00 |          1 |
|          5 | Процессор N 3 | 4856.00 |          1 |
+------------+---------------+---------+------------+
```

Как видно из листинга 9.20, в динамическом запросе `prd` в `WHERE`-условии использован параметр, который обозначается знаком вопроса. В дальнейшем, при вызове запроса `prd` при помощи оператора `EXECUTE` передать значение параметра можно посредством конструкции `USING`.

Динамический запрос может иметь более одного параметра, в этом случае они перечисляются после ключевого слова `USING` через запятую в том порядке, в котором встречаются в динамическом запросе (листинг 9.21).

Листинг 9.21. Использование нескольких параметров в динамическом запросе

```
PREPARE prd FROM 'SELECT * FROM products
WHERE id_catalog = ? AND price < ?';
SET @id_catalog = 1;
SET @price = 5000.0;
EXECUTE prd USING @id_catalog, @price;
+------------+---------------+---------+------------+
| id_product | name          | price   | id_catalog |
+------------+---------------+---------+------------+
|          4 | Процессор N 2 | 3523.00 |          1 |
|          5 | Процессор N 3 | 4856.00 |          1 |
+------------+---------------+---------+------------+
```

В листинге 9.21 представлен динамический запрос, в котором используются две переменные. В конечном итоге выполняется запрос, извлекающий все товарные позиции, соответствующие первому элементу каталога, цена которых не превышает 5000,0 рублей.

Динамические запросы имеют ряд ограничений: не допускается использование вложенных динамических запросов, а также нескольких запросов, т. е.

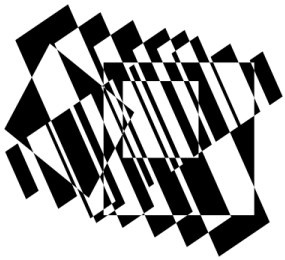
динамический запрос всегда представляет лишь один запрос. Параметр всегда передает строку, т. е. динамически задать имя таблицы или столбца не получится.

Удалить динамический запрос можно при помощи оператора `DROP PREPARE` или его синонима `DEALLOCATE PREPARE` (листинг 9.22).

Листинг 9.22. Удаление динамического запроса

```
DROP PREPARE prd;
```

ГЛАВА 10



Транзакции и блокировки

В промышленных базах данных одно событие может затрагивать большое число таблиц и требовать выполнения многочисленных запросов. Если на этапе выполнения одного из запросов происходит сбой, это может нарушить целостность базы данных, т. к. операция не будет доведена до конца. Чтобы сохранить целостность базы данных, все изменения должны выполняться как единое целое — либо все изменения успешно выполняются, либо в случае сбоя хотя бы одного из изменений база данных принимает состояние, которое было до начала изменений. Это обеспечивается средствами обработки транзакций, описываемых в данной главе.

10.1. Транзакции

Транзакция — это последовательность операторов SQL, выполняющихся как единая операция, которая не прерывается другими клиентами. То есть пока происходит работа с записями таблицы (их обновление или удаление), никто другой не может получить доступ к этим записям, т. к. СУБД MySQL автоматически блокирует доступ к ним.

Замечание

Таблицы типа MyISAM и HEAP не поддерживают транзакции. В настоящий момент их поддержка осуществляется только в таблицах типа BDB и InnoDB.

Транзакции позволяют объединять операторы в группу и гарантировать, что все операции внутри группы будут выполнены успешно. Если часть транзакции выполняется со сбоем, результаты выполнения всех операторов транзакции до места сбоя отменяются, приводя базу данных к виду, в котором она была до выполнения транзакции.

Для выяснения механизма транзакций рассмотрим ситуацию, которая возможна при работе с электронным магазином. Предположим, что два покупателя одновременно начинают покупку процессора Celeron D 320 2.4GHz, который остался на складе в одном экземпляре (листинг 10.1).

Листинг 10.1. Покупка Celeron D 320 2.4GHz

```
SELECT id_product, name, @total := count FROM products
WHERE name = 'Celeron D 320 2.4GHz';
+------------+----------------------+-------+
| id_product | name                 | count |
+------------+----------------------+-------+
|          4 | Celeron D 320 2.4GHz |     1 |
+------------+----------------------+-------+
```

Первый покупатель оформляет покупку, при этом происходят следующие события:

1. При помощи оператора `SELECT` запрашивается число товарных позиций на складе (листинг 10.1), которое помещается в переменную `@total` — дальнейшая регистрация покупки разрешена, если число процессоров больше нуля.
2. Регистрируется факт покупки в таблице `orders` — для этого в таблицу вставляется новая запись при помощи оператора `INSERT`.
3. Обновляется количество товарных позиций, доступных на складе, с первичным ключом `id_product` = 4 (таблица `products`). Для этого из переменной `@total` вычитается единица, и полученная цифра присваивается полю `count` товарной позиции (листинг 10.2).

Листинг 10.2. Обновление числа товарных позиций

```
UPDATE products SET count = @total - 1
WHERE name = 'Celeron D 320 2.4GHz';
```

Одновременно второй покупатель начинает оформление покупки. Запрос на выборку количества процессоров для него также возвращает значение 1, которое помещается в локальную переменную `@total`, видимую только в рамках текущего соединения с базой данных.

После регистрации покупки первым покупателем число товарных позиций, оставшихся на складе, обновляется с 1 на 0 (листинг 10.2).

После регистрации покупки вторым покупателем также происходит обновление таблицы `products` (листинг 10.2), но т. к. текущее соединение ничего не

знает о запросе, параллельно выполняющемся в другом соединении, то баланс по результату двух покупок будет равен 1-му процессору, в то время как покупатели заплатят за 2 и будут ожидать поставки двух процессоров.

Для предотвращения такой ситуации выполнение всех трех этапов покупки должно происходить как одна транзакция. В этом случае будет заключена лишь одна сделка — по второй сделке произойдет откат.

Другая проблема, связанная с выполнением группы операций, заключается в том, что при возникновении ошибки в середине группы операторов база данных останется в полумодифицированном состоянии.

Если при оплате покупки происходит перевод денежной суммы со счета клиента на счет электронного магазина, то счет клиента должен уменьшиться на сумму `sum`, а счет электронного магазина увеличится на ту же сумму (листинг 10.3).

Листинг 10.3. Обновление счетов

```
UPDATE account SET balance = balance – sum WHERE name = 'client';
UPDATE account SET balance = balance + sum WHERE name = 'bookshop';
```

Если в момент завершения первого запроса и перед началом выполнения второго произойдет сбой, транзакция окажется незавершенной.

Следует обратить внимание, что транзакции имеют смысл только в случае с типами таблиц, которые их поддерживают: InnoDB и BDB. Если существующие таблицы имеют другой тип, например MyISAM, для работы с транзакциями его следует изменить при помощи оператора `ALTER TABLE` (листинг 10.4).

Листинг 10.4. Изменение типа таблиц

```
ALTER TABLE catalogs ENGINE = InnoDB;
ALTER TABLE products ENGINE = InnoDB;
```

Следует особенно внимательно следить, чтобы при изменении типа MyISAM на любой другой в таблице отсутствовали `FULLTEXT`-индексы, т. к. никакой другой тип таблиц их не поддерживает.

По умолчанию СУБД MySQL работает в режиме автоматического завершения транзакций, т. е. как только выполняется оператор обновления данных, который модифицирует таблицу, MySQL тут же сохраняет изменения на диске. Для объединения в транзакцию нескольких операторов необходимо отключить этот режим. Это можно осуществить при помощи системной пере-

менной AUTOCOMMIT, значение которой выставляется при помощи оператора SET (листинг 10.5).

Листинг 10.5. Отключение режима автоматического завершения транзакций

```
SET AUTOCOMMIT=0;
```

После отключения режима автоматического завершения транзакций следует использовать оператор COMMIT, чтобы сохранять изменения на диске, либо ROLLBACK, чтобы отменять изменения, выполненные с момента начала транзакции (листинг 10.6). Для того чтобы включить режим автоматического завершения транзакций, необходимо выполнить оператор SET AUTOCOMMIT=1.

Листинг 10.6. Использование транзакций

```
SET AUTOCOMMIT=0;
INSERT INTO catalogs VALUES(NULL,'Периферия');
INSERT INTO catalogs VALUES(NULL,'Разное');
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
|          6 | Периферия          |
|          7 | Разное             |
+------------+--------------------+
ROLLBACK;
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
INSERT INTO catalogs VALUES(NULL,'Периферия');
INSERT INTO catalogs VALUES(NULL,'Разное');
COMMIT;
```

```
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
|          9 | Периферия          |
|         10 | Разное             |
+------------+--------------------+
SET AUTOCOMMIT=1;
```

В листинге 10.6 в таблицу `catalogs` добавляются новые записи, при этом выполнение оператора `ROLLBACK` позволяет отменить все изменения, которые были произведены над таблицей `catalogs`. Напротив, оператор `COMMIT` сохраняет изменения на диске, и уже следующий оператор `ROLLBACK` вернет последнее сохраненное состояние.

Для того чтобы включить режим автоматического завершения транзакций только для отдельной последовательности операторов, можно воспользоваться оператором `START TRANSACTION` (листинг 10.7).

Листинг 10.7. Использование оператора `START TRANSACTION`

```
START TRANSACTION;
SELECT @total := count FROM products
WHERE name = 'Celeron D 320 2.4GHz';
+-----------------+
| @total := count |
+-----------------+
|               1 |
+-----------------+
UPDATE products SET count = @total - 1
WHERE name = 'Celeron D 320 2.4GHz';
COMMIT;
```

После выполнения оператора `START TRANSACTION` режим автоматического завершения транзакций остается включенным до явного завершения транзакции с помощью оператора `COMMIT` или отката транзакции при помощи `ROLLBACK`.

Замечание

Оператор `SET TRANSACTION` возник в СУБД MySQL, начиная с версии 4.0.11, и имеет два синонима: `BEGIN` и `BEGIN WORK`, появившиеся еще в версии 3.23. Однако рекомендуется использовать именно `SET TRANSACTION`.

Для некоторых операторов нельзя выполнить откат при помощи оператора ROLLBACK. К их числу относят CREATE INDEX, DROP INDEX, CREATE TABLE, DROP TABLE, TRUNCATE TABLE, ALTER TABLE, RENAME TABLE, CREATE DATABASE, DROP DATABASE, ALTER DATABASE. Следует избегать помещать их в транзакции с другими операторами.

Кроме того, существует ряд операторов, которые неявно завершают транзакцию, как если бы был вызван оператор COMMIT. К их числу относятся: ALTER TABLE, BEGIN, CREATE INDEX, CREATE TABLE, CREATE DATABASE, DROP DATABASE, DROP INDEX, DROP TABLE, DROP DATABASE, LOAD MASTER DATA, LOCK TABLES, RENAME, SET AUTOCOMMIT=1, START TRANSACTION, TRUNCATE TABLE.

Замечание

Операторы CREATE TABLE, TRUNCATE TABLE, DROP DATABASE и CREATE DATABASE приводят к неявному завершению транзакции, начиная с версий MySQL 4.1.13 для ветки 4.1.*x* и 5.0.8 для ветки 5.0.*x* соответственно.

Транзакции не могут быть вложенными. Это связано с тем, что любой оператор, начинающий транзакцию, приводит к завершению предыдущей транзакции.

Начиная с версий MySQL 4.0.14 и 4.1.1, для таблиц типа InnoDB поддерживаются операторы SAVEPOINT и ROLLBACK TO SAVEPOINT, которые позволяют работать с именованными точками начала транзакции (листинг 10.8).

Листинг 10.8. Работа с именованными точками начала транзакции

```
SAVEPOINT cheops;
INSERT INTO catalogs VALUES(NULL,'Периферия');
INSERT INTO catalogs VALUES(NULL,'Разное');
ROLLBACK TO SAVEPOINT cheops;
```

В листинге 10.8 оператор SAVEPOINT устанавливает именованную точку начала транзакции с именем cheops. Если текущая транзакция имеет точку сохранения с таким же именем, старая точка удаляется и устанавливается новая. Оператор ROLLBACK TO SAVEPOINT cheops откатывает транзакцию к состоянию, в котором находилась база данных на момент установки именованной точки, с помощью оператора SAVEPOINT. Все точки сохранения транзакций удаляются, если выполняется оператор COMMIT или ROLLBACK без указания имени точки сохранения.

10.2. Когда можно обойтись без транзакций?

Использовать транзакции не всегда возможно, например, используемый тип таблиц может их не поддерживать. В этом случае можно попытаться пере-

формулировать последовательность SQL-запросов таким образом, чтобы они выполнялись в один запрос.

Листинг 10.9. Покупка товарной позиции

```
START TRANSACTION;
SELECT id_product, name, @total := count FROM products
WHERE name = 'Celeron D 320 2.4GHz';
UPDATE products SET count = @total — 1
WHERE name = 'Celeron D 320 2.4GHz';
COMMIT;
```

Представленные в листинге 10.9 запросы, которые осуществляются при покупке товарной позиции, можно заменить одним запросом, представленным в листинге 10.10.

Листинг 10.10. Альтернативный запрос для покупки товарной позиции

```
UPDATE products SET count = count — 1
WHERE id_product = 'Celeron D 320 2.4GHz' AND count > 0;
Query OK, 1 row affected (0.00 sec)
```

`UPDATE`-запрос изменяет количество товарных позиций "'Celeron D 320 2.4GHz", если их количество больше 0. В противном случае изменения записей производиться не будут. Отслеживать это состояние можно с помощью отчета по количеству затронутых строк в таблице.

В большинстве случаев можно преобразовать несколько запросов в один, воспользовавшись многотабличными или вложенными запросами.

10.3. Блокировка таблиц

Для таблиц типа MyISAM использование транзакций недоступно. Однако их можно эмулировать при помощи операторов `LOCK TABLES` и `UNLOCK TABLES`. В отличие от полноценных транзакций, данные операторы блокируют всю таблицу, в результате чего никто не может работать с таблицами до тех пор, пока они остаются заблокированными. Оператор `LOCK TABLES` выполняет блокировку таблиц, а `UNLOCK TABLES` снимает блокировку (листинг 10.11).

Замечание

Все таблицы, заблокированные в текущем соединении, разблокируются при повторном вызове оператора `LOCK TABLES`.

Замечание

Операторы `LOCK TABLES` и `UNLOCK TABLES` имеют синонимы: `LOCK TABLE` и `UNLOCK TABLE` соответственно.

Листинг 10.11. Использование операторов `LOCK TABLES` и `UNLOCK TABLES`

```
LOCK TABLES catalogs WRITE;
INSERT INTO catalogs VALUES(NULL,'Периферия');
INSERT INTO catalogs VALUES(NULL,'Разное');
UNLOCK TABLES;
```

Листинг 10.11 демонстрирует блокировку таблицы `catalogs` на время добавления данных в базу данных. Следует обратить внимание, что после оператора `LOCK TABLES` записывается имя блокируемой таблицы, при снятии блокировки при помощи оператора `UNLOCK TABLES` указание имени таблицы уже не требуется. Основная причина применения блокировки таблицы при помощи `LOCK TABLES` — это увеличение скорости обновления таблиц и добавления больших объемов данных.

При использовании блокировок можно явно указать тип блокировки — на чтение (`READ`) или на запись (`WRITE`) (листинг 10.12).

Листинг 10.12. Детализация типа блокировки

```
LOCK TABLES catalogs WRITE;
INSERT INTO catalogs VALUES(NULL,'Периферия');
INSERT INTO catalogs VALUES(NULL,'Разное');
UNLOCK TABLES catalogs;
LOCK TABLES catalogs READ;
INSERT INTO catalogs VALUES(NULL,'Периферия');
INSERT INTO catalogs VALUES(NULL,'Разное');
UNLOCK TABLES;
```

Различие между блокировками на чтение (`READ`) и запись (`WRITE`) заключается в том, что клиент, установивший блокировку на чтение, и остальные клиенты могут только читать из таблицы данные. При блокировке на запись (`WRITE`) установивший ее клиент может как вносить записи в таблицу, так и читать, в то время как доступ других клиентов к таблице блокируется. Причем все остальные клиенты ожидают, когда блокировка будет отменена оператором `UNLOCK TABLES`. Следует отметить, что время ожидания может быть значительным и это нужно предусмотреть в клиентской программе (листинг 10.13).

Листинг 10.13. Время блокировки может быть значительным

```
mysql> LOCK TABLES products READ;
Query OK, 0 rows affected (331.67 sec)
```

Один оператор `LOCK TABLES` может блокировать сразу несколько таблиц, причем рекомендуется блокировать все таблицы, которые участвуют в запросах внутри блокировки. Это связано с тем, что пока блокировка, установленная `LOCK TABLES`, активна, невозможно получить доступ ни к каким таблицам, которые не были блокированы этим оператором (листинг 10.14).

Листинг 10.14. Блокировка нескольких таблиц

```
LOCK TABLES catalogs WRITE, products WRITE;
SELECT * FROM catalogs, products
WHERE products.id_catalog = catalogs.id_catalogs
INSERT INTO catalogs VALUES(NULL,'Разное');
UNLOCK TABLES catalogs;
```

Если запрос обращается к таблице через псевдоним, требуется блокировать таблицу, используя тот же псевдоним. Блокировка таблицы не будет работать, если не указан псевдоним (листинг 10.15).

Листинг 10.15. Ошибка при использовании псевдонима

```
LOCK TABLES products READ;
SELECT * FROM products AS p;
ERROR 1100: Table 'p' was not locked with LOCK TABLES
```

И наоборот, если таблица блокирована по псевдониму, к ней следует обращаться, используя этот псевдоним (листинг 10.16).

Листинг 10.16. Обращение к таблице по псевдониму

```
mysql> LOCK TABLE products AS p READ;
mysql> SELECT * FROM products;
ERROR 1100: Table 'products' was not locked with LOCK TABLES
```

Блокировки по записи (`WRITE`) обычно имеют более высокий приоритет, чем блокировки по чтению (`READ`), чтобы гарантировать, как можно, более быстрое обновление данных. Это означает, что если один или несколько клиентов устанавливают блокировку на чтение (`READ`), а затем другой клиент устанавливает блокировку на запись (`WRITE`) по этим же таблицам, то все остальные клиенты будут ожидать, пока блокировка по записи не будет снята.

Можно использовать ключевое слово LOW_PRIORITY совместно с ключевым словом WRITE, чтобы позволить другим клиентам устанавливать блокировку по чтению, пока клиент ожидает возможности установки блокировки по записи (листинг 10.17).

Листинг 10.17. Использование ключевого слова LOW_PRIORITY

```
LOCK TABLES catalogs LOW_PRIORITY WRITE;
INSERT INTO catalogs VALUES(NULL,'Периферия');
UNLOCK TABLES catalogs;
```

Однако при интенсивном обращении к серверу, когда таблицы постоянно блокированы по чтению, клиент, установивший блокировку с низким приоритетом, может не дождаться своей очереди.

ГЛАВА 11

Администрирование MySQL

11.1. Создание и удаление пользователей

СУБД MySQL является многопользовательской средой. Это означает, что для доступа к таблицам базы данных могут быть созданы различные учетные записи с разным уровнем привилегий. Создать новую учетную запись можно при помощи оператора `CREATE USER`, который имеет следующий синтаксис:

```
CREATE USER user [IDENTIFIED BY [PASSWORD] 'password']
```

Оператор создает учетную запись *user* с необязательным паролем *password*. В листинге 11.1 представлен запрос, создающий пользователя `softtime`.

Замечание

Оператор `CREATE USER` появился в СУБД MySQL, начиная с версии 5.0.2.

Листинг 11.1. Создание новой учетной записи при помощи оператора `CREATE USER`

```
CREATE USER softtime;
```

Если пароль не указывается, по умолчанию в его качестве выступает пустая строка. После того, как новая учетная запись создана, в любом клиенте можно авторизоваться с именем пользователя `softtime` (листинг 11.2).

Листинг 11.2. Авторизация пользователя `softtime`

```
C:\mysql\bin>mysql -u softtime
```

Для того чтобы задать пароль, нужно использовать ключевое слово `IDENTIFIED BY`, за которым в одиночных кавычках следует пароль (листинг 11.3).

Листинг 11.3. Защита учетной записи паролем

```
CREATE USER softtime IDENTIFIED BY '1234567';
```

Пароль, заданный в листинге 11.3 для учетной записи `softtime`, будет храниться в виде обычного текста. Разумнее хранить пароль в виде хэш-кода, полученного в результате необратимого шифрования. При авторизации пароль, вводимый пользователем, также подвергается необратимому шифрованию, после чего сравнению подвергаются хэш-коды. При такой системе авторизации злоумышленник, получивший в руки хэш-коды, вынужден будет потратить время на расшифровку паролей пользователя путем перебора. Если пароли выбраны правильно, т. е. не являются осмысленной фразой, а представляют набор букв и цифр, время, которое может потребоваться на их расшифровку, сделает их неактуальными. Для того чтобы воспользоваться таким механизмом авторизации, необходимо поместить между ключевым словом `IDENTIFIED BY` и паролем ключевое слово `PASSWORD` (листинг 11.4), которое сообщает, что пароль перед сохранением следует пропустить через функцию `PASSWORD()`.

Замечание

При необратимом шифровании можно использовать только функцию `PASSWORD()`, применение других функций не предусмотрено.

Листинг 11.4. Использование ключевого слова `PASSWORD`

```
CREATE USER wet IDENTIFIED BY PASSWORD '123456';
```

При попытке создания учетной записи с уже существующим именем СУБД MySQL возвращает ошибку (листинг 11.5).

Листинг 11.5. Ошибка при создании уже существующей учетной записи

```
CREATE USER wet IDENTIFIED BY '123456';
ERROR 1396: Operation CREATE USER failed for 'wet'@'%'
```

Однако созданные при помощи оператора `CREATE USER` учетные записи не обладают никакими привилегиями — с использованием такой учетной записи невозможно просматривать таблицы и осуществлять запросы. Для наделения

учетной записи привилегиями необходимо воспользоваться оператором GRANT *(см. разд. 11.5)*.

Один оператор CREATE USER позволяет создать сразу несколько записей. Для этого необходимо перечислить определения учетных записей через запятую, как это продемонстрировано в листинг 11.6, где создаются сразу три учетные записи.

Листинг 11.6. Создание нескольких учетных записей

```
CREATE USER wet IDENTIFIED BY PASSWORD '123456',
            ret IDENTIFIED BY PASSWORD '654321',
            tor IDENTIFIED BY PASSWORD '678451';
```

Оператор DROP USER (листинг 11.7) позволяет удалить учетную запись *user* и имеет следующий синтаксис:

```
DROP USER user
```

Замечание

Оператор DROP USER появился в СУБД MySQL, начиная с версии 4.1.1.

Листинг 11.7. Использование оператора DROP USER

```
DROP USER wet;
```

Оператор DROP USER автоматически не закрывает соединение удаляемого пользователя, который сможет работать с базой данных до тех пор, пока не будет закрыто его соединение или связь с сервером не оборвется.

Точно так же, как и оператор CREATE USER, оператор DROP USER позволяет удалять сразу несколько учетных записей.

Изменять имя уже существующей учетной записи можно при помощи оператора RENAME USER, который имеет следующий синтаксис:

```
RENAME USER old_user TO new_user
```

Здесь *old_user* — старое имя пользователя, а *new_user* — новое имя пользователя. Пример вызова оператора RENAME USER приведен в листинге 11.8.

Замечание

Оператор RENAME USER введен в СУБД MySQL, начиная с версии 5.0.2.

Листинг 11.8. Использование оператора RENAME USER

```
RENAME USER wet TO wet1;
```

11.2. Назначение пароля

Обладая учетной записью root, можно назначить пароль пользователю при помощи оператора SET PASSWORD и функции PASSWORD(). В листинге 11.9 для пользователя user назначается новый пароль password.

Листинг 11.9. Назначение нового пароля

```
SET PASSWORD FOR user = PASSWORD('password')
```

11.3. Восстановление утерянного пароля

Если утерян пароль обычного пользователя, авторизовавшись с использованием учетной записи root, можно назначить пользователю пароль, как это описывается в *разд. 11.2*. Если утерян пароль root, необходимо запустить сервер с параметром --skip-grant-tables (в конфигурационном файле my.ini следует добавить директиву skip-grant-tables в секции [mysqld]), в этом режиме сервер игнорирует таблицу привилегий и все базы данных (в том числе и системная база данных mysql) доступны без привилегий. В этом случае оператор SET PASSWORD можно выполнять из-под любого пользователя по отношению к любому другому пользователю, в том числе и root. По завершению восстановительных работ сервера следует перезагрузить MySQL.

Замечание

Восстановить пароль можно также при помощи утилиты mysqladmin, выполнив команду mysqladmin -u root password "password".

11.4. Удаленный доступ к MySQL

Следует отменить, что учетная запись является составной и принимает форму *'username'@'host'*, где *username* — имя, а *host* — наименование хоста, с которого пользователю *username* разрешено обращаться к серверу MySQL. Так, учетные записи 'root'@'127.0.0.1' и 'wet'@'62.78.56.34' означают, что пользователь с именем root может обращаться с хоста, на котором расположен сервер, а wet — только с хоста с IP-адресом 62.78.56.34, ни с какого другого хоста обратиться к серверу MySQL с этим именем нельзя, в том числе и с машины, где расположен сам сервер.

Замечание

Если имя пользователя и хост не содержат специальных символов — и %, их необязательно брать в кавычки — root@127.0.0.1.

Замечание

IP-адрес `127.0.0.1` всегда относится к локальному хосту — если сервер и клиент установлены на одном хосте и сеть не используется, то сервер слушает соединения по этому адресу, а клиент отправляет на него SQL-запросы. Во всех операционных системах IP-адрес `127.0.0.1` имеет псевдоним `localhost`, поэтому вместо IP-адреса можно указывать этот псевдоним, а учетные записи вида `'root'@'127.0.0.1'` записывать так: `'root'@'localhost'`.

Если к IP-адресу хоста привязано какое-то доменное имя, например `'softtime.ru'`, то вместо данного хоста можно использовать доменное имя `'root'@'softtime.ru'` — такая учетная запись означает, что, зарегистрировавшись с именем `root`, можно обращаться к серверу с хоста, доменным именем которого является `softtime.ru`.

Если в дополнение к двум хостам `127.0.0.1` и `62.78.56.34` требуется разрешить пользователю `'wet'` обращаться к серверу MySQL с третьего хоста `158.0.55.62`, то потребуется создать три учетные записи: `'wet'@'127.0.0.1'`, `'wet'@'62.78.56.34'` и `'wet'@'158.0.55.62'`. Число адресов, с которых необходимо обеспечить доступ пользователю, может быть значительным и включать целые диапазоны. Для задания диапазона в имени хоста используется специальный символ `%`, выполняющий ту же функцию, что и в операторе `LIKE`. Так, учетная запись `'wet'@'%'` разрешает пользователю `'wet'` обращаться к серверу MySQL с любых компьютеров сети. Символ `%` позволяет задавать диапазоны, поэтому учетная запись `'wet'@'%.softtime.ru'` разрешает обращаться к серверу MySQL поддоменам домена `softtime.ru`. Учетная запись `'wet'@'62.78.56.%'` позволяет пользователю `'wet'` получить доступ с хостов, обладающих IP-адресами в диапазоне от `62.78.56.0` до `62.78.56.255`.

Отсутствие части *`host`* в учетной записи аналогично использованию `%`, таким образом, учетные записи `'wet'@'%'` и `'wet'` эквивалентны.

По умолчанию при использовании оператора `CREATE USER` *(см. разд. 11.1)* и `GRANT` *(см. разд. 11.5)*, если не указывается хост, создается сетевая учетная запись, т. е. доступ разрешается с любого хоста сети. Для того чтобы создать учетную запись, разрешающую обращаться к серверу MySQL лишь с определенного хоста, например, только с локального хоста (`localhost`), следует явно указать это в операторе `CREATE USER` (листинг 11.10).

Листинг 11.10. Создание нескольких учетных записей

```
CREATE USER 'wet'@'localhost' IDENTIFIED BY PASSWORD '123456',
            'ret'@'localhost' IDENTIFIED BY PASSWORD '654321',
            'tor'@'localhost' IDENTIFIED BY PASSWORD '678451';
```

11.5. Управление привилегиями пользователей

Рассмотренные ранее операторы позволяют создавать, удалять и редактировать учетные записи, однако они не разрешают изменять привилегии пользователя, т. е. сообщать СУБД MySQL, какой пользователь имеет право только на чтение информации, какой на чтение и редактирование, а кому предоставлены права изменять структуру базы данных и создавать учетные записи для остальных пользователей.

Для решения этих задач предназначены операторы `GRANT` и `REVOKE`: оператор `GRANT` назначает привилегии пользователю, а `REVOKE` — удаляет. Если учетная запись, которая появляется в операторе `GRANT`, не существует, то она автоматически создается. Однако удаление всех привилегий с помощью оператора `REVOKE` не приводит к автоматическому уничтожению учетной записи — для полного удаления пользователя необходимо воспользоваться оператором `DROP USER`.

В простейшем случае оператор `GRANT` выглядит так, как это представлено в листинге 11.11.

Листинг 11.11. Использование оператора `GRANT`

```
GRANT ALL ON *.* TO 'wet'@'localhost' IDENTIFIED BY 'pass';
```

Запрос в листинге 11.11 создает пользователя с именем `wet` и паролем `pass`. Этот пользователь может обращаться к серверу MySQL с локального хоста (`localhost`) и имеет все права (`ALL`) для всех баз данных (`*.*`). Если такой пользователь уже существует, то его привилегии будут изменены на `ALL`.

Замечание

На данный момент оператор `GRANT` поддерживает имена хостов, баз данных, таблиц и столбцов размером до 60 символов. Имя пользователя не может быть длиннее 16 символов.

Отобрать права у пользователя `'wet'@'localhost'` можно при помощи оператора `REVOKE`, представленного в листинге 11.12.

Листинг 11.12. Использование оператора `REVOKE`

```
REVOKE ALL ON *.* FROM 'wet'@'localhost';
```

Вместо ключевого слова `ALL`, которое обозначает все системные привилегии (кроме `GRANT OPTION` — передача привилегий другим пользователям), можно использовать любое из ключевых слов, представленных в табл. 11.1.

Таблица 11.1. *Системные привилегии, используемые в операторах* GRANT *и* REVOKE

Привилегия	Операция, разрешенная привилегией
ALL [PRIVILEGES]	Комбинация всех привилегий за исключением привилегии GRANT OPTION, которая всегда задается отдельно или с предложением WITH GRANT OPTION
ALTER	Позволяет редактировать таблицу при помощи оператора ALTER TABLE
ALTER ROUTINE	Позволяет редактировать или удалять хранимую процедуру
CREATE	Позволяет создавать таблицу при помощи оператора CREATE TABLE
CREATE ROUTINE	Позволяет создавать хранимую процедуру
CREATE TEMPORARY TABLES	Позволяет создавать временные файлы
CREATE USER	Позволяет работать с учетными записями при помощи операторов CREATE USER, DROP USER, RENAME USER и REVOKE ALL PRIVILEGES
CREATE VIEW	Позволяет создавать представление при помощи оператора CREATE VIEW
DELETE	Позволяет удалять записи при помощи оператора DELETE
DROP	Позволяет удалять таблицы при помощи оператора DROP TABLE
EVENT	Позволяет создать события для планировщика заданий
EXECUTE	Позволяет выполнять хранимые процедуры
FILE	Позволяет работать с файлами при помощи операторов SELECT...INTO OUTFILE и LOAD DATA INFILE
INDEX	Позволяет работать с индексами, в частности, использовать операторы CREATE INDEX и DROP INDEX
INSERT	Позволяет добавлять в таблицу новые записи при помощи оператора INSERT
LOCK TABLES	Позволяет осуществлять блокировки таблиц при помощи операторов LOCK TABLES и UNLOCK TABLES. Для вступления в действие этой привилегии должна быть установлена привилегия SELECT
PROCESS	Позволяет использовать оператор SHOW FULL PROCESSLIST
REFERENCES	Зарезервировано, но не реализовано
RELOAD	Позволяет использовать оператор FLUSH для обновления таблиц разрешений, кэша и журналов
REPLICATION CLIENT	Позволяет запрашивать, является ли сервер главным или подчиненным в репликации

Таблица 11.1 *(окончание)*

Привилегия	Операция, разрешенная привилегией
REPLICATION SLAVE	Данная привилегия необходима для подчиненных серверов репликации
SELECT	Позволяет осуществлять выборки таблиц при помощи оператора SELECT
SHOW DATABASES	Позволяет просматривать список всех таблиц на сервере MySQL при помощи оператора SHOW DATABASES
SHOW VIEW	Позволяет использовать оператор SHOW CREATE VIEW
SHUTDOWN	Позволяет завершать работу сервера при помощи mysqladmin shutdown
SUPER	Позволяет выполнять административные функции при помощи операторов CHANGE MASTER, KILL, PURGE MASTER LOGS и SET GLOBAL
TRIGGER	Позволяет создавать и уничтожать триггеры *(см. главу 13)*
UPDATE	Позволяет обновлять содержимое таблиц при помощи оператора UPDATE
USAGE	Синоним для статуса "отсутствуют привилегии"
GRANT OPTION	Позволяет управлять привилегиями других пользователей, без данной привилегии невозможно выполнить операторы GRANT и REVOKE

Замечание

Привилегии CREATE TEMPORARY TABLES, EXECUTE, LOCK TABLES, REPLICATION CLIENT, REPLICATION SLAVE, SHOW DATABASES и SUPER были добавлены в версии 4.0.2. (Привилегия EXECUTE не выполняла никаких действий до версии 5.0.3.) Привилегии CREATE VIEW и SHOW VIEW были добавлены в версии 5.0.1, а CREATE USER, CREATE ROUTINE и ALTER ROUTINE — в версии 5.0.3. Привилегии EVENT и TRIGGER были добавлены в версии 5.1.6.

Рассмотрим несколько примеров. Для того чтобы разрешить пользователю editor полный доступ на просмотр, заполнение, редактирование и удаление таблиц, необходимо воспользоваться запросом, представленным в листинге 11.13.

Листинг 11.13. Предоставление привилегий для пользователя editor

```
GRANT SELECT, INSERT, DELETE, UPDATE ON *.* TO editor;
```

Если в качестве пользователя выступает приложение, которое добавляет новые записи или обновляет текущие, то его права можно ограничить лишь

привилегиями INSERT и UPDATE (листинг 11.14). Это позволит избежать выполнения операций DELETE и SELECT, которыми может воспользоваться злоумышленник при помощи инъекционного кода.

Листинг 11.14. Предоставление привилегий INSERT и UPDATE

```
GRANT INSERT, UPDATE ON *.* TO program;
```

Для того чтобы назначить все привилегии сразу, следует воспользоваться запросами, представленными в листинге 11.15.

Листинг 11.15. Предоставление всех привилегий

```
GRANT ALL ON *.* TO superuser;
GRANT GRANT OPTION ON *.* TO superuser;
```

Выполнить операции по наделению пользователя всеми правами в одном запросе не получится, т. к. ключевое слово ALL всегда употребляется отдельно и не должно использоваться совместно с другими ключевыми словами из табл. 11.1 (листинг 11.16).

Листинг 11.16. Ошибка при совместном использовании ALL и GRANT OPTION

```
mysql> GRANT ALL, GRANT OPTION ON *.* TO superuser;
ERROR 1064: You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near '
GRANT OPTION ON *.* TO superuser' at line 1
```

Однако использование ключевого слова USEAGE, означающего полное отсутствие привилегий, совместно с другими ключевыми словами из табл. 11.1 вполне допустимо (листинг 11.17).

Листинг 11.17. Совместное использование ключевого слова USAGE с SELECT

```
GRANT USAGE, SELECT ON *.* TO superuser;
```

Следует отметить, что один оператор GRANT может быть использован для предоставления привилегий сразу нескольким пользователям (листинг 11.18).

Листинг 11.18. Предоставление привилегий сразу нескольким пользователям

```
GRANT ALL OPTION ON *.* TO superuser, wet, toor;
```

Ключевое слово ON в операторе GRANT задает уровень привилегий, которые могут быть заданы на одном из четырех уровней, представленных в табл. 11.2.

Таблица 11.2. *Уровни привилегий*

Ключевое слово ON	**Уровень**
ON *.*	Глобальный уровень — касается всех баз данных и таблиц, входящих в их состав. Таким образом, пользователь с полномочиями на глобальном уровне может обращаться ко всем базам данных
ON *	Если текущая база данных не была выбрана при помощи оператора USE, данное предложение эквивалентно ON *.*, если произведен выбор текущей базы данных, то устанавливаемые при помощи оператора GRANT привилегии относятся ко всем таблицам текущей базы данных
ON *db*.*	Уровень базы данных — это предложение означает, что привилегии распространяются на таблицы базы данных *db*
ON *db.tbl*	Уровень таблицы — предложение означает, что привилегии распространяются на таблицу *tbl* базы данных *db*
ON *db.tbl*	Уровень столбца — привилегии уровня столбца касаются отдельных столбцов в таблице *tbl* базы данных *db*. Список столбцов указывается в скобках через запятую после ключевых слов SELECT, INSERT, UPDATE

Привилегии EXECUTION, FILE, PROCESS, RELOAD, REPLICATION CLIENT, REPLICATION SLAVE, SHOW DATABASES, SHUTDOWN и SUPER могут быть установлены лишь на глобальном уровне. Для таблиц можно установить только следующие типы привилегий: SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, GRANT OPTION, INDEX и ALTER. Это следует учитывать при использовании конструкции GRANT ALL, которая назначает привилегии на текущем уровне. Так, запрос уровня базы данных GRANT ALL ON *db*.* не предоставляет никаких глобальных привилегий, таких как FILE или RELOAD.

Замечание

Не обязательно запоминать контекст применения привилегий. Доступные в текущей версии привилегии, а также контекст их применения можно уточнить при помощи оператора SHOW PRIVILEGES. Данный оператор введен в MySQL, начиная с версии 4.1.0.

В листинге 11.19 на примере создания пользователя editor демонстрируется использование привилегий уровня базы данных (test).

Листинг 11.19. Предоставление привилегий уровня базы данных

```
GRANT ALL ON test.* TO editor;
```

Замечание

Следует отметить, что отсутствие конструкции IDENTIFIED BY приводит к тому, что в качестве пароля пользователя назначается пустая строка.

Если у пользователя нет привилегий на доступ к базе данных, имя базы данных не отображается в ответ на запрос SHOW DATABASES, если только у пользователя нет специальной привилегии SHOW DATABASES.

Следующий уровень — это привилегии на уровне таблицы. В листинге 11.20 представлен запрос, который наделяет учетную запись manager полными привилегиями для доступа к таблице user базы данных test.

Листинг 11.20. Предоставление привилегий уровня таблицы

```
GRANT ALL ON test.user TO manager;
```

Если у пользователя нет привилегий на доступ к таблице, то эта таблица не отображается в ответ на запрос списка таблиц базы данных — SHOW TABLES.

В именах баз данных и таблиц допускается использование символов % и _, которые имеют тот же смысл, что и в операторе LIKE, т. е. заменяют произвольное число и один символ соответственно. Это означает, что если требуется использовать символ _ как часть имени базы данных, его необходимо экранировать обратным слэшем, иначе можно нечаянно предоставить доступ к базам данных, соответствующим введенному шаблону. Например, для базы данных list_user оператор GRANT может выглядеть так, как это представлено в листинге 11.21.

Листинг 11.21. Символ _ в имени базы данных

```
GRANT ALL ON 'list\_user'.* TO 'wet'@'localhost';
```

Еще более гибкие средства для назначения привилегий предоставляются на уровне столбца. Для того чтобы предоставить доступ для столбцов, требуется перечислить их имена через запятую после ключевых слов SELECT, INSERT и UPDATE — использовать другие ключевые слова для предоставления привилегий на уровне столбца нельзя. Пусть пользователю wet предоставляется привилегия просмотра (SELECT) столбца name таблицы catalogs базы данных test. В этом случае запрос, создающий такого пользователя, может выглядеть так, как это представлено в листинге 11.22.

Листинг 11.22. Предоставление привилегий уровня столбца

```
GRANT SELECT(name) ON test.catalogs TO 'wet'@'localhost';
```

После этого, авторизовавшись с использованием учетной записи wet, просмотреть запросы с явным или неявным участием столбцов, отличных от name, в таблице shop.catalogs уже не удастся (листинг 11.23).

Листинг 11.23. Пользователь wet имеет право просматривать только столбец name

```
SELECT * FROM catalogs;
ERROR 1143: SELECT command denied to user 'wet'@'localhost' for column
'id_catalog' in table 'catalogs'
SELECT name FROM catalogs;
+--------------------+
| name               |
+--------------------+
| Процессоры         |
| Материнские платы  |
| Видеоадаптеры      |
| Жесткие диски      |
| Оперативная память |
+--------------------+
```

Как видно из листинга 11.23, запрос SELECT * FROM catalogs заканчивается неудачей с сообщением о том, что пользователь 'wet'@'localhost' не имеет прав на просмотр содержимого столбца id_catalog. Однако запрос SELECT name FROM catalogs завершается успешно, т. к. в списке столбцов присутствует только столбец name, который пользователь wet имеет право просматривать. Такой вид прав доступа удобно использовать для скрытия конфиденциальной информации, например, пароля или счета.

Особняком в привилегиях стоит привилегия GRANT OPTION — способность наделять других пользователей правами на передачу пользовательских прав. Обычно эту привилегию назначают при помощи отдельного запроса (см. листинг 11.15), но язык запросов SQL предоставляет специальное предложение WITH GRANT OPTION, которое позволяет наделять учетную запись этой привилегией. Предложение WITH GRANT OPTION помещается в конце запроса (листинг 11.24).

Листинг 11.24. Использование предложения WITH GRANT OPTION

```
GRANT ALL ON test.* TO 'wet'@'localhost' WITH GRANT OPTION;
```

В результате выполнения запроса из листинга 11.24 пользователь wet наделяется правами вызова оператора GRANT ALL для предоставления привилегий другим пользователям на базу данных test и ее таблицы. Ни на какие другие

базы данных пользователь wet не имеет права выдавать привилегии. Другими словами, пользователь, обладающий правами GRANT OPTION, может передавать другим пользователям только те права, которые принадлежат ему самому.

Следует крайне осторожно обращаться с привилегией GRANT OPTION, т. к. она распространяется не только на те привилегии, которыми пользователь обладает на настоящий момент, но и на все привилегии, которые он может получить в будущем.

Для отмены привилегий учетной записи предназначен оператор REVOKE (листинг 11.25). Его синтаксис похож на синтаксис оператора GRANT с той лишь разницей, что ключевое слово TO заменено на FROM, а предложения IDENTIFIED BY, REQUIRE и WITH GRANT OPTION отсутствуют.

Листинг 11.25. Использование оператора REVOKE

```
REVOKE DELETE, UPDATE ON shop.* FROM 'wet'@'localhost';
```

Следует помнить, что оператор REVOKE отменяет привилегии, но не удаляет учетные записи, для их удаления необходимо воспользоваться оператором DROP USER.

До версии 4.1.2 нельзя было удалить сразу все привилегии, необходимо было выполнить два оператора (листинг 11.26).

Листинг 11.26. Удаление всех привилегий

```
REVOKE ALL ON shop.* FROM 'wet'@'localhost';
REVOKE GRANT OPTION ON shop.* FROM 'wet'@'localhost';
```

Начиная с версии 4.1.2, был добавлен синтаксис, позволяющий удалять сразу все привилегии при помощи одного запроса (листинг 11.27).

Листинг 11.27. Альтернативный синтаксис удаления всех привилегий

```
REVOKE ALL, GRANT OPTION ON shop.* FROM 'wet'@'localhost';
```

Синтаксис оператора REVOKE позволяет удалять привилегии сразу для нескольких пользователей (листинг 11.28).

Листинг 11.28. Альтернативный синтаксис удаления всех привилегий

```
REVOKE ALL, GRANT OPTION ON shop.*
FROM 'wet'@'localhost',
     'toor'@'localhost',
     'ret'@'localhost';
```

11.6. Просмотр существующих привилегий

Для просмотра существующих привилегий необходимо выполнить оператор SHOW GRANTS (листинг 11.29). Если в результирующей таблице учетной записи нет, пользователь не обладает никакими правами, и его учетная запись может быть удалена.

Листинг 11.29. Использование оператора SHOW GRANTS

```
SHOW GRANTS;
+-------------------------------------------------------------------+
| Grants for root@localhost                                         |
+-------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+-------------------------------------------------------------------+
```

Как видно из листинга 11.29, только пользователь 'root'@'localhost' обладает привилегиями, все остальные пользователи могут быть удалены при помощи оператора DROP USER.

11.7. Ограничение на число соединений с сервером и число запросов

Ключевое слово WITH в операторе GRANT помимо привилегии GRANT OPTION позволяет наложить ограничения на число подключений, запросов, обновлений и параллельных запросов в час. Это осуществляется при помощи опций MAX_CONNECTIONS_PER_HOUR, MAX_QUERIES_PER_HOUR, MAX_UPDATES_PER_HOUR и MAX_USER_CONNECTIONS соответственно. Запрос в листинге 11.30 устанавливает для пользователя wet полный доступ к базе данных shop, но с ограничением — не больше 10 подключений к серверу MySQL в час и не более 1000 запросов, из которых только 200 могут быть операциями обновления UPDATE, а 3 запроса протекать одновременно.

Замечание

Опции MAX_QUERIES_PER_HOUR, MAX_UPDATES_PER_HOUR, MAX_CONNECTIONS_PER_HOUR появились в СУБД MySQL, начиная с версии 4.0.2. Опция MAX_USER_CONNECTIONS появилась в СУБД MySQL, начиная с версии 5.0.3.

Листинг 11.30. Ограничение учетной записи

```
GRANT ALL ON shop.* TO 'wet'@'localhost' IDENTIFIED BY 'pass'
WITH MAX_CONNECTIONS_PER_HOUR 10
     MAX_QUERIES_PER_HOUR 1000
     MAX_UPDATES_PER_HOUR 200
     MAX_USER_CONNECTIONS 3;
```

Если значение каждой из опции равно 0 (по умолчанию), это означает, что у пользователя нет ограничений по этому параметру.

MySQL позволяет шифровать весь сетевой трафик между сервером и клиентом при помощи протокола SSL.

Замечание

Протоколы Интернета, в том числе и транспортный протокол TCP, который используется в СУБД MySQL, разрабатывались достаточно давно и не были изначально предназначены для коммерции и передачи конфиденциальных сведений. Для повышения безопасности в сети компания Netscape в свое время разработала протокол обмена данными, который гарантирует защиту транзакций. Этот протокол она назвала протоколом защищенных сокетов (SSL, Secure Sockets Layer). Это промежуточный протокол, расположенный между транспортным и прикладным уровнями. Перед отправкой по сети он шифрует данные, отправляемые прикладными протоколами (HTTP, FTP, SMTP, SQL). Поэтому если злоумышленник с сетевым анализатором перехватит трафик, он получит не чистый текст, а зашифрованный. На другом конце происходит расшифровка.

Для того чтобы явно потребовать от пользователя применять те или иные средства шифрования данных, оператор `GRANT` снабжен ключевым словом `REQUIRE`, которое может принимать перечисленные далее формы.

- `REQUIRE SSL` — сообщает серверу, что использование данной учетной записи разрешается только для шифрованных SSL-подключений, попытки установить соединение по обычному нешифрованному каналу будут отвергаться.
- `REQUIRE X509` — данная опция сообщает серверу, что клиент должен иметь действительный сертификат, но какой именно, кем и кому выданный — не имеет значения.
- `REQUIRE ISSUER 'provider'` — сообщает серверу, что клиент должен обладать сертификатом, выданным сертификационным центром *provider*. Даже если сертификат действительный, но выдан кем-то другим, то сервер откажет в предоставлении доступа.
- `REQUIRE SUBJECT 'subject'` — сообщает серверу, что сертификат должен быть выдан на имя *subject*.

- `REQUIRE CIPHER 'cod'` — данная опция необходима для того, чтобы гарантировать, что будет использоваться достаточно строгий шифр и длина ключа. SSL сам по себе может быть ослаблен, если применяются старые алгоритмы с короткими ключами шифрования. Используя эту опцию, можно потребовать конкретный метод шифрования, чтобы разрешить соединение.

Для того чтобы потребовать от пользователя `root` обращаться к серверу только по зашифрованному SSL-каналу, следует воспользоваться запросом, представленным в листинге 11.31.

Листинг 11.31. Использование опции `REQUIRE SSL`

```
GRANT ALL ON *.* TO 'root'@'*'
IDENTIFIED BY 'secret' REQUIRE SSL;
```

Требование использования сертификата представлено в листинге 11.32.

Листинг 11.32. Использование опции `REQUIRE X509`

```
GRANT ALL ON *.* TO 'root'@'*'
IDENTIFIED BY 'secret' REQUIRE X509;
```

Если сертификат должен быть выдан конкретным сертификационным центром, оператор `GRANT` может выглядеть так, как это представлено в листинге 11.33. Важным требованием в этом случае является то, что строка *'provider'* должна быть представлена как единое целое и не может разрываться символом перевода строки.

Листинг 11.33. Использование опции `REQUIRE ISSUER 'provider'`

```
GRANT ALL ON *.* TO 'root'@'*'
IDENTIFIED BY 'secret'
REQUIRE ISSUER '/C=FI/ST=Some-State/L=Helsinki/ O=MySQL Finland AB/CN=Tonu
Samuel/Email=tonu@example.com';
```

Если сертификат должен быть выдан на конкретное лицо, то оператор `GRANT` может выглядеть так, как это представлено в листинге 11.34. Здесь, так же как и в предыдущем примере, не допускается разрыв символом перевода строки.

Листинг 11.34. Использование опции `REQUIRE SUBJECT 'subject'`

```
GRANT ALL ON *.* TO 'root'@'*'
IDENTIFIED BY 'secret'
```

```
REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/O=MySQL demo client
certificate/CN=Tonu Samuel/Email=tonu@example.com';
```

Применение опции `REQUIRE CIPHER 'cod'` демонстрируется в листинге 11.35.

Листинг 11.35. Использование опции `REQUIRE CIPHER 'cod'`

```
GRANT ALL ON *.* TO 'root'@'*'
IDENTIFIED BY 'secret'
REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

Опции `SUBJECT`, `ISSURE` и `CIPHER` могут комбинироваться в конструкции `REQUIRE` так, как это представлено в листинге 11.36.

Листинг 11.36. Комбинирование опций `SUBJECT`, `ISSURE` и `CIPHER`

```
GRANT ALL ON *.* TO 'root'@'*'
IDENTIFIED BY 'secret'
REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/O=MySQL demo client
certificate/CN=Tonu Samuel/Email=tonu@example.com';
AND REQUIRE ISSUER '/C=FI/ST=Some-State/L=Helsinki/ O=MySQL Finland AB/CN=Tonu
Samuel/Email=tonu@example.com';
AND REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

Замечание

Начиная с версии MySQL 4.0.4, допускается необязательное слово `AND` между опциями `REQUIRE`.

11.8. Репликация

Репликация — это процесс, в ходе которого создается копия сервера. Изменения, которые происходят на главном (master) сервере, копируются по сети на подчиненный сервер (slave), воссоздавая их. Изменять данные на подчиненном сервере (если данные обновляются) нельзя.

Подчиненный сервер может сам выступать главным по отношению к другому подчиненному серверу, в результате чего можно выстраивать цепочки из нескольких серверов. Существует несколько причин для создания репликационного сервера:

- распределение нагрузки по нескольким хостам — воссоздавая цепочку репликационных серверов с одинаковым содержимым можно снизить нагрузку, т. к. клиенты с немодифицирующими запросами (`SHOW`, `SELECT`) могут обращаться к разным хостам. Модифицирующие запросы (`INSERT`,

UPDATE, REPLACE, DELETE, CREATE TABLE и т. п.) необходимо будет проводить на основном сервере;

- резервное копирование информации — создание резервной копии часто требует блокировки таблиц, что приводит к падению производительности, которая не желательна на рабочем сервере. Для предотвращения этого можно организовать репликацию данных и снимать резервную копию с подчиненного сервера.

Для реализации репликации на главном сервере необходимо активировать бинарный журнал при помощи параметра `--log-bin` *(см. разд. 1.12.4)* или директивы `log-bin` в конфигурационном файле my.ini или my.cnf. Причем имя бинарного журнала лучше прописать явно, например `log-bin='elf-bin'`, т. к. если имя сервера (`elf`) вдруг поменяется, то в случае, когда имя не определено явно, имя бинарного журнала автоматически изменится в соответствии с новым именем хоста, что, в свою очередь, может привести к сбою процесса репликации.

Помимо этого каждый сервер в схеме репликации необходимо снабдить уникальным номером, по которому серверы будут отличаться друг от друга. Для этого используют либо параметр `--server-id` при запуске MySQL-сервера, либо директиву `server-id` конфигурационного файла my.ini или my.cnf. Параметр `server-id` может принимать целое положительное число в интервале от 1 до $2^{32} - 1$. Если не указать параметр `server-id` на одном из серверов, процесс репликации работать не будет. Секция `[mysqld]` в конфигурационном файле my.ini главного сервера может выглядеть так, как это представлено в листинге 11.37.

Замечание

После редактирования конфигурационного файла сервер необходимо перезапустить.

Листинг 11.37. Настройка главного сервера

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

Секция `[mysqld]` в конфигурационном файле my.ini на подчиненном сервере может выглядеть так, как это представлено в листинге 11.38.

Замечание

Подчиненный сервер должен находиться в выключенном состоянии, пока не будет проведена его полная настройка.

Листинг 11.38. Настройка подчиненного сервера

```
[mysqld]
server-id=2
```

Если параметр `server-id` не указывается, то он по умолчанию принимает значение 1, если не определен главный сервер, и 2 — в противном случае.

Для того чтобы репликационный сервер (пусть он имеет IP-адрес `10.0.0.2`) мог обращаться к главному (пусть он имеет IP-адрес `10.0.0.1`), на главном сервере необходимо создать учетную запись, которую подчиненный сервер мог бы использовать для загрузки обновлений (листинг 11.39).

Листинг 11.39. Создание учетной записи для репликации

```
GRANT REPLICATION SLAVE ON *.*
TO 'repl'@'10.0.0.2' IDENTIFIED BY 'slavepass';
```

Перед запуском подчиненного сервера на нем необходимо расположить точную копию данных главного сервера, при этом снять координаты в бинарном журнале — они потребуются для того, чтобы сообщить подчиненному серверу, с какой точки бинарного журнала следует начинать репликацию.

Чтобы в момент снятия копии баз данных с главного сервера и координат в бинарном журнале пользователи не выполняли операций, которые бы могли исказить данные, необходимо заблокировать таблицы для записи при помощи оператора `FLUSH TABLES WITH READ LOCK` (листинг 11.40). Оператор `FLUSH TABLES` кроме этого сохранит на жесткий диск все изменения в таблицах, которые остались в буфере в оперативной памяти.

Листинг 11.40. Использование оператора `FLUSH TABLES WITH READ LOCK`

```
FLUSH TABLES WITH READ LOCK;
```

Клиентскую программу, в которой был запущен оператор `FLUSH TABLES`, необходимо оставить работать, чтобы блокировка не была снята. Затем следует создать дамп базы данных главного сервера *(см. главу 1)*.

Замечание

Нет необходимости реплицировать все базы данных — репликации подвергнутся лишь те, которые будут развернуты на подчиненном сервере.

Пока действует блокировка по чтению, установленная командой `FLUSH TABLES WITH READ LOCK`, необходимо прочитать значение имени текущего би-

нарного журнала и смещения на главном сервере при помощи оператора SHOW MASTER STATUS (листинг 11.41).

Листинг 11.41. Использование оператора SHOW MASTER STATUS

```
SHOW MASTER STATUS;
+---------------+----------+--------------+------------------+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+---------------+----------+--------------+------------------+
| mysql-bin.003 | 73       | test         | manual,mysql     |
+---------------+----------+--------------+------------------+
```

В столбце File результирующей таблицы оператора SHOW MASTER STATUS указывается имя текущего файла журнала, в столбце Position — смещение относительно начала файла. Эти значения необходимо записать, т. к. они понадобятся позже для настройки подчиненного сервера. Эти значения представляют собой координаты репликации, с которых подчиненный сервер должен начать обработку новых изменений, полученных от главного сервера.

После того как получен SQL-дамп и зафиксированы имя журнала и смещение, можно снять блокировку и разрешить изменения баз данных для других клиентов при помощи оператора UNLOCK TABLES (листинг 11.42).

Листинг 11.42. Разблокировка таблиц

```
UNLOCK TABLES;
```

Далее необходимо развернуть базу данных главного сервера на подчиненном. Если для этого используются бинарные файлы баз данных, их следует скопировать в каталог данных подчиненного сервера (подчиненный сервер должен быть остановлен). Если используется SQL-дамп, необходимо запустить подчиненный сервер без запуска главного сервера и развернуть дамп. Для того чтобы подчиненный сервер не осуществлял попыток репликации сразу же, до того как он настроен, его необходимо запустить с параметром --skip-slave-start.

На подчиненном сервере следует выполнить оператор CHANGE MASTER TO, который настроит его для репликации (листинг 11.43).

Листинг 11.43. Подготовка подчиненного сервера к репликации

```
CHANGE MASTER TO
    MASTER_HOST='master_host_name',
    MASTER_USER='replication_user_name',
```

```
    MASTER_PASSWORD='replication_password',
    MASTER_LOG_FILE='recorded_log_file_name',
    MASTER_LOG_POS=recorded_log_position;
```

Здесь `master_host_name` — имя хоста главного сервера, `replication_user_name` — имя пользователя репликации (в листинге 11.39 он был назван `repl`), `replication_password` — пароль пользователя репликации, `recorded_log_file_name` — имя текущего бинарного журнала на главном сервере, `recorded_log_position` — позиция в журнале, начиная с которой необходимо осуществлять репликацию.

После того как все приготовления выполнены, необходимо запустить репликацию на подчиненном сервере при помощи оператора `START SLAVE` (листинг 11.44).

Листинг 11.44. Запуск репликации

```
START SLAVE;
```

Проверить успешность запуска процесса репликации на подчиненном сервере можно при помощи оператора `SHOW SLAVE STATUS` (листинг 11.45).

Листинг 11.45. Проверка репликации на подчиненном сервере

```
SHOW SLAVE STATUS\G
*************************** 1. row ***************************
       Slave_IO_State: Waiting for master to send event
          Master_Host: localhost
          Master_User: root
          Master_Port: 3306
        Connect_Retry: 3
      Master_Log_File: gbichot-bin.005
  Read_Master_Log_Pos: 79
       Relay_Log_File: gbichot-relay-bin.005
        Relay_Log_Pos: 548
Relay_Master_Log_File: gbichot-bin.005
     Slave_IO_Running: Yes
    Slave_SQL_Running: Yes
      Replicate_Do_DB:
  Replicate_Ignore_DB:
           Last_Errno: 0
           Last_Error:
         Skip_Counter: 0
  Exec_Master_Log_Pos: 79
      Relay_Log_Space: 552
```

```
      Until_Condition: None
       Until_Log_File:
        Until_Log_Pos: 0
   Master_SSL_Allowed: No
   Master_SSL_CA_File:
   Master_SSL_CA_Path:
      Master_SSL_Cert:
    Master_SSL_Cipher:
       Master_SSL_Key:
Seconds_Behind_Master: 8
```

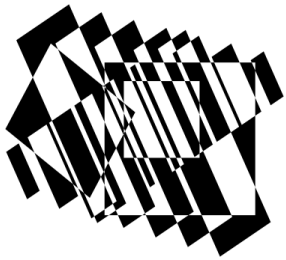
Число полей `SHOW SLAVE STATUS` в отчете оператора зависит от установленной версии MySQL. Не вдаваясь в подробности, отметим, что процесс работает успешно, если оба поля `Slave_IO_Running` (процесс загрузки данных с главного сервера) и `Slave_SQL_Running` (процесс обработки полученных данных) имеют значение `Yes`, а поле `Last_Error` пустое.

Замечание

СУБД MySQL предоставляет большое количество операторов для управления процессом репликации как со стороны главного, так и со стороны подчиненного серверов. Более подробно ознакомиться с ними можно либо в справочном руководстве, либо в нашей книге "MySQL 5"[1].

[1] Кузнецов М., Симдянов И. MySQL 5. — СПб.: БХВ-Петербург, 2006.

ГЛАВА 12



Хранимые процедуры и функции

Хранимые процедуры и функции позволяют сохранить последовательность SQL-операторов и вызывать их по имени функции или процедуры.

12.1. Создание хранимой процедуры (функции)

Для создания хранимой процедуры предназначен оператор `CREATE PROCEDURE`, который имеет следующий синтаксис:

```
CREATE PROCEDURE sp_name ([parameter[, ...]])
    [characteristic ...] routine_body
```

Хранимая функция создается при помощи аналогичного оператора `CREATE FUNCTION`:

```
CREATE FUNCTION sp_name ([parameter[, ...]])
    RETURNS type
    [characteristic ...] routine_body
```

Здесь *sp_name* — имя хранимой процедуры (функции). В скобках, следующих за именем, передается необязательный список параметров, перечисленных через запятую. Каждый параметр *parameter* позволяет передать в процедуру или из нее входные данные или результат работы функции и имеет следующий синтаксис:

```
[ IN | OUT | INOUT ] param_name type
```

Параметр *param_name* предваряет одно из ключевых слов `IN`, `OUT`, `INOUT`, которые позволяют задать направление передачи данных:

- `IN` — данные передаются строго внутрь хранимой процедуры, но если параметру с данным модификатором внутри функции присваивается новое

значение, по выходу из нее оно не сохраняется и параметр принимает значение, которое он имел до вызова процедуры;

- `OUT` — данные передаются строго из хранимой процедуры, даже если параметр имеет какое-то начальное значение, внутри хранимой процедуры это значение не принимается во внимание. С другой стороны, если параметр изменяется внутри процедуры, после вызова процедуры параметр имеет значение, присвоенное ему внутри процедуры;
- `INOUT` — параметр как принимается во внимание внутри процедуры, так и сохраняет свое значение после выхода из нее.

Замечание

Список аргументов, заключенных в круглые скобки, должен присутствовать всегда. Если аргументы отсутствуют, следует использовать пустой список аргументов `()`.

Замечание

Атрибуты `IN`, `OUT` и `INOUT` доступны лишь для хранимой процедуры, в хранимой функции все параметры всегда имеют атрибут `IN`.

После имени параметра *`param_name`* указывается его тип *`type`*, который может принимать один из допустимых в MySQL типов. По умолчанию, если ни один из модификаторов не указан, считается, что параметр объявлен с ключевым словом `IN`.

Хранимая функция, которая определяется при помощи оператора `CREATE FUNCTION`, возвращает единственное значение. Тип этого значения позволяет задать оператор `RETURN`.

Характеристика *`characteristic`* может принимать одно из следующих значений или их комбинацию:

```
  LANGUAGE SQL
| [NOT] DETERMINISTIC
| SQL SECURITY {DEFINER | INVOKER}
| COMMENT 'string'
```

Тело процедуры *`routine_body`* состоит из составного оператора `BEGIN...END`, внутри которого могут располагаться другие операторы, в том числе и другие составные операторы `BEGIN...END`. Оператор имеет следующий синтаксис:

```
[label:] BEGIN
  statements
END [label]
```

Если оператор начинается с необязательной метки *`label`*, в качестве которой может выступать любое уникальное имя, то он может заканчиваться выражением `END label`.

Рассмотрим простейшую хранимую процедуру `my_version()`, определение которой представлено в листинге 12.1.

Листинг 12.1. Создание хранимой процедуры `my_version()`

```
CREATE PROCEDURE my_version ()
BEGIN
  SELECT VERSION();
END
```

Вызвать хранимую процедуру, созданную в листинге 12.1, можно при помощи оператора `CALL` (листинг 12.2).

Листинг 12.2. Вызов хранимой процедуры при помощи ключевого слова `CALL`

```
CALL my_version ();
+---------------------+
| VERSION()           |
+---------------------+
| 5.0.26-community-nt |
+---------------------+
```

Процесс создания и использования хранимой функции немного отличается от процесса создания и использования хранимой процедуры.

Листинг 12.3. Создание хранимой функции `my_version()`

```
CREATE FUNCTION my_version ()
RETURNS TEXT
BEGIN
  RETURN VERSION();
END
```

Как видно из листинга, сразу после объявления имени функции и пустого списка параметров следует ключевое слово `RETURNS`, которое определяет тип возвращаемого значения (`TEXT`). В теле функции обязательно должен присутствовать один или более операторов `RETURN`, которые останавливают работу функции и возвращают значение. Для вызова хранимой функции не требуется специального оператора, как в случае хранимых процедур. Порядок их вызова совпадает с порядком вызова встроенных функций MySQL (листинг 12.4).

Листинг 12.4. Вызов хранимой функции

```
SELECT my_version();
+---------------------+
| my_version()        |
+---------------------+
| 5.0.26-community-nt |
+---------------------+
```

Основная трудность, которая возникает при работе с хранимыми процедурами и функциями, заключается в том, что обязательный символ точки запятой `;` в конце каждого запроса воспринимается консольным клиентом `mysql` как сигнал к отправке запроса на сервер. Для того чтобы избежать этого, при работе с хранимыми процедурами следует переопределить разделитель запросов при помощи параметра `--delimiter=//` консольного клиента `mysql` (листинг 12.5). В этом случае для обозначения окончания ввода вместо точки запятой необходимо будет использовать последовательность `//`.

Листинг 12.5. Переопределение символа окончания запроса

```
mysql -u root --delimiter=//
```

Кроме того, можно менять разделитель в любой момент в консольном клиенте `mysql`. Для этого необходимо воспользоваться командой `DELIMITER`, после которой указать разделитель (листинг 12.6).

Листинг 12.6. Смена разделителя запросов при помощи команды `DELIMITER`

```
DELIMITER //
SELECT VERSION()//
+---------------------+
| VERSION()           |
+---------------------+
| 5.0.26-community-nt |
+---------------------+
DELIMITER ;
SELECT VERSION();
+---------------------+
| VERSION()           |
+---------------------+
| 5.0.26-community-nt |
+---------------------+
```

В теле хранимой процедуры можно использовать многострочный комментарий в стиле языка C, который начинается с последовательности `/*` и заканчивается последовательностью `*/` (листинг 12.7).

Листинг 12.7. Использование комментария в теле функции

```
CREATE PROCEDURE my_version ()
BEGIN
 /* Многострочный комментарий внутри
    функции my_version()
 */
 SELECT VERSION(); /* Вызов единственного оператора */
END
```

12.2. Использование параметров

Хранимые процедуры и функции, представленные в предыдущем разделе, не очень функциональны, т. к. не используя параметры, практически невозможно влиять на состояние функций и процедур во время вызова.

Параметры могут передавать значения внутрь функции и извлекать результаты вычисления. Для этого каждый из параметров снабжается одним из атрибутов `IN`, `OUT` или `INOUT`, описанным в *разд. 12.1*. В листинге 12.8 приводится пример функции, которая присваивает пользовательской переменной `@x` новое значение.

Замечание

Здесь и далее в текущей главе листинги будут сформированы в предположении, что в качестве разделителя используется `//`.

Листинг 12.8. Использование ключевого слова `IN`

```
CREATE PROCEDURE set_x (IN value INT)
BEGIN
  SET @x = value;
END//
CALL set_x(123456)//
SELECT @x//
+--------+
| @x     |
+--------+
| 123456 |
+--------+
```

Как видно из листинга 12.8, через параметр `value` хранимой процедуре передается числовое значение `123456`, которое она присваивает пользовательской переменной `@x`. Использование ключевого слова `IN` не является обязательным — если ни один из атрибутов не указан, СУБД MySQL считает, что параметр объявлен с атрибутом `IN`.

Замечание

В отличие от пользовательской переменной `@x`, которая является глобальной и доступна как внутри хранимой процедуры `set_x()`, так и вне ее, параметры функции являются локальными и доступны для использования только внутри функции.

Атрибут `IN` сообщает СУБД MySQL, что при помощи параметра `value` пользователи передают данные внутрь функции. То есть если хранимая процедура будет изменена в соответствии с листингом 12.9, новое значение параметра `value`, которое присваивается внутри функции, не будет доступно после вызова процедуры.

Листинг 12.9. Переменная `@y` не изменяется внутри процедуры

```
CREATE PROCEDURE set_x (IN value INT)
BEGIN
  SET @x = value;
  SET value = value - 1000;
END//
SET @y = 10000//
CALL set_x(@y)//
SELECT @x, @y//
+-------+-------+
| @x    | @y    |
+-------+-------+
| 10000 | 10000 |
+-------+-------+
```

Замечание

Следует отметить, что имена параметров при объявлении хранимой процедуры и при вызове не обязательно должны совпадать. Внутри хранимой процедуры все локальные переменные используются без символа `@`, в то время как для глобальных пользовательских переменных символ `@` перед именем обязателен.

Хранимая процедура `set_x()` принимает единственный `IN`-параметр `value`, при помощи оператора `SET` значение параметра изменяется внутри функции. Однако, как видно из листинга 12.9, после выполнения хранимой процедуры

значение пользовательской переменной `@y`, переданной функции в качестве параметра, не изменяется. Если требуется, чтобы значение переменной подвергалось изменению, необходимо объявить параметр процедуры с модификатором `OUT` (листинг 12.10).

Листинг 12.10. Использование ключевого слова `OUT`

```
CREATE PROCEDURE set_x (OUT value INT)
BEGIN
  SET @x = value;
  SET value = 1000;
END//
SET @y = 10000//
CALL set_x(@y)//
SELECT @x, @y//
+------+------+
| @x   | @y   |
+------+------+
| NULL | 1000 |
+------+------+
```

Как видно из листинга 12.10, при использовании модификатора `OUT` любые изменения параметра внутри процедуры отражаются на параметре. Передача в качестве значения пользовательской переменной позволяет использовать результат процедуры для дальнейших вычислений. Однако передать значение внутрь функции при помощи `OUT`-параметра уже не получится.

Замечание

Если локальная или пользовательская переменная не инициируется при помощи оператора `SET` или ключевого слова `DEFAULT`, она получает значение `NULL`.

Для того чтобы через параметр можно было передать значение внутрь процедуры и получить значение, которое попадает в параметр в результате вычислений внутри процедуры, его следует объявить с атрибутом `INOUT` (листинг 12.11).

Листинг 12.11. Использование ключевого слова `INOUT`

```
CREATE PROCEDURE set_x (INOUT value INT)
BEGIN
  SET @x = value;
  SET value = value - 1000;
END//
```

```
SET @y = 10000//
CALL set_x(@y)//
SELECT @x, @y//
+-------+------+
| @x    | @y   |
+-------+------+
| 10000 | 9000 |
+-------+------+
CREATE PROCEDURE select_table (INOUT tab TEXT)
BEGIN
  SELECT * FROM tab;
END//
```

Теперь через параметр `value` можно как передавать значения внутрь процедуры, так и извлекать значения, которые получает параметр внутри процедуры. Тем не менее рекомендуется использовать только `IN`- и `OUT`-параметры, не прибегая к комбинированным `INOUT`-параметрам, т. к. это приводит к нечитабельному и непоследовательному коду.

Замечание

Атрибуты `IN`, `OUT` и `INOUT` доступны лишь для хранимой процедуры, в хранимой функции все параметры всегда имеют атрибут `IN`.

12.3. Локальные переменные

До этого локальные переменные в хранимой процедуре или функции объявлялись как параметры, однако это не всегда удобно. Можно объявить независимую от входных параметров локальную переменную при помощи оператора `DECLARE`, который имеет следующий синтаксис:

```
DECLARE var_name[, ...] type [DEFAULT value]
```

Один оператор `DECLARE` позволяет объявить сразу несколько переменных одного типа, причем необязательное слово `DEFAULT` позволяет назначить инициирующее значение (листинг 12.12).

Листинг 12.12. Использование оператора `DECLARE`

```
CREATE PROCEDURE declare_var ()
BEGIN
  DECLARE id, num INT(11) DEFAULT 0;
  DECLARE name, hello, temp TINYTEXT;
END
//
```

В листинге 12.12 объявляются две переменные типа `INT(11)` — `id` и `num`, инициированные значением 0, и три текстовые переменные — `name`, `hello` и `temp`, объявленные без дополнительной инициализации. Инициировать локальные переменные можно и позже при помощи оператора `SET`.

Оператор `DECLARE` может появляться только внутри блока `BEGIN...END`, область видимости объявленной переменной также ограничена этим блоком. Это означает, что в разных блоках `BEGIN...END` могут быть объявлены переменные с одинаковыми именами, и действовать они будут только в рамках данного блока, не пересекаясь с переменными других блоков (листинг 12.13).

Листинг 12.13. Область видимости локальных переменных с одинаковыми именами

```
CREATE PROCEDURE declare_var ()
outer: BEGIN
  DECLARE var TINYTEXT DEFAULT 'внешняя переменная';
  inner: BEGIN
    DECLARE var TINYTEXT DEFAULT 'внутренняя переменная';
    SELECT var;
  END inner;
  SELECT var;
END outer
//
CALL declare_var()//
+-----------------------+
| var                   |
+-----------------------+
| внутренняя переменная |
+-----------------------+
+--------------------+
| var                |
+--------------------+
| внешняя переменная |
+--------------------+
```

В листинге 12.13 переменная `var` объявляется сначала со значением `'внешняя переменная'` во внешнем блоке `BEGIN...END`, после чего во вложенном блоке объявляется вторая переменная `var` со значением `'внутренняя переменная'`, которая экранирует первую переменную.

Однако переменная, объявленная во внешнем блоке `BEGIN...END`, будет доступна во вложенном блоке, если не будет объявлена экранирующая ее переменная (листинг 12.14).

Листинг 12.14. Область видимости локальных переменных, объявленных во внешнем блоке `BEGIN...END`

```
CREATE PROCEDURE one_declare_var ()
BEGIN
  DECLARE var TINYTEXT DEFAULT 'внешняя переменная';
  BEGIN
    SELECT var;
  END;
  SELECT var;
END
//
CALL one_declare_var()//
+--------------------+
| var                |
+--------------------+
| внешняя переменная |
+--------------------+
+--------------------+
| var                |
+--------------------+
| внешняя переменная |
+--------------------+
```

Обратное неверно: переменная, объявленная во вложенном блоке, недоступна во внешнем (листинг 12.15).

Листинг 12.15. Область видимости локальных переменных, объявленных во вложенном блоке `BEGIN...END`

```
CREATE PROCEDURE inner_declare_var ()
BEGIN
  BEGIN
    DECLARE var TINYTEXT DEFAULT 'внутренняя переменная';
    SELECT var;
  END;
  SELECT var;
END
//
CALL inner_declare_var()//
+-----------------------+
| var                   |
+-----------------------+
| внутренняя переменная |
+-----------------------+
ERROR 1054 (42S22): Unknown column 'var' in 'field list'
```

Как видно из листинга 12.15, вызов оператора `SELECT var` во внешнем блоке `BEGIN...END` приводит к ошибке, т. к. время жизни переменной `var` ограничено ее блоком, и после выхода из этого блока ее существование прекращается.

В заключение следует отметить, что не допускается и повторное объявление переменной в рамках одного блока `BEGIN...END` (листинг 12.16). Это приводит к возникновению ошибки 1331: "Повторное объявление переменной".

Листинг 12.16. Повторное объявление переменной `var`

```
CREATE PROCEDURE dbl_declare_var ()
BEGIN
  DECLARE var TINYTEXT DEFAULT 'внешняя переменная';
  DECLARE var TINYTEXT DEFAULT 'внутренняя переменная';
  SELECT var;
END
//
ERROR 1331 (42000): Duplicate variable: var
```

12.4. Присвоение значения локальной переменной

Помимо ключевого слова `DEFAULT`, позволяющего присваивать значение переменной при ее объявлении, допускается присвоение значения переменной по мере работы процедуры или функции. Существуют два способа назначить переменной новое значение:

- оператор `SET`;
- оператор `SELECT...INTO...FROM`.

В листинге 12.17 первый оператор `SET` присваивает переменной `var` значение 100, а второй оператор `SET` увеличивает значение переменной `var` на единицу.

Листинг 12.17. Использование оператора `SET`

```
SET var = 100;
SET var = var + 1;
```

В листинге 12.18 представлен синтаксис оператора `SELECT...INTO...FROM`, который позволяет сохранять результаты `SELECT`-запроса без их немедленного вывода и без использования внешних переменных.

Листинг 12.18. Использование оператора `SELECT...INTO...FROM`

```
SELECT id, data INTO x, y FROM test LIMIT 1;
```

В листинге 12.18 `id` и `data` являются полями таблицы `test`, а `x` и `y` — локальными переменными хранимой процедуры или функции.

12.5. Форматирование временного интервала

Пусть имеется разница между двумя датами, выраженная в секундах. Необходимо создать функцию `FormatSecond()`, которая бы форматировала временной интервал и возвращала его в виде строки:

```
X дней Y часов Z минут W секунд
```

В листинге 12.19 представлено возможное решение задачи. Следует обратить внимание, что при создании хранимой функции параметр `second` не может предваряться атрибутами `IN`, `OUT` и `INOUT`.

Листинг 12.19. Форматирование временного интервала

```
CREATE FUNCTION FormatSecond (second INT)
RETURNS TEXT
BEGIN
  DECLARE day, hour, minute INT;
  SET day = FLOOR(second/86400);
  SET second = second — day*86400;
  SET hour = FLOOR(second/3600);
  SET second = second — hour*3600;
  SET minute = FLOOR(second/60);
  SET second = second — minute*60;
  RETURN CONCAT(day, " дней ",
                hour, " часов ",
                minute, " минут ",
                second, " секунд");
END//
SELECT FormatSecond(123456)//
+-----------------------------------+
| FormatSecond(123456)              |
+-----------------------------------+
| 1 дней 10 часов 17 минут 36 секунд |
+-----------------------------------+
```

Как видно из листинга 12.19, из параметра `second` последовательно вычитаются дни, часы и минуты, которые заносятся в локальные переменные `day`, `hour`, `minute` соответственно. После чего результат собирается в конечную строку при помощи встроенной функции `CONCAT()`.

Пусть имеется таблица `catalogs` (листинг 12.20), содержащая три столбца:

- `id_catalog` — целочисленный столбец, играющий роль первичного ключа таблицы;
- `name` — текстовый столбец, содержащий названия элементов каталога;
- `putdate` — поле типа `DATETIME`, в котором хранится время последнего обновления элемента каталога.

Листинг 12.20. SQL-дамп таблицы `catalogs`

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL,
  name TINYTEXT NOT NULL,
  putdate DATETIME NOT NULL,
  PRIMARY KEY (id_catalog)
);
INSERT INTO catalogs VALUES (3, 'Видеоадаптеры', '2007-01-10 16:48:05'),
                    (4, 'Жесткие диски', '2007-01-05 20:01:58'),
                    (2, 'Материнские платы', '2006-12-28 18:32:41'),
                    (5, 'Оперативная память', '2006-12-20 10:00:13'),
                    (1, 'Процессоры', '2007-01-10 12:47:00');
```

В листинге 12.21 приводится пример запроса, позволяющий узнать, сколько времени прошло с момента последней правки каждого из элементов каталога, и вывести это время в соответствии с форматом функции `FormatSecond()` из листинга 12.19.

Листинг 12.21. Использование функции `FormatSecond()`

```
SELECT name,
       FormatSecond(TIMESTAMPDIFF(SECOND, putdate, NOW())) AS putdate
FROM catalogs;
+--------------------+-----------------------------------+
| name               | putdate                           |
+--------------------+-----------------------------------+
| Видеоадаптеры      | 22 дней 5 часов 59 минут 42 секунд  |
| Жесткие диски      | 27 дней 2 часов 45 минут 49 секунд  |
| Материнские платы  | 35 дней 4 часов 15 минут 6 секунд   |
| Оперативная память | 43 дней 12 часов 47 минут 34 секунд |
| Процессоры         | 22 дней 10 часов 0 минут 47 секунд  |
+--------------------+-----------------------------------+
```

Функция `TIMESTAMPDIFF()` возвращает разницу между двумя датами в единицах измерения, указанных в первом аргументе. Функция доступна, начиная с вер-

сии MySQL 5.0. В более ранних версиях необходимо преобразовать обе даты к формату UNIXSTAMP (количество секунд, прошедших с полуночи 1 января 1970 года) при помощи функции `FROM_UNIXTIME()` и вычесть полученные значения друг из друга.

В листинге 12.22 представлен более сложный пример — вычисляется разница между правками отдельных элементов каталога, и это достигается при помощи самообъединения таблицы `catalogs`. Для удобства таблица сортируется по первичному ключу `id_catalog`, и разница в правке вычисляется для двух соседних позиций.

Листинг 12.22. Использование функции `FormatSecond()`

```
SELECT * FROM catalogs ORDER BY id_catalog;
+------------+--------------------+---------------------+
| id_catalog | name               | putdate             |
+------------+--------------------+---------------------+
|          1 | Процессоры         | 2007-01-10 12:47:00 |
|          2 | Материнские платы  | 2006-12-28 18:32:41 |
|          3 | Видеоадаптеры      | 2007-01-10 16:48:05 |
|          4 | Жесткие диски      | 2007-01-05 20:01:58 |
|          5 | Оперативная память | 2006-12-20 10:00:13 |
+------------+--------------------+---------------------+
SELECT CONCAT(fst.name, " — ", snd.name) AS name,
       FormatSecond(ABS(TIMESTAMPDIFF(SECOND,
                                      fst.putdate,
                                      snd.putdate))) AS putdate
FROM catalogs AS fst, catalogs AS snd
WHERE fst.id_catalog = snd.id_catalog — 1
ORDER BY fst.id_catalog;
+---------------------------------+-------------------------------------+
| name                            | putdate                             |
+---------------------------------+-------------------------------------+
| Процессоры — Материнские платы  | 12 дней 18 часов 14 минут 19 секунд |
| Материнские платы — Видеоадаптеры| 12 дней 22 часов 15 минут 24 секунд|
| Видеоадаптеры — Жесткие диски   | 4 дней 20 часов 46 минут 7 секунд   |
| Жесткие диски — Оперативная память|16 дней 10 часов 1 минут 45 секунд |
+---------------------------------+-------------------------------------+
```

Ряд значений, которые возвращает функция `TIMESTAMPDIFF()`, оказываются отрицательными, поэтому при помощи функции `ABS()` возвращается абсолютное значение результата.

12.6. Группа характеристик хранимых процедур

Синтаксис хранимых процедур допускает использование следующих характеристик *characteristic* в определении CREATE PROCEDURE и CREATE FUNCTION *(см. разд. 12.1)*:

```
LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY {DEFINER | INVOKER}
  | COMMENT 'string'
```

Данные ключевые слова описывают характеристики хранимых процедур и функций. Они размещаются после списка параметров, но до начала тела хранимой процедуры. Характеристика LANGUAGE SQL пока не имеет особого смысла и сообщает, что хранимая процедура написана на языке SQL (листинг 12.23).

Листинг 12.23. Использование ключевого слова LANGUAGE SQL

```
CREATE PROCEDURE characteristics ()
LANGUAGE SQL
BEGIN
  SELECT RAND();
END
//
```

В следующих версиях MySQL планируется предоставить возможность создания хранимых процедур с использованием языка, отличного от SQL. Скорее всего, одним из первых поддерживаемых языков станет PHP, т. к. базовый механизм PHP невелик по размерам, безопасен в отношении потоков и легко встраивается. Рекомендуется использовать ключевое слово LANGUAGE SQL уже сейчас для обеспечения совместимости с будущими версиями MySQL и другими СУБД.

Ключевое слово DETERMINISTIC позволяет сообщить оптимизатору, что процедура всегда возвращает один и тот же результат для одних и тех же входных параметров, в противном случае следует использовать ключевое слово NOT DETERMINISTIC (листинг 12.24).

Листинг 12.24. Использование ключевого слова NOT DETERMINISTIC

```
CREATE PROCEDURE characteristics ()
LANGUAGE SQL
```

```
NOT DETERMINISTIC
BEGIN
  SELECT RAND();
END
//
```

Замечание

В настоящий момент ключевое слово DETERMINISTIC распознается, но еще не используется оптимизатором MySQL.

Хранимая процедура или функция может быть снабжена дополнительными характеристиками, содержащими информацию о природе данных, используемых внутри процедуры:

- CONTAINS SQL — ключевое слово сообщает, что хранимая процедура или функция не содержит операторов, читающих или записывающих данные (если не указаны никакие другие ключевые слова, этот режим используется по умолчанию);
- NO SQL — ключевое слово сообщает, что хранимая процедура или функция не содержит SQL-операторов;
- READS SQL DATA — ключевое слово сообщает, что хранимая процедура или функция содержит только операторы, осуществляющие чтение данных, и не содержит записывающих операторов;
- MODIFIES SQL DATA — ключевое слово сообщает, что хранимая процедура или функция содержит операторы, которые могут записывать информацию.

Замечание

Ключевые слова CONTAINS SQL, NO SQL, READS SQL DATA и MODIFIES SQL DATA являются информационными и не используются сервером MySQL для того, чтобы разрешать или запрещать хранимым процедурам и функциям выполнение тех или иных операций.

Ключевое слово SQL SECURITY может быть записано в двух формах: SQL SECURITY DEFINER и SQL SECURITY INVOKER. Если применяется форма SQL SECURITY DEFINER, то хранимая процедура вызывается с привилегиями пользователя, создавшего ее. При использовании SQL SECURITY INVOKER процедура вызывается с привилегиями пользователя, вызывающего процедуру оператором CALL (листинг 12.25).

Листинг 12.25. Использование ключевого слова SQL SECURITY

```
CREATE PROCEDURE characteristics ()
LANGUAGE SQL
```

```
NOT DETERMINISTIC
SQL SECURITY INVOKER
BEGIN
  SELECT RAND();
END
//
```

Замечание

Если ключевое слово SQL SECURITY не указано, по умолчанию устанавливается режим SQL SECURITY DEFINER — хранимая процедура выполняется с привилегиями создавшего ее пользователя.

Для выполнения хранимой процедуры и ее создатель, и ее пользователь должны иметь доступ к базе данных, в которой сохранена процедура. Кроме того, они оба должны иметь привилегию EXECUTE, независимо от того, в какой форме использовано ключевое слово SQL SECURITY.

Ключевое слово COMMENT (листинг 12.26) позволяет снабжать хранимую процедуру кратким описанием, которое отображается операторами SHOW CREATE PROCEDURE и SHOW CREATE FUNCTION.

Замечание

Ключевое слово COMMENT является расширением MySQL и может не поддерживаться другими СУБД.

Листинг 12.26. Использование ключевого слова COMMENT

```
CREATE PROCEDURE characteristics ()
LANGUAGE SQL
NOT DETERMINISTIC
SQL SECURITY INVOKER
COMMENT 'Функция возвращает случайное значение'
BEGIN
  SELECT RAND();
END
//
mysql> SHOW CREATE PROCEDURE characteristics\G//
*************************** 1. row ***************************
       Procedure: characteristics
        sql_mode:
Create Procedure: CREATE PROCEDURE `shop`.`characteristics`()
    SQL SECURITY INVOKER
    COMMENT 'Функция возвращает случайное значение'
BEGIN
  SELECT RAND();
END
```

12.7. Операторы ветвления

Хранимые процедуры — это не просто удобные контейнеры для группы запросов, они позволяют реализовать достаточно сложную логику, используя операторы ветвления.

Замечание

Вне хранимых процедур описываемые в данном разделе операторы применять нельзя.

12.7.1. Оператор *IF...THEN...ELSE*

Оператор `IF` позволяет реализовать ветвление программы по условию и имеет следующий синтаксис:

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

Логическое выражение *`search_condition`* может принимать два значения: 0 (ложь) и значение, отличное от нуля (истина). Если логическое выражение истинно, то оператор *`statement_list`* после ключевого слова `THEN` выполняется, иначе выполняется список операторов в блоке `ELSE` (если блок `ELSE` имеется). В качестве *`statement_list`* может выступать составной оператор `BEGIN...END`.

Замечание

Следует отметить, что в СУБД MySQL, кроме оператора `IF`, представленного в этом разделе, существует функция `IF()`, которая реализует логику С-оператора `X ? Y : Z` и описывается в *главе 6*.

Рассмотрим простейший пример использования оператора `IF`. Хранимая процедура `format_now()`, представленная в листинге 12.27, выводит текущую дату в формате `"DD.MM.YYYY"`, если аргумент `flag` принимает значение 0, и текущее время в формате `"hh:ii:ss"`, если аргумент `flag` принимает значение 1.

Замечание

Для создания логических выражений можно использовать все операторы сравнения (=, >, >=, <>, <, <=). Кроме того, логические выражения можно комбинировать при помощи операторов `&&` (И) и `||` (ИЛИ).

Листинг 12.27. Использование оператора `IF`

```
CREATE PROCEDURE format_now (flag INT)
LANGUAGE SQL
```

```
BEGIN
  IF(flag = 0) THEN
    SELECT DATE_FORMAT(NOW(), "%d.%m.%Y") AS format_now;
  END IF;
  IF(flag = 1) THEN
    SELECT DATE_FORMAT(NOW(), "%H:%i:%s") AS format_now;
  END IF;
END
//
CALL format_now(0)//
+------------+
| format_now |
+------------+
| 01.02.2007 |
+------------+
CALL format_now(1)//
+------------+
| format_now |
+------------+
| 23:51:28   |
+------------+
```

Оператор `IF` может быть снабжен дополнительным блоком `ELSE`, после которого выполняются операторы, если условие оказалось ложным. Функцию `format_now()` можно переписать так, как это показано в листинге 12.28.

Замечание

После ключевого слова `ELSE` ставить точку с запятой не нужно, т. к. этим единый оператор `IF` разбивается на части — точку с запятой ставят после ключевого слова `END IF`.

Листинг 12.28. Использование оператора `IF...ELSE`

```
CREATE PROCEDURE format_now (flag INT)
LANGUAGE SQL
BEGIN
  IF(flag) THEN
    SELECT DATE_FORMAT(NOW(), "%H:%i:%s") AS format_now;
  ELSE
    SELECT DATE_FORMAT(NOW(), "%d.%m.%Y") AS format_now;
  END IF;
END
//
```

С точки зрения функциональности, хранимые процедуры из листингов 12.27 и 12.28 абсолютно одинаковы, но в последнем случае понадобился лишь один оператор `IF`, который принимает в качестве логического выражения аргумент `flag`. Если `flag` равен 1, что является истиной, выполняется первый оператор, выводящий текущее время, если аргумент `flag` равен 0, что является ложью, выполняется запрос в блоке `ELSE`.

Оператор `IF` позволяет выбрать и большее число альтернатив. Процедуру `format_now()` можно изменить таким образом, чтобы она выводила количество секунд, прошедших с полуночи 1 января 1970 года (листинг 12.29). Таким образом, аргумент `flag` может принимать следующие значения:

- 0 — дата в формате `"DD.MM.YYYY"`;
- 1 — время в формате `"%H:%i:%s"`;
- 2 и выше — время в формате UNIXSTAMP.

Листинг 12.29. Использование оператора `IF...ELSEIF...ELSE`

```
CREATE PROCEDURE format_now (flag INT)
LANGUAGE SQL
BEGIN
  IF(flag = 0) THEN
    SELECT DATE_FORMAT(NOW(), "%d.%m.%Y") AS format_now;
  ELSEIF (flag = 1) THEN
    SELECT DATE_FORMAT(NOW(), "%H:%i:%s") AS format_now;
  ELSE
    SELECT UNIX_TIMESTAMP(NOW()) AS format_now;
  END IF;
END
//
CALL format_now(2)//
+------------+
| format_now |
+------------+
| 1170370484 |
+------------+
```

Оператор `IF` в листинге 12.29 проверяет, не равен ли параметр `flag` нулю, если это так, срабатывает первый запрос, и процедура выходит из `IF`. Если параметр `flag` не равен нулю, проверка перемещается к блоку `ELSEIF`, где происходит сравнение параметра `flag` с единицей. Равенство `flag` единице приводит к выполнению второго оператора `SELECT`. Если оператор `flag` принимает любое другое значение, отличное от 0 и 1, выполняется третий запрос. Тре-

тий запрос будет выполнен, даже если flag является отрицательным значением.

Число блоков ELSEIF не ограничено — можно использовать любое их количество. Однако злоупотреблять блоками ELSEIF не рекомендуется, т. к. большое число блоков ELSEIF снижает читабельность кода.

Если в блоках IF, ELSEIF и ELSE используются два или более операторов, для наглядности можно прибегать к составному оператору BEGIN...END.

12.7.2. Оператор *CASE*

Оператор CASE позволяет осуществить множественный выбор и имеет две формы. Синтаксис первой формы оператора выглядит следующим образом:

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Синтаксис второй формы:

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Замечание

Синтаксис оператора CASE внутри хранимой процедуры немного отличается от синтаксиса SQL-выражения CASE. Оператор CASE не может содержать конструкцию ELSE NULL, и его выполнение завершается с помощью выражения END CASE, а не END.

В первой форме оператор CASE сравнивает выражение *case_value* с *when_value*. Как только соответствие найдено, выполняется необходимый оператор *statement_list*. Если ни одно соответствие не найдено, выполняется оператор *statement_list*, размещенный после ключевого слова ELSE (если оно, конечно, присутствует).

Перепишем хранимую процедуру farmat_now() из листинга 12.29 при помощи оператора CASE (листинг 12.30).

Листинг 12.30. Использование оператора CASE (первая форма)

```
CREATE PROCEDURE format_now (flag INT)
LANGUAGE SQL
```

```
BEGIN
  CASE flag
    WHEN 0 THEN
      SELECT DATE_FORMAT(NOW(), "%d.%m.%Y") AS format_now;
    WHEN 1 THEN
      SELECT DATE_FORMAT(NOW(), "%H:%i:%s") AS format_now;
    WHEN 2 THEN
      SELECT UNIX_TIMESTAMP(NOW()) AS format_now;
    ELSE
      SELECT 'Ошибка в параметре flag';
  END CASE;
END
//
CALL format_now (5)//
+-------------------------+
| Ошибка в параметре flag |
+-------------------------+
| Ошибка в параметре flag |
+-------------------------+
```

В листинге 12.30 представлена ситуация, когда в качестве параметра `flag` передано ошибочное значение, обработана специально в блоке `ELSE`. Подход с применением оператора `IF` это также допускает, но в операторе `CASE` ключевое слово `ELSE` лучше выделяется на фоне последовательности ключевых слов `WHEN` по сравнению с ключевыми словами `ELSEIF` в операторе `IF`.

Вторая форма оператора `CASE` позволяет осуществлять сравнение непосредственно в конструкции `WHEN` — как только будет найдено первое истинное значение, выполняется оператор *`statement_list`*, и процедура выходит из оператора `CASE`. В листинге 12.31 представлена процедура `format_now()`, реализованная с применением второй формы оператора `CASE`.

Листинг 12.31. Использование оператора `CASE` (вторая форма)

```
CREATE PROCEDURE format_now (flag INT)
LANGUAGE SQL
BEGIN
  CASE
    WHEN flag = 0 THEN
      SELECT DATE_FORMAT(NOW(), "%d.%m.%Y") AS format_now;
    WHEN flag = 1 THEN
      SELECT DATE_FORMAT(NOW(), "%H:%i:%s") AS format_now;
    WHEN flag = 2 THEN
      SELECT UNIX_TIMESTAMP(NOW()) AS format_now;
```

```
    ELSE
      SELECT 'Ошибка в параметре flag';
  END CASE;
END
//
```

Если в одном блоке WHEN необходимо выполнить несколько запросов, следует использовать блок BEGIN...END.

12.7.3. Проверка правильности ввода

Пусть необходимо выводить название элементов каталога из таблицы catalogs, представленной в листинге 12.20. Вывод названия должен осуществляться по первичному ключу id_catalog. При этом если в таблице отсутствует запись с переданным номером, необходимо вывести соответствующее сообщение. В листинге 12.32 представлено возможное решение.

Листинг 12.32. Хранимая процедура, выводящая запись по первичному ключу

```
CREATE PROCEDURE show_catalog (id INT)
LANGUAGE SQL
BEGIN
   DECLARE name_catalog TINYTEXT;
   SELECT name INTO name_catalog FROM catalogs WHERE id_catalog = id;
   IF(name_catalog IS NOT NULL) THEN
     SELECT name_catalog FROM DUAL;
   ELSE
     SELECT "Запрашиваемый элемент каталога отсутствует" FROM DUAL;
   END IF;
END
//
CALL show_catalog (0)//
+-------------------------------------------+
| Запрашиваемый элемент каталога отсутствует |
+-------------------------------------------+
| Запрашиваемый элемент каталога отсутствует |
+-------------------------------------------+
CALL show_catalog (1)//
+--------------+
| name_catalog |
+--------------+
| Процессоры   |
+--------------+
```

Если в таблице catalogs отсутствует запись, соответствующая передаваемому параметру id, оператор SELECT...INTO...FROM не заполняет локальную переменную name_catalog и ее значение остается не определенным (т. е. равное NULL). Поэтому далее, если значение переменной равно NULL, выводится надпись об отсутствии записи в таблице, если переменная name_catalog не равна NULL — выводится название элемента каталога.

12.7.4. Вывод приветствия в зависимости от времени суток

Создадим хранимую функцию hello(), которая будет возвращать приветствие, в зависимости от текущего времени суток. С 6:00 до 12:00 функция должна возвращать фразу "Доброе утро", с 12:00 до 18:00 функция должна возвращать фразу "Добрый день", с 18:00 до 00:00 — "Добрый вечер", с 00:00 до 6:00 — "Доброй ночи". В листинге 12.33 представлено возможное решение.

Листинг 12.33. Вывод приветствия в зависимости от времени суток

```
CREATE FUNCTION hello ()
RETURNS TINYTEXT
BEGIN
  DECLARE hour INT;
  SET hour = HOUR(NOW());
  CASE
    WHEN hour BETWEEN 0 AND 5 THEN
      RETURN "Доброй ночи";
    WHEN hour BETWEEN 6 AND 11 THEN
      RETURN "Доброе утро";
    WHEN hour BETWEEN 12 AND 17 THEN
      RETURN "Добрый день";
    WHEN hour BETWEEN 18 AND 23 THEN
      RETURN "Добрый вечер";
  END CASE;
END
//
SELECT NOW(), hello ()//
+---------------------+-------------+
| NOW()               | hello ()    |
+---------------------+-------------+
| 2007-02-02 02:42:07 | Доброй ночи |
+---------------------+-------------+
```

12.8. Циклы

Циклы являются важнейшей конструкцией, без которой хранимые процедуры и функции не имели бы достаточно функциональности. Таблицы, как правило, имеют множество записей, поэтому циклическая обработка данных встречается в SQL-программировании достаточно часто.

Замечание

Вне хранимых процедур описываемые в данном разделе операторы применять нельзя.

12.8.1. Оператор *WHILE*

Оператор `WHILE` выполняет цикл и имеет следующий синтаксис:

```
[label:] WHILE search_condition DO
    statement_list
END WHILE [label]
```

Цикл `WHILE` выполняет операторы *`statement_list`* до тех пор, пока условие *`search_condition`* истинно. При каждой итерации условие *`search_condition`* проверяется, и если при очередной проверке оно будет ложным (0), цикл завершит свое выполнение. Это означает, что если условие *`search_condition`* ложно с самого начала, цикл не выполнит ни одной итерации.

Если в цикле требуется выполнить более одного оператора, не обязательно заключать их в блок `BEGIN...END`, т. к. эту функцию выполняет сам оператор `WHILE`.

Выведем 3 раза текущую дату при помощи цикла `WHILE` (листинг 12.34).

Листинг 12.34. Использование оператора `WHILE` (пример 1)

```
CREATE PROCEDURE NOW3 ()
LANGUAGE SQL
BEGIN
  DECLARE i INT DEFAULT 3;
  WHILE i > 0 DO
    SELECT NOW();
    SET i = i — 1;
  END WHILE;
END
//
CALL NOW3()//
```

```
+---------------------+
| NOW()               |
+---------------------+
| 2005-07-18 12:36:59 |
+---------------------+
+---------------------+
| NOW()               |
+---------------------+
| 2005-07-18 12:36:59 |
+---------------------+
+---------------------+
| NOW()               |
+---------------------+
| 2005-07-18 12:36:59 |
+---------------------+
```

Первый оператор в цикле `WHILE` выводит текущую дату, а второй вычитает из локальной переменной `i` единицу. Если единицу не вычитать, то образуется бесконечный цикл, из которого процедура никогда не выйдет, а будет бесполезно нагружать сервер, пока сеанс с ним не прекратится. Следует очень внимательно проектировать циклы, чтобы предотвратить бесконечные циклы.

В листинге 12.35 представлен код хранимой процедуры, которая выводит текущую дату `num` раз, где `num` — параметр, задаваемый пользователем.

Листинг 12.35. Использование оператора `WHILE` (пример 2)

```
CREATE PROCEDURE NOWN (IN num INT)
LANGUAGE SQL
BEGIN
  DECLARE i INT DEFAULT 0;
  IF (num > 0) THEN
    wet : WHILE i < num DO
      SELECT NOW();
      SET i = i + 1;
    END WHILE wet;
  ELSE
    SELECT 'Ошибочное значение параметра';
  END IF;
END
//
CALL NOWN(2)//
+---------------------+
| NOW()               |
+---------------------+
| 2005-07-18 12:51:27 |
+---------------------+
```

```
+---------------------+
| NOW()               |
+---------------------+
| 2005-07-18 12:51:27 |
+---------------------+
```

Как видно из листинга 12.35, цикл `WHILE` снабжен меткой `wet`. Метка в цикле предназначена не только для того, чтобы облегчить чтение кода при очень длинных циклах, она позволяет осуществлять досрочный выход из цикла.

12.8.2. Досрочный выход из цикла

Для досрочного выхода из цикла предназначен оператор `LEAVE`, который имеет следующий синтаксис:

```
LEAVE label
```

Оператор `LEAVE` прекращает выполнение блока, помеченного меткой *`label`*.

Замечание

Оператор `LEAVE` эквивалентен оператору `break` в С-подобных языках программирования.

Хранимая процедура `NOWN()` из листинга 12.35 обладает недостатком — если задать очень большое значение аргумента `num`, можно создать псевдобесконечный цикл, который позволит злоумышленнику загрузить сервер бесполезной работой. Для предотвращения такой ситуации можно воспользоваться оператором `LEAVE`, который прекратит выполнение цикла по достижении критического числа итераций. В листинге 12.36 приводится пример хранимой процедуры, где число итераций ограничено двумя.

Листинг 12.36. Досрочный выход из цикла `WHILE`

```
CREATE PROCEDURE NOWN (IN num INT)
LANGUAGE SQL
BEGIN
  DECLARE i INT DEFAULT 0;
  IF (num > 0) THEN
    wet : WHILE i < num DO
      IF i > 2 THEN LEAVE wet;
      END IF;
      SELECT NOW();
      SET i = i + 1;
    END WHILE wet;
```

```
  ELSE
    SELECT 'Ошибочное значение параметра';
  END IF;
END
//
CALL NOWN(10)//
+---------------------+
| NOW()               |
+---------------------+
| 2005-07-18 12:51:27 |
+---------------------+
+---------------------+
| NOW()               |
+---------------------+
| 2005-07-18 12:51:27 |
+---------------------+
```

Условие `IF i > 2 THEN LEAVE wet;` проверяет, не превысило ли значение счетчика `i` числа 2, и если это так, происходит прекращение цикла `WHILE`. Использование меток позволяет точно указать, какой цикл необходимо прервать. Если имеется вложенный цикл, можно явно указать, какой из двух циклов требуется прервать (листинг 12.37).

Листинг 12.37. Досрочный выход из вложенного цикла (пример 1)

```
first : WHILE i < num DO
  second : WHILE j < num DO
    IF i > 2 && j > 2 THEN LEAVE first;
    END IF;
    SELECT NOW();
    SET j = j + 1;
  END WHILE second;
  SET i = i + 1;
END WHILE first;
```

При достижении условия `i > 2 && j > 2` оператор `LEAVE` прервет не вложенный цикл `second`, а внешний цикл `first`, т. к. метка внешнего цикла явно указана после оператора.

При использовании досрочного выхода `LEAVE` можно даже создавать бесконечные циклы, т. к. рано или поздно внешний цикл будет завершен (листинг 12.38).

Листинг 12.38. Досрочный выход из вложенного цикла (пример 2)

```
first : WHILE 1 DO
  second : WHILE 1 DO
```

```
    IF i > 2 && j > 2 THEN LEAVE first;
    END IF;
    SELECT NOW();
    SET j = j + 1;
  END WHILE second;
  SET i = i + 1;
END WHILE first;
```

Значение 1 в условии цикла всегда будет истинным, и цикл не прекратится до тех пор, пока его не прервет оператор `LEAVE`.

Еще одним оператором, выполняющим досрочное прекращение цикла, является оператор `ITERATE`, который имеет следующий синтаксис:

`ITERATE` *`label`*

В отличие от оператора `LEAVE`, оператор `ITERATE` не прекращает выполнение цикла, он лишь выполняет досрочное прекращение текущей итерации.

Замечание

Оператор `ITERATE` эквивалентен оператору `continue` в С-подобных языках программирования.

Рассмотрим программу, которая в цикле формирует бинарную последовательность, добавляя к строке две единицы на четных итерациях и две единицы и два нуля на нечетных (листинг 12.39).

Листинг 12.39. Использование оператора `ITERATE`

```
CREATE PROCEDURE binstring (IN num INT)
LANGUAGE SQL
BEGIN
  DECLARE i INT DEFAULT 0;
  DECLARE bin TINYTEXT DEFAULT '';
  IF (num > 0) THEN
    wet : WHILE i < num DO
      SET i = i + 1;
      SET bin = CONCAT(bin, '11');
      IF !(i/2 — CEILING(i/2)) THEN ITERATE wet;
      END IF;
      SET bin = CONCAT(bin, '00');
    END WHILE wet;
    SELECT bin;
  ELSE
    SELECT 'Ошибочное значение параметра';
  END IF;
END
```

```
//
CALL binstring(10)//
+-------------------------------+
| bin                           |
+-------------------------------+
| 1100111100111100111100111110011 |
+-------------------------------+
```

Бинарная последовательность хранится во временной строке `bin`, которая обязательно должна быть инициирована пустой строкой:

```
DECLARE bin TINYTEXT DEFAULT '';
```

Если инициализация не проведена, переменная получит значение `NULL`, и все операции с этой переменной также будут приводить к `NULL`. На каждой итерации переменной `bin` при помощи функции `CONCAT()` прибавляется последовательность `'11'`. Если индекс `i` является четным (0, 2, 4, 6, 8), текущая итерация досрочно прекращается при помощи ключевого слова `ITERATE`, если индекс является нечетным (1, 3, 5, 7, 9), то итерация выполняется до конца, т. е. к временной строке `bin` добавляется еще и последовательность `'00'`. На четность индекс `i` проверяется при помощи строки

```
i/2 — CEILING(i/2)
```

Если индекс `i` делится на 2 без остатка, это выражение вернет 0 (ложь), если число является нечетным, то выражение вернет 0,5 (истина).

Замечание

Нужно следить, чтобы оператор `SET`, увеличивающий значение счетчика `i` на единицу, на каждой итерации был расположен до оператора `ITERATE`, иначе это приведет к созданию бесконечного цикла — значение счетчика будет оставаться нечетным и увеличиваться не станет, т. к. оператор `ITERATE` будет прекращать выполнение итерации цикла досрочно.

12.8.3. Оператор *REPEAT*

Оператор `REPEAT`, так же как и оператор `WHILE`, реализует цикл:

```
[label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [label]
```

Отличительной особенностью данного цикла является тот факт, что условие цикла *search_condition* проверяется не в начале, как в цикле `WHILE`, а в конце оператора (ключевое слово `UNTIL`). Таким образом, цикл выполняет, по край-

ней мере, одну итерацию независимо от условия. Следует отметить, что цикл REPEAT выполняется, пока условие *search_condition* ложно.

Оператор REPEAT может быть снабжен необязательной меткой *label*, по которой легко осуществлять досрочный выход из цикла при помощи операторов LEAVE и ITERATE, рассмотренных в предыдущем разделе.

В листинге 12.40 представлена хранимая процедура binrand(), которая генерирует и выводит случайную бинарную последовательность из 20 символов. Для формирования бинарной последовательности применяется цикл REPEAT.

Листинг 12.40. Использование цикла REPEAT

```
CREATE PROCEDURE binrand ()
LANGUAGE SQL
BEGIN
  DECLARE i INT DEFAULT 0;
  DECLARE bin TINYTEXT DEFAULT '';
  REPEAT
      SET i = i + 1;
      IF RAND() > 0.5 THEN SET bin = CONCAT(bin, '1');
      ELSE SET bin = CONCAT(bin, '0');
      END IF;
  UNTIL i >= 20
  END REPEAT;
  SELECT bin;
END
//
CALL binrand()//
+-----------------------+
| bin                   |
+-----------------------+
| 10101000111100011111110 |
+-----------------------+
```

12.8.4. Оператор *LOOP*

Оператор LOOP предназначен для реализации циклов и имеет следующий синтаксис:

```
[label:] LOOP
    statement_list
END LOOP [label]
```

Цикл LOOP, в отличие от операторов WHILE и REPEAT, не имеет условий выхода. Поэтому данный вид цикла должен обязательно иметь в своем составе оператор LEAVE.

Хранимую процедуру binrand(), приведенную в листинге 12.40, можно переписать с использованием цикла LOOP, как это сделано в листинге 12.41.

Листинг 12.41. Использование оператора LOOP

```
CREATE PROCEDURE binrand ()
LANGUAGE SQL
BEGIN
  DECLARE i INT DEFAULT 0;
  DECLARE bin TINYTEXT DEFAULT '';
  wet : LOOP
      SET i = i + 1;
      IF RAND() > 0.5 THEN SET bin = CONCAT(bin, '1');
      ELSE SET bin = CONCAT(bin, '0');
      END IF;
      IF i >= 20 THEN LEAVE wet;
      END IF;
  END LOOP wet;
  SELECT bin;
END
//
CALL binrand()//
+----------------------+
| bin                  |
+----------------------+
| 01000111011110001110 |
+----------------------+
```

12.8.5. Отсутствующие идентификаторы

Таблица tbl имеет первичный ключ id, снабженный атрибутом AUTO_INCREMENT. По мере работы с таблицей некоторые записи удаляются таким образом, что образуются пропуски (листинг 12.42).

Листинг 12.42. Таблица tbl

```
CREATE TABLE tbl (
  id INT(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY  (id)
);
INSERT INTO tbl VALUES (3), (5), (10), (12), (15), (19), (20), (21),
                       (23), (26), (29), (30), (35), (37), (41), (44),
                       (50), (54), (67), (72), (86), (94), (99), (100);
```

Пусть стоит задача создать список отсутствующих идентификаторов. Решить эту задачу можно при помощи хранимой процедуры, представленной в листинге 12.43.

Листинг 12.43. Получение списка отсутствующих идентификаторов

```
CREATE PROCEDURE empty_value (OUT result TEXT)
LANGUAGE SQL
BEGIN
  DECLARE i INT DEFAULT 0;
  DECLARE x INT DEFAULT 0;
  SET result = "";
  WHILE i < 100 DO
    SET x = NULL;
    SELECT i INTO x FROM DUAL WHERE i IN (SELECT id FROM tbl);
    IF(x IS NULL) THEN
      SET result = CONCAT(result," ",i);
    END IF;
    SET i = i + 1;
  END WHILE;
END//
CALL empty_value(@id)//
SELECT @id\G
*************************** 1. row ***************************
@id:  0 1 2 4 6 7 8 9 11 13 14 16 17 18 22 24 25 27 28 31 32 33 34 36 38 39 40
42 43 45 46 47 48 49 51 52 53 55 56 57 58 59 60 61 62 63 64 65 66 68 69 70 71
73 74 75 76 77 78 79 80 81 82 83 84 85 87 88 89 90 91 92 93 95 96 97 98
```

12.9. Получение списка созданных хранимых процедур и функций

Просмотреть список уже созданных хранимых процедур можно при помощи оператора `SHOW PROCEDURE STATUS`, который имеет следующий синтаксис:

```
SHOW PROCEDURE STATUS [LIKE 'pattern'];
```

Оператор возвращает список хранимых процедур, который содержит перечень хранимых функций. При использовании ключевого слова `LIKE` можно вывести информацию только о тех процедурах, имена которых удовлетворяют шаблону *`pattern`* (листинг 12.44).

Листинг 12.44. Использование оператора `SHOW PROCEDURE STATUS`

```
SHOW PROCEDURE STATUS LIKE 'bin%'\G;
*************************** 1. row ***************************
           Db: shop
         Name: binrand
         Type: PROCEDURE
```

```
      Definer: root@localhost
     Modified: 2005-07-18 23:43:23
      Created: 2005-07-18 23:43:23
Security_type: DEFINER
      Comment:
*************************** 2. row ***************************
           Db: shop
         Name: binstring
         Type: PROCEDURE
      Definer: root@localhost
     Modified: 2005-07-18 14:09:55
      Created: 2005-07-18 14:09:55
Security_type: DEFINER
      Comment:
```

Как видно из листинга 12.44, оператор возвращает результирующую таблицу, в которой каждая строка соответствует одной хранимой процедуре. При этом таблица содержит 8 столбцов:

- `Db` — имя базы данных, в которой сохранена процедура;
- `Name` — имя процедуры;
- `Type` — тип хранимой процедуры, принимает значение `PROCEDURE` для оператора хранимой процедуры и `FUNCTION` для хранимой функции (список хранимых функций возвращается оператором `SHOW PROCEDURE STATUS`, который рассматривается далее);
- `Definer` — учетная запись, от имени которой была создана хранимая процедура;
- `Modified` — дата последней модификации хранимой процедуры;
- `Created` — дата создания хранимой процедуры;
- `Security_type` — режим выполнения хранимой процедуры. Если это поле принимает значение `DEFINER`, то хранимая процедура выполняется с правами доступа пользователя, создавшего данную процедуру. Если поле `Security_type` принимает значение `INVOKER`, то хранимая процедура выполняется с правами доступа пользователя, вызывающего процедуру при помощи оператора `CALL`;
- `Comment` — комментарий к хранимой процедуре.

Для просмотра списка хранимых функций предназначен оператор `SHOW FUNCTION STATUS`. Оператор имеет следующий синтаксис:

```
SHOW FUNCTION STATUS [LIKE 'pattern'];
```

Оператор выводит список хранимых функций, который не включает в свой состав хранимые процедуры (листинг 12.45). При использовании ключевого

слова LIKE можно вывести информацию только о тех функциях, имена которых удовлетворяют шаблону *pattern*.

Листинг 12.45. Использование оператора SHOW FUNCTION STATUS

```
SHOW FUNCTION STATUS LIKE 'count%'\G;
*************************** 1. row ***************************
           Db: shop
         Name: count_product
         Type: FUNCTION
      Definer: root@localhost
     Modified: 2005-07-17 01:26:41
      Created: 2005-07-17 01:26:41
Security_type: DEFINER
      Comment:
*************************** 2. row ***************************
           Db: shop
         Name: count_product_in_catalog
         Type: FUNCTION
      Definer: root@localhost
     Modified: 2005-07-17 01:26:40
      Created: 2005-07-17 01:26:40
Security_type: DEFINER
      Comment:
```

Формат вывода оператора SHOW FUNCTION STATUS совпадает с форматом оператора SHOW PROCEDURE STATUS, рассмотренного ранее.

Помимо представленных выше операторов SHOW, существует еще один способ извлечь информацию о хранимых процедурах — извлечь строки таблицы proc системной базы данных mysql, куда помещаются все хранимые процедуры. В листинге 12.46 приводится пример SELECT-запроса, извлекающего запись, соответствующую хранимой функции count_product_in_catalog().

Листинг 12.46. Извлечение информации из таблицы mysql.proc

```
SELECT * FROM mysql.proc
WHERE name = 'count_product_in_catalog'\G;
*************************** 1. row ***************************
              db: shop
            name: count_product_in_catalog
            type: FUNCTION
   specific_name: count_product_in_catalog
        language: SQL
```

```
 sql_data_access: CONTAINS_SQL
is_deterministic: NO
   security_type: DEFINER
      param_list: id INT
         returns: int(11)
            body: BEGIN
  DECLARE total INT;
  SELECT SUM(count) INTO total FROM products
  WHERE id_catalog = id LIMIT 1;
  RETURN total;
END
         definer: root@localhost
         created: 2005-07-17 01:26:40
        modified: 2005-07-17 01:26:40
        sql_mode:
         comment:
```

Как видно из листинга 12.46, результирующая таблица имеет 16 полей:

- db — имя базы данных, в которую сохранена процедура;
- name — имя процедуры;
- type — тип хранимой процедуры, может принимать два значения: PROCEDURE или FUNCTION, для хранимой процедуры или функции соответственно;
- specific_name — имя процедуры;
- language — язык, на котором написана хранимая процедура, в настоящий момент принимает единственное значение — SQL;
- sql_data_access — поле показывает, насколько зависит хранимая процедура от данных, и может принимать следующие значения: CONTAINS_SQL, NO_SQL, READS_SQL_DATA, MODIFIES_SQL_DATA, если процедура, соответственно, использует константы, не использует SQL, читает данные или модифицирует их;
- is_deterministic — данное поле определяет, является ли хранимая процедура детерминированной (YES) или нет (NO). На состояние данного поля оказывает влияние характеристика DETERMINISTIC;
- security_type — режим выполнения хранимой процедуры. Если это поле принимает значение DEFINER, то хранимая процедура выполняется с правами доступа создавшего ее пользователя. Если поле security_type принимает значение INVOKER, то хранимая процедура выполняется с правами доступа пользователя, вызывающего процедуру при помощи оператора CALL;
- param_list — список параметров хранимой процедуры;

- `returns` — тип результата, возвращаемого хранимой функцией, для хранимых процедур поле принимает пустую строку;
- `body` — тело хранимой процедуры от оператора `BEGIN` до оператора `END`;
- `definer` — учетная запись, из-под которой была создана хранимая процедура;
- `created` — дата и время создания хранимой процедуры;
- `modified` — дата и время последней модификации хранимой процедуры;
- `sql_mode` — режимы выполнения хранимой процедуры;
- `comment` — комментарий к хранимой процедуре.

Удобство данного подхода заключается в том, что при формировании отчета можно использовать всю гибкость, предоставляемую оператором `SELECT`, и извлекать только ту информацию, которая действительно необходима. В листинге 12.47 формируется список всех хранимых процедур и функций, причем выводится только название и признак — процедура или функция.

Листинг 12.47. Извлечение списка хранимых процедур

```
SELECT name, type FROM proc;
+--------------------------+-----------+
| name                     | type      |
+--------------------------+-----------+
| binrand                  | PROCEDURE |
| binstring                | PROCEDURE |
| catalogname              | PROCEDURE |
| catalog_by_product       | PROCEDURE |
| characteristics          | PROCEDURE |
| count_by_id              | PROCEDURE |
| count_product            | FUNCTION  |
| count_product_in_catalog | FUNCTION  |
| declare_var              | PROCEDURE |
| func_catalog             | FUNCTION  |
| inner_declare_var        | PROCEDURE |
| NOW5                     | PROCEDURE |
| NOWN                     | PROCEDURE |
| numcatalogs              | PROCEDURE |
| numcatalogsview          | PROCEDURE |
| one_declare_var          | PROCEDURE |
| pricelist                | PROCEDURE |
| say_hello                | FUNCTION  |
| set_x                    | PROCEDURE |
| set_y                    | PROCEDURE |
+--------------------------+-----------+
```

12.10. Содержимое хранимой процедуры

При помощи запроса к таблице `proc` базы данных `mysql` можно получить отдельные параметры хранимой процедуры или функции. Однако это не всегда удобно, особенно если хранимую процедуру или функцию необходимо воспроизвести на другом сервере MySQL. Для того чтобы получить оператор `CREATE PROCEDURE` или `CREATE FUNCTION`, необходимо воспользоваться операторами `SHOW CREATE PROCEDURE` и `SHOW CREATE FUNCTION` соответственно. Операторы имеют следующий синтаксис:

```
SHOW CREATE PROCEDURE procname;
SHOW CREATE FUNCTION funcname;
```

Пример использования оператора `SHOW CREATE PROCEDURE` приводится в листинге 12.48.

Листинг 12.48. Использование оператора `SHOW CREATE PROCEDURE`

```
SHOW CREATE PROCEDURE binstring\G;
*************************** 1. row ***************************
       Procedure: binstring
        sql_mode:
Create Procedure: CREATE PROCEDURE `shop`.`binstring`(IN num INT)
BEGIN
  DECLARE i INT DEFAULT 0;
  DECLARE bin TINYTEXT DEFAULT '';
  IF (num > 0) THEN
    wet : WHILE i < num DO
      SET i = i + 1;
      SET bin = CONCAT(bin, '11');
      IF !(i/2 — CEILING(i/2)) THEN ITERATE wet;
      END IF;
      SET bin = CONCAT(bin, '00');
    END WHILE wet;
    SELECT bin;
  ELSE
    SELECT 'Ошибочное значение параметра';
  END IF;
END
```

12.11. Удаление хранимых процедур и функций

Для удаления хранимых процедур и функций предназначены операторы `DROP PROCEDURE` и `DROP FUNCTION`, которые имеют следующий синтаксис:

```
DROP PROCEDURE [IF EXISTS] nameproc
DROP FUNCTION [IF EXISTS] nameproc
```

Оператор `DROP ROCEDURE` позволяет удалить хранимую процедуру *nameproc*. Если процедура с таким именем не существует, оператор возвращает ошибку, которую можно подавить, если использовать необязательное ключевое слово `IF EXISTS`.

Создадим хранимую процедуру `test()` и применим к ней оператор `DROP PROCEDURE` (листинг 12.49).

Листинг 12.49. Использование оператора `DROP PROCEDURE`

```
CREATE PROCEDURE test() SELECT VERSION();
SELECT name, type FROM mysql.proc WHERE db = 'test';
+------+-----------+
| name | type      |
+------+-----------+
| test | PROCEDURE |
+------+-----------+
DROP PROCEDURE test;
```

Однако использование оператора `DROP PROCEDURE` по отношению к хранимой функции заканчивается ошибкой (листинг 12.50).

Листинг 12.50. Удаление хранимой функции

```
CREATE FUNCTION test() RETURNS TINYTEXT RETURN VERSION();
SELECT name, type FROM mysql.proc WHERE db = 'test';
+------+----------+
| name | type     |
+------+----------+
| test | FUNCTION |
+------+----------+
DROP PROCEDURE test;
ERROR 1305 (42000): PROCEDURE test.test does not exist
SELECT name, type FROM mysql.proc WHERE db = 'test';
+------+----------+
| name | type     |
+------+----------+
| test | FUNCTION |
+------+----------+
```

Для удаления хранимых функций необходимо использовать специальный оператор `DROP FUNCTION` (листинг 12.51).

Листинг 12.51. Использование оператора DROP FUNCTION

```
SELECT name, type FROM mysql.proc WHERE db = 'test';
+------+----------+
| name | type     |
+------+----------+
| test | FUNCTION |
+------+----------+
DROP FUNCTION test;
```

12.12. Редактирование хранимых процедур

Для изменения характеристик хранимой процедуры предназначен оператор ALTER PROCEDURE. Редактирование хранимой функции выполняется с помощью оператора ALTER FUNCTION. Операторы имеют следующий синтаксис:

```
ALTER PROCEDURE sp_name [characteristic ...]
ALTER FUNCTION sp_name [characteristic ...]
```

Характеристика *characteristic* может принимать следующие значения:

- SQL SECURITY {DEFINER | INVOKER} — данное предложение определяет режим выполнения: хранимая процедура выполняется либо с правами создавшего ее пользователя (DEFINER), либо с правами пользователя, вызвавшего ее (INVOKER);
- COMMENT *'string'* — данное предложение позволяет назначить комментарий хранимой процедуре.

Замечание

Для выполнения операторов ALTER PROCEDURE и ALTER FUNCTION необходимо обладать привилегией ALTER ROUTINE. Данная привилегия автоматически передается пользователю, создавшему хранимую процедуру.

Пример использования оператора ALTER PROCEDURE приведен в листинге 12.52.

Листинг 12.52. Использование оператора ALTER PROCEDURE

```
CREATE PROCEDURE test() SELECT VERSION();
SELECT name, type, security_type, comment FROM mysql.proc
WHERE db = 'test';
+------+-----------+---------------+---------+
| name | type      | security_type | comment |
+------+-----------+---------------+---------+
| test | PROCEDURE | DEFINER       |         |
+------+-----------+---------------+---------+
```

```
ALTER PROCEDURE test
SQL SECURITY INVOKER
COMMENT 'Функция возвращает версию сервера';
SELECT name, type, security_type, comment FROM mysql.proc
WHERE db = 'test';
+------+-----------+---------------+----------------------------------+
| name | type      | security_type | comment                          |
+------+-----------+---------------+----------------------------------+
| test | PROCEDURE | INVOKER       | Функция возвращает версию сервера |
+------+-----------+---------------+----------------------------------+
```

Как видно из листинга 12.52, режим выполнения и комментарий к хранимой процедуре были успешно изменены.

12.13. Обработчики ошибок

Во время выполнения хранимых процедур и функций могут происходить самые разнообразные ошибки. При интерактивном выполнении запросов отследить возникающие ошибки и предпринять действия, направленные на их предотвращение, гораздо легче, чем при пакетном выполнении десятков запросов в хранимой процедуре. Поэтому СУБД MySQL поддерживает обработчики ошибок, которые позволяют каждой возникающей в хранимой процедуре ошибке назначить свой собственный обработчик. Кроме того, обработчик в зависимости от ситуации и серьезности ошибки может как прекратить, так и продолжить выполнение процедуры.

Для объявления такого обработчика предназначен оператор `DECLARE...HANDLER FOR`, который имеет следующий синтаксис:

`DECLARE` *`handler_type`* `HANDLER FOR` *`condition_value`*`[, ...]` *`sp_statement`*

Тип обработчика *`handler_type`* может принимать одно из трех значений:

- `CONTINUE` — выполнение текущей операции продолжается после выполнения оператора обработчика;
- `EXIT` — выполнение составного оператора `BEGIN...END`, в котором объявлен обработчик, прекращается;
- `UNDO` — данный вид обработчика пока не поддерживается.

Конструкция *`condition_value`* задает код ошибки, для которой будет происходить срабатывание обработчика. Обработчик может быть привязан сразу к нескольким ошибкам, для этого их коды следует перечислить через запятую. Конструкция *`condition_value`* может принимать одно из следующих значений:

- `SQLSTATE [VALUE]` *`sqlstate_value`* — значение `SQLSTATE` является пятисимвольным кодом ошибки в шестнадцатеричном формате и является стан-

дартом в SQL. В СУБД MySQL данный код поддерживается, начиная с версии 4.1. Примерами таких кодов являются `'HY000'`, `'HY001'`, `'42000'` и т. п. Следует отметить, что один код обозначает сразу несколько ошибок MySQL;

- `SQLWARNING` — любое предупреждение MySQL. Данное ключевое слово позволяет назначить обработчик для всех предупреждений MySQL. В терминах `SQLSTATE` обрабатываются любые события, для которых код `SQLSTATE` начинается с `'01'`;
- `NOT FOUND` — любая ошибка MySQL, связанная с отсутствием или невозможностью найти объект (таблицу, процедуру, функцию, столбец и т. п.). Данное ключевое слово позволяет назначить обработчик для всех ошибок такого рода. В терминах `SQLSTATE` обрабатываются любые события, для которых код `SQLSTATE` начинается с `'02'`;
- `SQLEXCEPTION` — ошибки, не охваченные ключевыми словами `SQLWARNING` и `NOT FOUND`;
- `mysql_error_code` — обычные четырехзначные ошибки MySQL, такие как `'1020'`, `'1232'`, `'1324'` и т. п.;
- *`condition_name`* — имя условия, которое объявляется при помощи оператора `DECLARE...CONDITON FOR`, рассмотренного далее.

Выражение *`sp_statement`* содержит SQL-запрос, который выполняется при срабатывании обработчика.

Для демонстрации приемов работы с обработчиками ошибок создадим таблицу `tbl`, содержащую единственный столбец `id`, являющийся первичным ключом (листинг 12.53).

Листинг 12.53. Создание таблицы `tbl`

```
CREATE TABLE tbl (id INT, PRIMARY KEY(id));
```

Так как поле `id` объявлено первичным ключом, его значения обязаны быть строго уникальными. Добавление в таблицу значений, совпадающих с одним из тех, которые уже существуют в таблице, приведет к возникновению ошибочной ситуации. Создадим хранимую процедуру `handler_key()`, которая содержит обработчик такой ситуации, и попытаемся добавить в таблицу `tbl` три одинаковых значения (листинг 12.54).

Листинг 12.54. Использование оператора `DECLARE...HANDLER FOR`

```
CREATE PROCEDURE handler_key ()
LANGUAGE SQL
```

```
BEGIN
  DECLARE CONTINUE HANDLER
  FOR SQLSTATE '23000' SET @error = 'Ошибка';
  INSERT INTO tbl VALUES(1);
  INSERT INTO tbl VALUES(1);
  INSERT INTO tbl VALUES(1);
  SELECT VERSION();
END
//
SELECT @error//
+--------+
| @error |
+--------+
| NULL   |
+--------+
CALL handler_key()//
+-----------------+
| VERSION()       |
+-----------------+
| 5.0.18-standard |
+-----------------+
SELECT @error//
+--------+
| @error |
+--------+
| Ошибка |
+--------+
```

Как видно из листинга 12.54, в результате обработки ошибки не происходит остановка работы процедуры и последний запрос `SELECT VERSION()` успешно выполняется, несмотря на то, что предпринимаются две попытки присвоить первичному ключу неуникальное значение. В результате срабатывания обработчика пользовательская переменная `@error` получает значение `'Ошибка'`.

В качестве обработчика ошибки можно использовать и составной оператор (листинг 12.55).

Листинг 12.55. Использование составного оператора в обработчике

```
CREATE PROCEDURE handler_key ()
LANGUAGE SQL
BEGIN
  DECLARE CONTINUE HANDLER
  FOR SQLSTATE '23000'
  BEGIN
    SET @error = 'Ошибка';
```

```
    SELECT 'Ошибка при вставке нового значения в таблицу tbl';
  END;
  INSERT INTO tbl VALUES(2);
  INSERT INTO tbl VALUES(2);
  INSERT INTO tbl VALUES(2);
END
//
CALL handler_key()//
+-------------------------------------------------+
| Ошибка при вставке нового значения в таблицу tbl |
+-------------------------------------------------+
| Ошибка при вставке нового значения в таблицу tbl |
+-------------------------------------------------+
+-------------------------------------------------+
| Ошибка при вставке нового значения в таблицу tbl |
+-------------------------------------------------+
| Ошибка при вставке нового значения в таблицу tbl |
+-------------------------------------------------+
SELECT @error//
+--------+
| @error |
+--------+
| Ошибка |
+--------+
```

Как видно из листинга 12.55, процедура выполняет все запросы, не прерываясь и лишь вызывая операторы из обработчика при возникновении каждой ошибки. Для того чтобы прекратить выполнение функции при возникновении ошибки, необходимо воспользоваться обработчиком `DECLARE EXIT` (листинг 12.56).

Листинг 12.56. Использование обработчика `EXIT`

```
CREATE PROCEDURE handler_key ()
LANGUAGE SQL
BEGIN
  DECLARE EXIT HANDLER
  FOR SQLSTATE '23000'
  BEGIN
    SET @error = 'Ошибка';
    SELECT 'Ошибка при вставке нового значения в таблицу tbl';
  END;
  INSERT INTO tbl VALUES(3);
  INSERT INTO tbl VALUES(3);
```

```
  INSERT INTO tbl VALUES(3);
  SELECT VERSION();
END
//
CALL handler_key()//
+-------------------------------------------------+
| Ошибка при вставке нового значения в таблицу tbl |
+-------------------------------------------------+
| Ошибка при вставке нового значения в таблицу tbl |
+-------------------------------------------------+
SELECT @error//
+--------+
| @error |
+--------+
| Ошибка |
+--------+
```

Как видно из листинга 12.56, если используется обработчик `EXIT`, то при возникновении первой же ошибки функция прекращает свою работу.

При указании кода ошибки можно использовать не только их целочисленные значения, но и именованные условия, которые объявляются при помощи оператора `DECLARE...CONDITION FOR`, имеющего следующий синтаксис:

`DECLARE` *`condition_name`* `CONDITION FOR` *`condition_value`*

Оператор объявляет именованное условие *`condition_name`* для ошибки *`condition_value`*, которая может принимать одно из следующих значений:

- `SQLSTATE [VALUE]` *`sqlstate_value`* — значение `SQLSTATE` является пятисимвольным кодом ошибки в шестнадцатеричном формате и стандартом в SQL. В СУБД MySQL данный код поддерживается, начиная с версии 4.1. Примерами таких кодов являются `'HY000'`, `'HY001'`, `'42000'` и т. п. Следует отметить, что один код обозначает сразу несколько ошибок MySQL;
- *`mysql_error_code`* — обычные четырехзначные ошибки MySQL, такие как `'1020'`, `'1232'`, `'1324'` и т. п.

Например, для обрабатываемой ранее ошибки 1062 (23000) — дублирование уникального индекса, оператор `DECLARE...CONDITION FOR` может выглядеть так, как это представлено в листинге 12.57.

Листинг 12.57. Использование оператора `DECLARE...CONDITION FOR`

```
DECLARE `violation` CONDITION FOR SQLSTATE '23000';
DECLARE `violation` CONDITION FOR 1062;
```

Первое объявление является более широким и охватывает все ошибки со статусом `'23000'`, второй вид ошибок более узкий и включает только дублирование уникального индекса.

В листинге 12.58 демонстрируется использование именованных условий, при этом следует обратить внимание, что название именованного условия заключается в необязательные обратные апострофы.

Листинг 12.58. Использование именованных условий

```
CREATE PROCEDURE handler_key ()
LANGUAGE SQL
BEGIN
  DECLARE `violation` CONDITION FOR 1062;
  DECLARE EXIT HANDLER FOR `violation` ROLLBACK;
  START TRANSACTION;
    INSERT INTO tbl VALUES(3);
    INSERT INTO tbl VALUES(3);
    INSERT INTO tbl VALUES(3);
  COMMIT;
END
//
```

В листинге 12.58 хранимая процедура выполняет транзакцию. При возникновении ошибки дублирования записей состояние базы данных откатывается при помощи оператора `ROLLBACK`.

12.14. Курсоры

Если результирующий запрос возвращает одну запись, поместить результаты в промежуточные переменные можно при помощи оператора `SELECT... INTO...FROM`. Однако результирующие таблицы чаще содержат несколько записей, и использование такого запроса совместно с оператором `SELECT...INTO...FROM` приводит к возникновению ошибки 1172: "Результат содержит более чем одну строку" (листинг 12.59).

Листинг 12.59. Ошибочное выполнение хранимой процедуры

```
CREATE PROCEDURE idcatalogs (OUT total INT)
BEGIN
  SELECT id_catalog INTO total FROM catalogs;
END
//
```

```
CALL idcatalogs(@total)//
ERROR 1172 (42000): Result consisted of more than one row
```

Избежать возникновения ошибки можно, добавив предложение `LIMIT 1` или назначив `CONTINUE`-обработчик ошибок. Однако функция будет реализовывать совсем не то поведение, которое ожидает пользователь. Кроме того, существуют ситуации, когда требуется обработать именно многострочную результирующую таблицу.

Например, пусть требуется вернуть записи таблицы `catalogs` учебной базы данных `shop` и на основании этих записей создать новую таблицу `letter_catalogs`, в которой названия элементов каталога будут представлены в верхнем регистре.

Решить эту задачу можно при помощи курсоров, которые позволяют в цикле просмотреть каждую строку результирующей таблицы запросов. Работа с курсорами происходит по следующему алгоритму:

1. При помощи инструкции `DECLARE CURSOR` имя курсора связывается с выполняемым запросом.
2. Оператор `OPEN` выполняет запрос, связанный с курсором, и устанавливает курсор перед первой записью результирующей таблицы.
3. Оператор `FETCH` помещает курсор на первую запись результирующей таблицы и извлекает данные из записи в локальные переменные хранимой процедуры. Повторный вызов оператора `FETCH` приводит к перемещению курсора к следующей записи, и так до тех пор, пока записи в результирующей таблице не будут исчерпаны. Эту операцию удобно осуществлять в цикле.
4. Оператор `CLOSE` прекращает доступ к результирующей таблице и ликвидирует связь между курсором и результирующей таблицей.

Замечание

Работа с курсорами похожа на работу с файлами — сначала происходит открытие курсора, затем чтение и после закрытие.

Оператор `DECLARE CURSOR` объявляет курсор и имеет следующий синтаксис:

```
DECLARE cursor_name CURSOR FOR select_statement
```

Оператор объявляет курсор с именем *`cursor_name`* для `SELECT`-запроса *`select_statement`*. В рамках хранимой процедуры имя *`cursor_name`* должно быть уникальным.

Замечание

В `SELECT`-запросе *`select_statement`* не допускается использовать запрос вида `SELECT...INTO...FROM`.

В момент объявления курсора при помощи оператора DECLARE CURSOR запрос *select_statement* не выполняется. Его выполнение откладывается до момента вызова оператора OPEN, который имеет следующий синтаксис:

```
OPEN cursor_name
```

Оператор OPEN принимает имя курсора *cursor_name*. Далее записи извлекаются при помощи оператора FETCH, который имеет следующий синтаксис:

```
FETCH cursor_name INTO var, var1, ...
```

После ключевого слова INTO должно быть приведено столько локальных переменных, сколько полей возвращает SELECT-запрос *select_statement*.

Оператор CLOSE имеет синтаксис, схожий с оператором OPEN:

```
CLOSE cursor_name
```

Вернемся к поставленной ранее задаче — созданию новой таблицы letter_catalogs с именами элементов каталога в верхнем регистре. Создадим таблицу letter_catalogs при помощи запроса, представленного в листинге 12.60.

Листинг 12.60. Создание таблицы letter_catalogs

```
CREATE TABLE IF NOT EXISTS letter_catalogs
(id_catalog INT,
 name TINYTEXT);
```

Тогда процедура, выполняющая поставленную задачу, может выглядеть так, как это представлено в листинге 12.61.

Листинг 12.61. Использование курсоров

```
CREATE PROCEDURE curcatalogs ()
BEGIN
  /* Объявляем локальные переменные */
  DECLARE id INT;
  DECLARE is_end INT DEFAULT 0;
  DECLARE cat TINYTEXT;

  /* Объявляем курсор */
  DECLARE curcat CURSOR FOR SELECT * FROM catalogs;

  /* Объявляем обработчик для ситуации, когда курсор достигает
     конца результирующей таблицы */
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET is_end = 1;
```

```
  /* Открываем курсор */
  OPEN curcat;

  /* В цикле читаем данные и курсор и добавляем записи */
  wet : LOOP
    FETCH curcat INTO id, cat;
    IF is_end THEN LEAVE wet;
    END IF;
    INSERT INTO letter_catalogs VALUES(id, UPPER(cat));
  END LOOP wet;

  /* Закрываем курсор */
  CLOSE curcat;
END
//
CALL curcatalogs()//
SELECT * FROM letter_catalogs//
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | ПРОЦЕССОРЫ         |
|          2 | МАТЕРИНСКИЕ ПЛАТЫ  |
|          3 | ВИДЕОАДАПТЕРЫ      |
|          4 | ЖЕСТКИЕ ДИСКИ      |
|          5 | ОПЕРАТИВНАЯ ПАМЯТЬ |
+------------+--------------------+
```

Как видно из листинга 12.61, хранимая процедура `curcatalogs()` заполняет таблицу `letter_catalogs` записями из таблицы `catalogs`, переводя названия элементов каталога в верхний регистр. Нетрудно переделать хранимую процедуру таким образом, чтобы она изменяла записи существующей таблицы `catalogs`. Для этого запрос

```
INSERT INTO letter_catalogs VALUES(id, UPPER(cat));
```

следует заменить `UPDATE`-запросом

```
UPDATE catalogs SET name = UPPER(cat) WHERE id_catalog = id;
```

12.15. Рекурсивные хранимые процедуры

Хранимую процедуру, которая вызывает сама себя, называют рекурсивной хранимой процедурой. Основная область применения рекурсивных процедур — деревья. Данные, которые имеют древовидную структуру, проще всего обрабатывать именно при помощи рекурсивных процедур.

Пусть имеется каталог бесконечной степени вложенности, хранящийся в таблице catalogs, дамп которой представлен в листинге 12.62.

Листинг 12.62. Каталог бесконечной степени вложенности

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  id_parent INT(11),
  PRIMARY KEY (id_catalog)
);
INSERT INTO catalogs VALUES (1, "Компьютеры", 0);
INSERT INTO catalogs VALUES (2, "Материнские платы", 1);
INSERT INTO catalogs VALUES (3, "Процессоры", 1);
INSERT INTO catalogs VALUES (4, "Intel", 3);
INSERT INTO catalogs VALUES (5, "AMD", 3);
INSERT INTO catalogs VALUES (6, "Видеокарты", 1);
INSERT INTO catalogs VALUES (7, "NVidia", 6);
INSERT INTO catalogs VALUES (8, "AMD", 6);
INSERT INTO catalogs VALUES (9, "Оперативная память", 1);
INSERT INTO catalogs VALUES (10, "Жесткие диски", 1);
INSERT INTO catalogs VALUES (11, "Оргтехника", 0);
INSERT INTO catalogs VALUES (12, "Принтеры", 11);
INSERT INTO catalogs VALUES (13, "Сканеры", 11);
INSERT INTO catalogs VALUES (14, "Копировальные аппараты", 11);
```

Таблица catalogs состоит из трех полей:

- id_catalog — первичный ключ таблицы, снабженный атрибутом AUTO_INCREMENT;
- name — название элемента каталога;
- id_parent — вторичный ключ, указывающий на первичный ключ родительской записи, для корневых каталогов значение данного поля равно 0.

Структура представленного в листинге 12.62 каталога выглядит следующим образом:

```
Компьютеры
  Материнские платы
  Процессоры
    Intel
    AMD
  Видеокарты
    NVidia
    ATI
```

```
  Оперативная память
  Жесткие диски
Оргтехника
  Принтеры
  Сканеры
  Копировальные аппараты
```

В листинге 12.63 представлена рекурсивная хранимая функция, которая по первичному ключу `id_catalog` удаляет элемент каталога и все подчиненные элементы каталога. Такая хранимая процедура может быть полезна при использовании типов таблиц, не поддерживающих каскадное удаление и обновление данных по внешнему ключу.

Замечание

СУБД MySQL не поддерживает рекурсивных функций.

Листинг 12.63. Рекурсивное удаление элемента каталога

```
CREATE PROCEDURE delcat (IN id INT)
BEGIN
  DECLARE is_end INT DEFAULT 0;
  DECLARE id_cat INT DEFAULT 0;

  DECLARE cat CURSOR FOR
  SELECT id_catalog FROM catalogs WHERE id_parent = id;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET is_end = 1;

  OPEN cat;
  wet : LOOP
    FETCH cat INTO id_cat;
    IF id_cat > 0 THEN CALL delcat(id_cat);
    END IF;
    DELETE FROM catalogs WHERE id_catalog = id_cat;
    IF is_end THEN LEAVE wet;
    END IF;
  END LOOP wet;
  CLOSE cat;
  DELETE FROM catalogs WHERE id_catalog = id;
END
//
```

Однако вызов процедуры `delcat()` на неподготовленной машине, скорее всего, закончится неудачно с выводом сообщения об ошибке: "ERROR 1456 (HY000): Recursive limit 0 (as set by the max_sp_recursion_depth variable) was

exceeded for routine delcat" ("Количество рекурсивных спусков функции delcat ограничено нулем, установите системную переменную max_sp_recursion_depth"). Значение системной переменной max_sp_recursion_depth по умолчанию равно 0, поэтому, чтобы функция могла осуществлять рекурсивный спуск, необходимо установить значение этой переменной, отличное от нуля (максимум 255). Так как максимальная глубина каталога не превышает 4, установим значение переменной, равное 4 (листинг 12.64).

Замечание

Более подробно порядок работы с системными переменными обсуждается в *главе 9*.

Листинг 12.64. Изменение значения переменной max_sp_recursion_depth

```
SET SESSION max_sp_recursion_depth = 4;
```

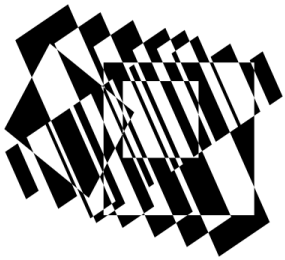
Теперь можно воспользоваться процедурой delcat(). В листинге 12.65 удаляется элемент каталога "Компьютеры" со всеми подчиненными элементами, для этого в качестве аргумента функции передается первичный ключ каталога — 1.

Листинг 12.65. Рекурсивное удаление каталога

```
SELECT * FROM catalogs//
+------------+-----------------------+-----------+
| id_catalog | name                  | id_parent |
+------------+-----------------------+-----------+
|          1 | Компьютеры            |         0 |
|          2 | Материнские платы     |         1 |
|          3 | Процессоры            |         1 |
|          4 | Intel                 |         3 |
|          5 | AMD                   |         3 |
|          6 | Видеокарты            |         1 |
|          7 | NVidia                |         6 |
|          8 | AMD                   |         6 |
|          9 | Оперативная память    |         1 |
|         10 | Жесткие диски         |         1 |
|         11 | Оргтехника            |         0 |
|         12 | Принтеры              |        11 |
|         13 | Сканеры               |        11 |
|         14 | Копировальные аппараты |        11 |
+------------+-----------------------+-----------+
CALL delcat(1)//
```

```
SELECT * FROM catalogs//
+------------+-----------------------+-----------+
| id_catalog | name                  | id_parent |
+------------+-----------------------+-----------+
|         11 | Оргтехника            |         0 |
|         12 | Принтеры              |        11 |
|         13 | Сканеры               |        11 |
|         14 | Копировальные аппараты |        11 |
+------------+-----------------------+-----------+
```

ГЛАВА 13



Триггеры

Триггер — эта специальная хранимая процедура, привязанная к событию изменения содержимого таблицы. Существуют три события изменения таблицы, к которым можно привязать триггер: это изменение содержимого таблицы при помощи операторов `INSERT`, `DELETE` и `UPDATE`.

Замечание

Поддержка триггеров введена в СУБД MySQL, начиная с версии 5.0.2.

13.1. Создание триггера

Создание триггера осуществляется при помощи оператора `CREATE TRIGGER`, который имеет следующий синтаксис:

```
CREATE TRIGGER trigger_name trigger_time trigger_event
    ON tbl_name FOR EACH ROW trigger_stmt
```

Оператор `CREATE TRIGGER` создает *`trigger_name`*, привязанный к таблице *`tbl_name`*. Таблица должна существовать физически, т. е. не допускается привязка триггера к временной таблице или представлению.

Конструкция *`trigger_time`* указывает момент выполнения триггера и может принимать два значения:

- `BEFORE` — действия триггера производятся до выполнения операции изменения таблицы;
- `AFTER` — действия триггера производятся после выполнения операции изменения таблицы.

Конструкция *`trigger_event`* показывает, на какое из событий должен реагировать триггер, и может принимать три значения:

- INSERT — триггер привязан к событию вставки новой записи в таблицу;
- UPDATE — триггер привязан к событию обновления записи таблицы;
- DELETE — триггер привязан к событию удаления записей таблицы.

Замечание

Для таблицы *tbl_name* может быть создан только один триггер для каждого из событий *trigger_event* и момента *trigger_time*. То есть для каждой из таблиц может быть создано всего шесть триггеров.

Конструкция *trigger_stmt* представляет тело триггера, т. е. оператор, который необходимо выполнить при возникновении события *trigger_event* в таблице *tbl_name*. Если требуется выполнить несколько операторов, то следует прибегать к составному оператору BEGIN...END, в котором размещаются все требуемые запросы.

Синтаксис и допустимые операторы совпадают с телом хранимых процедур *(см. главу 12)*. Внутри составного оператора BEGIN...END допускаются все специфичные для хранимых процедур операторы и конструкции:

- другие составные операторы BEGIN...END;
- операторы управления потоком (IF, CASE, WHILE, LOOP, REPEAT, LEAVE, ITERATE);
- объявления локальных переменных при помощи оператора DECLARE и присвоение им значений при помощи оператора SET;
- именованные условия и обработчики ошибок.

Замечание

В СУБД MySQL триггеры нельзя привязать к каскадному обновлению или удалению записей из таблицы типы InnoDB по связи "первичный ключ/внешний ключ".

Замечание

Для создания триггера при помощи оператора CREATE TRIGGER до версии MySQL 5.1.6 требуется наличие привилегии SUPER, после версии 5.1.6 требуется специальная привилегия TRIGGER.

Триггеры очень сложно использовать, не имея доступа к новым записям, которые вставляются в таблицу, или старым записям, которые обновляются или удаляются. Для доступа к новым и старым записям используются префиксы NEW и OLD соответственно. То есть если в таблице обновляется поле total, то получить доступ к старому значению можно по имени OLD.total, а к новому — NEW.total.

Создадим простейший триггер, который при оформлении нового заказа (добавление новой записи в таблицу orders) будет присваивать значение 1 пользовательской переменной @tot (листинг 13.1).

Листинг 13.1. Создание простейшего триггера

```
CREATE TRIGGER sub_count AFTER INSERT ON orders
FOR EACH ROW
BEGIN
  SET @tot = 1;
END
//
select @tot//
+------+
| @tot |
+------+
| NULL |
+------+
INSERT INTO orders VALUES (NULL,1,NOW(),1,10)//
select @tot//
+------+
| @tot |
+------+
| 1    |
+------+
```

Как видно из листинга 13.1, в результате добавления новой записи в таблицу `orders` пользовательской переменной присваивается значение 1. Отредактируем триггер `sub_count` таким образом, чтобы к пользовательской переменной `@tot` прибавлялось каждый раз число заказанных товарных позиций `number` (листинг 13.2).

Листинг 13.2. Новый вариант триггера `sub_count`

```
CREATE TRIGGER sub_count AFTER INSERT ON orders
FOR EACH ROW
BEGIN
  SET @tot = @tot + NEW.number;
END
//
select @tot//
+------+
| @tot |
+------+
| 1    |
+------+
INSERT INTO orders VALUES (NULL,1,NOW(),5,10)//
select @tot//
```

```
+------+
| @tot |
+------+
| 6    |
+------+
```

Как видно из листинга 13.2, для того чтобы получить число товарных позиций, внутри триггера происходит обращение к переменной `NEW.number`, которая связана с полем `number` `INSERT`-запроса.

Замечание

Для корректной работы триггера необходимо, чтобы пользовательская переменная `@tot` имела значение, отличное от `NULL`, т. к. операции сложения с `NULL` приводят к `NULL`.

Замечание

При создании триггеров для таблицы в каталоге данных формируется файл, название которого совпадает с именем таблицы. Файл является текстовым и имеет расширение trg.

Предыдущие два примера демонстрировали работу триггеров после добавления записи в таблицу (`AFTER`) без вмешательства в запрос. Рассмотрим триггер, который будет включаться до (`BEFORE`) вставки новых записей в таблицу `orders`. Основная задача триггера заключается в ограничении числа заказываемых товаров до 1 (листинг 13.3).

Листинг 13.3. Использование ключевого слова `BEFORE`

```
CREATE TRIGGER restrict_count BEFORE INSERT ON orders
FOR EACH ROW
BEGIN
  SET NEW.number = 1;
END
//
INSERT INTO orders VALUES (NULL,1,NOW(),2,10)//
SELECT * FROM orders WHERE id_order = LAST_INSERT_ID()//
+----------+---------+--------------------+--------+------------+
| id_order | id_user | ordertime          | number | id_product |
+----------+---------+--------------------+--------+------------+
|       16 |       1 | 2005-07-22 21:40:02 |      1 |         10 |
+----------+---------+--------------------+--------+------------+
```

Как видно из листинга 13.3, несмотря на то, что заказ оформлялся на две товарные позиции, триггер `restrict_count` изменил значение поля `number` на 1.

Часто при обновлении одних полей таблицы операторы баз данных забывают обновить связанные поля таблицы или производится попытка добавления некорректных значений. Пусть при добавлении нового покупателя необходимо преобразовать имена и отчества покупателей в инициалы. Триггер для обработки этой ситуации может выглядеть так, как это представлено в листинге 13.4.

Листинг 13.4. Контролирующий триггер

```
CREATE TRIGGER restrict_user BEFORE INSERT ON users
FOR EACH ROW
BEGIN
  SET NEW.name = LEFT(NEW.name,1);
  SET NEW.patronymic = LEFT(NEW.patronymic,1);
END
//
INSERT INTO users VALUES (NULL, 'Ремизов', 'Алексеевич', 'Сергей',
                          '83-89-00', NULL, NULL, 'active')//
SELECT surname, patronymic, name FROM users
WHERE id_user = LAST_INSERT_ID()//
+---------+------------+------+
| surname | patronymic | name |
+---------+------------+------+
| Ремизов | A          | C    |
+---------+------------+------+
```

Как видно из листинга 13.4, имя и отчество любого нового посетителя урезается до одной буквы. Для того чтобы имя и отчество не были отредактированы при помощи оператора `UPDATE`, следует также создать триггер, привязанный к событию `UPDATE`.

13.2. Удаление триггера

Для удаления триггера предназначен оператор `DROP TRIGGER`, который имеет следующий синтаксис:

```
DROP TRIGGER tbl_name.trigger_name
```

Оператор удаляет триггер с именем *`trigger_name`* таблицы *`tbl_name`*. В листинге 13.5 демонстрируется удаление триггера `restrict_user` таблицы `users`, созданного в листинге 13.4.

Листинг 13.5. Удаление триггера `restrict_user`

```
DROP TRIGGER users.restrict_user;
```

13.3. Список созданных триггеров

Получить список триггеров, доступных текущему пользователю, можно при помощи оператора `SHOW TRIGGERS`, который имеет следующий синтаксис:

```
SHOW TRIGGERS [FROM db_name] [LIKE expr]
```

Оператор позволяет извлечь список и характеристики триггера из базы данных *db_name*. Если база данных имеет большое количество триггеров, результирующую таблицу можно ограничить при помощи ключевого слова `LIKE`, имеющего тот же синтаксис, что и ключевое слово `LIKE` в операторе `SELECT`. В листинге 13.6 приводится пример использования оператора `SHOW TRIGGERS`.

Замечание

Оператор `SHOW TRIGGERS` введен, начиная с MySQL 5.0.10.

Листинг 13.6. Использование оператора `SHOW TRIGGERS`

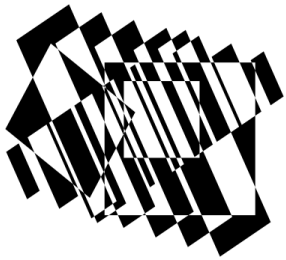
```
SHOW TRIGGERS\G
*************************** 1. row ***************************
  Trigger: restrict_user
    Event: INSERT
    Table: users
Statement: BEGIN
  SET NEW.name = LEFT(NEW.name,1);
  SET NEW.patronymic = LEFT(NEW.patronymic,1);
END
   Timing: BEFORE
  Created: NULL
 sql_mode: ALLOW_INVALID_DATES
  Definer: root@localhost
```

Столбцы из результирующей таблицы оператора `SHOW TRIGGERS` имеют следующий смысл:

- `Trigger` — название триггера;
- `Event` — тип события, который служит сигналом для запуска триггера, может принимать три значения: `INSERT`, `UPDATE` и `DELETE`;
- `Table` — название таблицы, к которой привязан триггер;

- `Statement` — тело триггера, именно этот блок будет выполняться при запуске триггера;
- `Timing` — момент выполнения триггера: до или после срабатывания оператора операции `Event`, данный параметр может принимать два значения: `BEFORE` и `AFTER`;
- `Created` — зарезервированный параметр, в настоящий момент всегда принимающий значение `NULL`;
- `sql_mode` — SQL-режим, который действует во время выполнения триггера;
- `Definer` — учетная запись, из-под которой был создан триггер.

ГЛАВА 14



Представления

Основными структурными единицами в реляционных базах данных являются таблицы. Однако язык запросов SQL предоставляет еще один способ организации данных — представления. *Представление* — это запрос на выборку (`SELECT`), которому присваивается уникальное имя и который можно сохранять или удалять из базы данных как обычную хранимую процедуру. Представления позволяют увидеть результаты сохраненного запроса таким образом, как будто это полноценная таблица базы данных. Представления позволяют более гибко управлять правами доступа к таблицам, так можно запретить прямое обращение пользователей к таблицам, а разрешить доступ только к представлениям. Представления позволяют также обеспечить обратную совместимость для программ, ориентирующихся на старую структуру базы данных — достаточно создать представления со структурой, соответствующей старым таблицам.

Замечание

Представления и все связанные с ними операторы доступны в СУБД MySQL, начиная с версии 5.0.1.

14.1. Создание представлений

Создание представлений осуществляется при помощи оператора `CREATE VIEW`, который имеет следующий синтаксис:

```
CREATE
    [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = { user | CURRENT_USER }]
    [SQL SECURITY { DEFINER | INVOKER }]
```

```
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Оператор создает представление *view_name* со столбцами, перечисленными в *column_list*, на основании SELECT-запроса *select_statement*.

Пусть имеется таблица catalogs, дамп которой представлен в листинге 14.1.

Листинг 14.1. Таблица catalogs

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL,
  name TINYTEXT NOT NULL,
  PRIMARY KEY (id_catalog)
);
INSERT INTO catalogs VALUES (3, 'Видеоадаптеры');
INSERT INTO catalogs VALUES (4, 'Жесткие диски');
INSERT INTO catalogs VALUES (2, 'Материнские платы');
INSERT INTO catalogs VALUES (5, 'Оперативная память');
INSERT INTO catalogs VALUES (1, 'Процессоры');
```

В листинге 14.2 приводится пример создания представления cat, которое дублирует SELECT-запрос к таблице catalogs.

Листинг 14.2. Создание представления cat

```
CREATE VIEW cat AS SELECT * FROM catalogs ORDER BY name;
SELECT * FROM cat;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры         |
+------------+--------------------+
```

Как видно из листинга 14.2, при формировании представления результирующая таблица сортируется при помощи ключевого слова ORDER BY.

В каталоге данных для представления cat будет создан файл cat.frm, который в отличие от файлов определения структуры таблиц является не бинарным, а текстовым (листинг 14.3).

Замечание

В зависимости от используемой версии СУБД MySQL структура файла, в котором хранится определение представления, может отличаться от приведенного в листинге 14.3.

Листинг 14.3. Структура файла cat.frm представления `cat`

```
TYPE=VIEW
query=select `test`.`catalogs`.`id_catalog` AS
`id_catalog`,`test`.`catalogs`.`name` AS `name` from `test`.`catalogs` order
by `test`.`catalogs`.`name`
md5=6a74c911b3180b7838d8bfd3ddc53563
updatable=0
algorithm=0
definer_user=root
definer_host=localhost
suid=2
with_check_option=0
revision=1
timestamp=2007-02-09 20:53:48
create-version=1
source=SELECT * FROM catalogs ORDER BY name
```

Представление рассматривается СУБД MySQL как полноценная таблица и может быть просмотрено в списке таблиц базы данных при помощи оператора `SHOW TABLES` (листинг 14.4).

Листинг 14.4. Просмотр представления при помощи оператора `SHOW TABLES`

```
SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| cat            |
| catalogs       |
| forums         |
| goods          |
| products       |
+----------------+
```

В листинге 14.4 список столбцов представления не уточняется, поэтому представление принимает структуру таблицы `catalogs`. Однако при создании представления можно явно указать список столбцов и даже изменить их названия и порядок следования (листинг 14.5).

Листинг 14.5. Изменения названия столбцов и порядка их следования

```
CREATE VIEW cat (catalog, id)
AS SELECT name, id_catalog FROM catalogs;
SELECT * FROM cat;
+--------------------+----+
| catalog            | id |
+--------------------+----+
| Видеоадаптеры      |  3 |
| Жесткие диски      |  4 |
| Материнские платы  |  2 |
| Оперативная память |  5 |
| Процессоры         |  1 |
+--------------------+----+
```

Как видно из листинга 14.5, название столбца `name` меняется на `catalog`, а `id_catalog` на `id`. При этом порядок их следования меняется на обратный. При формировании списка столбцов представления задаются только имена столбцов, тип данных, их размер и другие характеристики берутся из определения столбца исходной таблицы.

В качестве столбцов представления могут выступать вычисляемые столбцы. Создадим представление `namecat`, которое выводит столбцы `id_catalog` и `name` таблицы `catalogs`, а также столбец `total`, в который будет помещаться количество символов в имени элемента каталога (листинг 14.6).

Листинг 14.6. Создание представления с вычисляемыми столбцами

```
CREATE OR REPLACE VIEW namecat (id_catalog, name, total)
AS SELECT id_catalog, name, LENGTH(name) FROM catalogs;
SELECT * FROM namecat ORDER BY total DESC;
+------------+--------------------+-------+
| id_catalog | name               | total |
+------------+--------------------+-------+
|          5 | Оперативная память |    18 |
|          2 | Материнские платы  |    17 |
|          3 | Видеоадаптеры      |    13 |
|          4 | Жесткие диски      |    13 |
|          1 | Процессоры         |    10 |
+------------+--------------------+-------+
```

Предложение `ALGORITHM` (листинг 14.7) определяет алгоритм формирования конечного запроса с участием представления и может принимать три значения:

- MERGE — при использовании данного алгоритма запрос объединяется с представлением таким образом, что представление заменяет собой соответствующие части в запросе;
- TEMPTABLE — результирующая таблица представления помещается во временную таблицу, которая затем используется в конечном запросе;
- UNDEFINED — в данном случае СУБД MySQL самостоятельно пытается выбрать алгоритм, предпочитая использовать подход MERGE и прибегая к алгоритму TEMPTABLE (создание временной таблицы) только в случае необходимости, т. к. метод MERGE более эффективен.

Замечание

Если ни одно из значений ALGORITHM не указано, по умолчанию назначается UNDEFINED.

Листинг 14.7. Использование ключевого слова ALGORITHM

```
mysql> CREATE ALGORITHM = TEMPTABLE VIEW cat AS SELECT * FROM catalogs;
```

Запрос в листинге 14.7 требует от MySQL при каждом обращении к представлению cat создавать временную таблицу. При создании представления с помощью запроса, приведенного в листинге 14.8, временная таблица для результирующей таблицы SELECT-запроса не будет создаваться никогда.

Листинг 14.8. Использование ключевого слова MERGE

```
mysql> CREATE ALGORITHM = MERGE VIEW cat AS SELECT * FROM catalogs;
```

Ключевое слово SQL SECURITY, так же как и в хранимых процедурах, может быть записано в двух формах: SQL SECURITY DEFINER и SQL SECURITY INVOKER. Если используется форма SQL SECURITY DEFINER, то представление доступно только пользователю, создавшему ее, или пользователю, указанному в параметре DEFINER. При использовании SQL SECURITY INVOKER представление доступно любому пользователю, который имеет привилегию SHOW VIEW.

Ключевое слово DEFINER определяет, кому принадлежит представление, по умолчанию — это текущий пользователь (DEFINER = CURRENT_USER), но можно задать произвольного пользователя в формате *'user'@'host'*, где *user* — имя пользователя, а *host* — хост, с которого ему разрешено обращение к MySQL-серверу.

Замечание

Если в параметре DEFINER указывается имя пользователя, а не текущий пользователь при помощи ключевого слова CURRENT_USER или функции CURRENT_USER(), следует учитывать, что указываемый пользователь обязан иметь привилегию SUPER.

14.2. Сокрытие столбцов

Следует отметить, что представления способны скрывать ряд столбцов за счет того, что SELECT-запросы могут извлекать не все столбцы таблицы. Такие представления называются *вертикальными представлениями*. Вертикальные представления широко применяются для ограничения доступа пользователей к столбцам таблицы.

Пусть имеется таблица users, в которой содержится разнообразная информация о пользователях, включая их e-mail, телефон и URL (листинг 14.9).

Листинг 14.9. Таблица с личной информацией о пользователях

```
CREATE TABLE users (
  id_user INT(11) NOT NULL AUTO_INCREMENT,
  surname TINYTEXT,
  patronymic TINYTEXT,
  name TINYTEXT,
  phone varchar(12) default null,
  email TINYTEXT,
  url TINYTEXT,
  `status` ENUM('active','passive','lock','gold') DEFAULT 'active',
  PRIMARY KEY (id_user)
);
INSERT INTO users VALUES(1, 'Иванов', 'Валерьевич', 'Александр', '58-98-78',
'ivanov@email.ru', NULL, 'active');
INSERT INTO users VALUES (2, 'Лосев', 'Иванович', 'Сергей', '9057777777',
'losev@email.ru', NULL, 'passive');
INSERT INTO users VALUES (3, 'Симдянов', 'Вячеславович', 'Игорь',
'9056666100', 'simdyanov@softtime.ru', 'http://www.softtime.ru/', 'active');
INSERT INTO users VALUES (4, 'Кузнецов', 'Валерьевич', 'Максим', NULL,
'kuznetsov@softtime.ru', 'http://www.softtime.ru', 'active');
INSERT INTO users VALUES (5, 'Нехорошев', 'Юрьевич', 'Анатолий', NULL, NULL,
NULL, 'lock');
INSERT INTO users VALUES (6, 'Корнеев', 'Александрович', 'Александр', '89-78-
36', 'korneev@domen.ru', NULL, 'gold');
```

Можно создать представление list_user, которое будет отображать фамилию и инициалы пользователя, скрывая поля с контактными телефонами и адресами электронной почты (листинг 14.10).

Листинг 14.10. Создание представления списка пользователей `list_user`

```
CREATE OR REPLACE VIEW list_user
AS SELECT CONCAT(surname, " ",
                 SUBSTRING(name,1,1), ".",
                 SUBSTRING(patronymic,1,1), ".") AS name
FROM users ORDER BY name;
SELECT * FROM list_user;
+----------------+
| name           |
+----------------+
| Иванов А.В.    |
| Корнеев А.А.   |
| Кузнецов М.В.  |
| Лосев С.И.     |
| Нехорошев А.Ю. |
| Симдянов И.В.  |
+----------------+
```

Таким образом, псевдотаблица `list_user` всегда содержит текущий список пользователей, отсортированный по алфавиту. Запросы к представлениям сами могут содержать условия `WHERE` и собственные сортировки, правильное составление запроса к исходным таблицам — это забота MySQL. В листинге 14.11 выводится список пользователей, отсортированный в обратном порядке, который получен с помощью представления `list_user`.

Листинг 14.11. Список пользователей, отсортированный в обратном порядке

```
SELECT * FROM list_user ORDER BY name DESC;
+----------------+
| name           |
+----------------+
| Симдянов И.В.  |
| Нехорошев А.Ю. |
| Лосев С.И.     |
| Кузнецов М.В.  |
| Корнеев А.А.   |
| Иванов А.В.    |
+----------------+
```

При сортировке результатов запроса к представлению `list_user` предложение `ORDER BY` самого представления игнорируется.

14.3. Сокрытие строк

Наряду с вертикальными представлениями используются *горизонтальные представления*, которые ограничивают доступ пользователей к строкам таблиц, делая видимыми только те строки, с которыми они работают.

Воспользуемся таблицей пользователей `users` из листинга 14.9. Пусть требуется создать два представления `a_l` и `l_z`. Первое представление `a_l` будет извлекать всех пользователей, чья фамилия начинается с первой половины алфавита, от А до Л. Второе представление `l_z` будет извлекать пользователей, чья фамилия начинается с букв из второй половины алфавита, от М до Я.

Листинг 14.12. Представления `a_l` и `l_z`

```
CREATE VIEW a_l
AS SELECT * FROM users
WHERE SUBSTRING(surname,1,1) BETWEEN 'A' AND 'Л';
CREATE VIEW l_z
AS SELECT * FROM users
WHERE SUBSTRING(surname,1,1) BETWEEN 'M' AND 'Я';
SELECT surname, patronymic, name FROM a_l;
+----------+---------------+-----------+
| surname  | patronymic    | name      |
+----------+---------------+-----------+
| Иванов   | Валерьевич    | Александр |
| Лосев    | Иванович      | Сергей    |
| Кузнецов | Валерьевич    | Максим    |
| Корнеев  | Александрович | Александр |
+----------+---------------+-----------+
SELECT surname, patronymic, name
+-----------+--------------+----------+
| surname   | patronymic   | name     |
+-----------+--------------+----------+
| Симдянов  | Вячеславович | Игорь    |
| Нехорошев | Юрьевич      | Анатолий |
+-----------+--------------+----------+
```

В реальной практике могут встречаться смешанные представления, которые ограничивают таблицу и по горизонтали, и по вертикали. Термины "горизонтальное представление" и "вертикальное представление" являются условными и предназначены для того, чтобы лучше понять, как из исходной таблицы формируется представление.

14.4. Обновление и удаление записей при помощи представлений

Использование конструкции WITH CHECK OPTION при создании представления при помощи оператора CREATE VIEW позволяет применять к представлению операторы INSERT и UPDATE для вставки или обновления записей в таблице. Во время вставки происходит проверка, чтобы вставляемые данные удовлетворяли WHERE-условию SELECT-запроса, лежащего в основе представления.

Замечание

Предложение WITH CHECK OPTION было добавлено в СУБД MySQL, начиная с версии 5.0.2.

Ключевые слова LOCAL и CASCADED в предложении WITH CHECK OPTION определяют область видимости ограничения применительно к другим представлениям. Ключевое слово LOCAL сообщает, что ограничение WITH CHECK OPTION распространяется только на текущее представление, а WHERE-ограничения представлений, на которые ссылается текущее представление, не влияют на процесс вставки новых данных. Тогда как ключевое слово CASCADED требует, чтобы проверка на соответствие вставляемых данных учитывала WHERE-условия и тех представлений, на которые ссылается текущее представление.

Замечание

Если ни одно из ключевых слов LOCAL или CASCADED не указано, считается, что выбран вариант LOCAL.

Для демонстрации эффекта предложения WITH CHECK OPTION создадим таблицу tbl с одним-единственным целочисленным столбцом a (листинг 14.13).

Листинг 14.13. Использование предложения WITH CHECK OPTION

```
CREATE TABLE tbl (a INT);
CREATE VIEW v1 AS SELECT * FROM tbl WHERE a < 2
WITH CHECK OPTION;
CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0
WITH LOCAL CHECK OPTION;
CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0
WITH CASCADED CHECK OPTION;
```

Кроме того, в листинге 14.13 создается представление v1, ограничивающее выборку значениями меньше двух. На представление v1 ссылаются представления v2 и v3, которые ограничивают выборку снизу значениями больше нуля. Представление v2 ограничивает вставку новых значений при помощи

предложения WITH LOCAL CHECK OPTION и, следовательно, не учитывает ограничение a < 2 из родительского представления. Представление v3 ограничивает вставку новых данных при помощи предложения WITH CASCADED CHECK OPTION, и, следовательно, реально вставка новых значений ограничена условиями 0 < a < 2 (листинг 14.14).

Листинг 14.14. Вставка новых значений в представления v1, v2 и v3

```
INSERT INTO v1 VALUES (-1);
Query OK, 1 row affected (0.03 sec)
INSERT INTO v1 VALUES (2);
ERROR 1369: CHECK OPTION failed 'test.v1'
INSERT INTO v2 VALUES (-1);
ERROR 1369: CHECK OPTION failed 'test.v2'
INSERT INTO v2 VALUES (2);
Query OK, 1 row affected (0.05 sec)
INSERT INTO v3 VALUES (-1);
ERROR 1369: CHECK OPTION failed 'test.v3'
INSERT INTO v3 VALUES (2);
ERROR 1369: CHECK OPTION failed 'test.v3'
```

14.5. Редактирование представления

Для редактирования представлений предназначен оператор ALTER VIEW, который имеет следующий синтаксис:

```
ALTER
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = { user | CURRENT_USER }]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Оператор ALTER TABLE практически полностью копирует структуру оператора CREATE VIEW и позволяет переопределить структуру. На самом деле эффект операторов ALTER VIEW и CREATE OR REPLACE VIEW один и тот же. В листинге 14.15 оба запроса являются эквивалентными.

Листинг 14.15. Использование оператора ALTER VIEW

```
ALTER VIEW cat AS SELECT * FROM catalogs;
CREATE OR REPLACE VIEW cat AS SELECT * FROM catalogs;
```

Замечание

Применение оператора ALTER VIEW требует привилегии CREATE VIEW и DELETE, а также привилегий SELECT для каждого столбца, используемого в представлении.

14.6. Удаление представлений

Для удаления представлений предназначен оператор DROP VIEW, который имеет следующий синтаксис

```
DROP VIEW [IF EXISTS]
    view_name [, view_name] ...
```

Оператор DROP VIEW позволяет уничтожить представление или сразу несколько представлений по их именам (листинг 14.16).

Листинг 14.16. Использование оператора DROP VIEW

```
DROP VIEW cat, tbl1, tbl2;
SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| catalogs       |
| forums         |
| goods          |
| products       |
+----------------+
```

Замечание

Применение по отношению к представлению оператора DROP TABLE приводит к возникновению ошибки 1051: "Unknown table 'cat'" ("Неизвестная таблица 'cat'").

Необязательное ключевое слово IF EXISTS позволяет избежать ошибки при использовании оператора DROP VIEW к уже удаленному представлению (листинг 14.17). Это особенно удобно при пакетном выполнении группы операторов, когда эта незначительная ошибка может привести к остановке выполнения всех оставшихся операторов в пакетном файле.

Листинг 14.17. Использование ключевого слова IF EXISTS

```
DROP VIEW cat;
ERROR 1051: Unknown table 'test.cat'
DROP VIEW IF EXISTS cat;
Query OK, 0 rows affected (0.00 sec)
```

Если в прикладной программе все же требуется выяснить статус завершения операции удаления представления, удобнее воспользоваться оператором SHOW WARNINGS (листинг 14.18).

Листинг 14.18. Возвращение статуса удаления представления

```
SHOW WARNINGS;
+-------+------+-------------------------+
| Level | Code | Message                 |
+-------+------+-------------------------+
| Note  | 1051 | Unknown table 'test.cat' |
+-------+------+-------------------------+
```

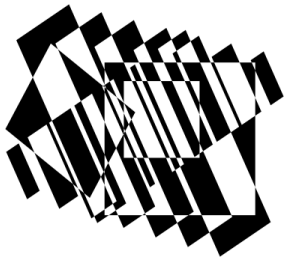
14.7. Просмотр структуры представления

Оператор SHOW CREATE VIEW позволяет просмотреть структуру оператора CREATE VIEW, при помощи которого было создано представление. Следует отметить, что оператор возвращает полную форму оператора CREATE VIEW, включая все опции и расширенные имена (листинг 14.19). Оператор возвращает содержимое файла с расширением frm, соответствующего представлению (листинг 14.19).

Листинг 14.19. Использование оператора SHOW CREATE VIEW

```
CREATE OR REPLACE VIEW cat AS SELECT * FROM catalogs;
SHOW CREATE VIEW cat\G;
*************************** 1. row ***************************
       View: cat
Create View: CREATE ALGORITHM=UNDEFINED VIEW `test`.`cat` AS select
`test`.`catalogs`.`id_catalog` AS `id_catalog`,`test`.`catalogs`.`name` AS
`name` from `test`.`catalogs`
```

ГЛАВА 15



Информационная схема

В предыдущей главе были рассмотрены представления, надструктурные образования которых позволяют создать собственный интерфейс к таблицам базы данных. Реальная структура таблиц может изменяться, например, с целью увеличения производительности, однако при помощи представлений всегда можно предоставлять старую структуру таблиц для клиентов, переработка которых с учетом новой структуры базы данных не представляется целесообразным.

Каждая СУБД имеет системную базу данных, в которой хранятся учетные записи, таблицы привилегий и другая информация, необходимая для управления СУБД. В СУБД MySQL такая системная база данных носит название `mysql`. Структура системной базы данных для разных СУБД отличается. Для того чтобы унифицировать процесс обращения к системной базе данных, вводится специальный набор представлений, оформленных в виде базы данных `INFORMATION_SCHEMA`, которая доступна каждому клиенту MySQL. База данных `INFORMATION_SCHEMA` является виртуальной и располагается в оперативной памяти — для нее не имеется физического соответствия на жестком диске, как для других баз данных. Это означает, что невозможно выбрать базу данных `INFORMATION_SCHEMA` при помощи оператора `USE`, как и выполнить по отношению к таблицам этой базы данных запросы с участием операторов `INSERT`, `UPDATE` и `DELETE`. Допускается использование только оператора `SELECT`. Однако строка, соответствующая `INFORMATION_SCHEMA`, присутствует в результирующей таблице оператора `SHOW DATABASES` (листинг 15.1).

Листинг 15.1. База данных `INFORMATION_SCHEMA`

```
SHOW DATABASES//
+--------------------+
| Database           |
+--------------------+
```

```
| information_schema |
| mysql              |
| test               |
+--------------------+
```

До введения информационной схемы получить разнообразную информацию о состоянии СУБД MySQL и ее компонентов можно было при помощи операторов SHOW. В табл. 15.1 приводятся представления информационной схемы, их краткое описание и соответствие оператору SHOW.

Замечание

Зачастую представление информационной схемы и один из операторов SHOW соответствуют друг другу лишь частично, для ряда представлений вообще не существует соответствующего оператора SHOW.

Таблица 15.1. *Представления информационной схемы*

Представление	Оператор SHOW	Описание
SCHEMATA	SHOW DATABASES	Содержит список и характеристики баз данных, доступных текущему пользователю
TABLES	SHOW TABLE STATUS	Содержит список таблиц и их характеристик
COLUMNS	SHOW COLUMNS	Содержит информацию о доступных текущему пользователю столбцах во всех таблицах всех баз данных
STATISTICS	SHOW INDEX	Содержит разнообразную информацию об индексах
USER_PRIVILEGES		Содержит информацию о глобальных привилегиях базы данных
SCHEMA_PRIVILEGES		Содержит глобальные привилегии всех пользователей сервера MySQL
TABLE_PRIVILEGES		Содержит информацию о табличных привилегиях
COLUMN_PRIVILEGES		Содержит информацию о привилегиях текущего пользователя на столбцы таблиц
CHARACTER_SETS	SHOW CHARACTER SET	Содержит список и характеристики кодировок, доступных текущему пользователю

Таблица 15.1 *(продолжение)*

Представление	Оператор SHOW	Описание
COLLATIONS	SHOW COLLATION	Содержит список и характеристики сортировок, доступных текущему пользователю
COLLATION_CHARACTER_SET_APPLICABILITY		Содержит всевозможные комбинации кодировок и сортировок, доступные текущему пользователю
TABLE_CONSTRAINTS		Содержит информацию об ограничивающих индексах, т. е. индексах, которые имеют ограничение уникальности значения (PRIMARY KEY, UNIQUE) или ограничение внешнего ключа (FOREIGN KEY)
KEY_COLUMN_USAGE		Содержит информацию об индексированных столбцах, доступных текущему пользователю
ROUTINES	SHOW PROCEDURE STATUS, SHOW FUNCTION STATUS	Содержит список и параметры хранимых процедур и функций, доступных текущему пользователю для выполнения
VIEWS		Содержит информацию о представлениях, доступных текущему пользователю
TRIGGERS	SHOW TRIGGERS	Содержит информацию о триггерах, доступных текущему пользователю
PLUGINS	SHOW PLUGINS	Содержит информацию о доступных плагинах MySQL
ENGINES	SHOW ENGINES	Содержит информацию о доступных типах таблиц
PARTITIONS		Содержит информацию о разделах сегментирования *(см. главу 17)*
EVENTS	SHOW EVENTS	Содержит информацию о заданиях планировщика задач *(см. главу 18)*
FILES		Содержит информацию о файлах, участвующих в создании кластера
PROCESSLIST	SHOW PROCESSLIST	Содержит информацию о выполняющихся на сервере процессах

Таблица 15.1 *(окончание)*

Представление	Оператор SHOW	Описание
REFERENTIAL_CONSTRAINTS		Содержит информацию о внешних ключах
GLOBAL_STATUS	SHOW GLOBAL STATUS	Содержит информацию о глобальных настройках сервера MySQL (эти настройки могут быть отменены сессионными настройками)
SESSION_STATUS	SHOW SESSION STATUS	Содержит информацию о сессионных настройках сервера применительно к текущему соединению
GLOBAL_VARIABLES	SHOW GLOBAL VARIABLES	Содержит информацию о глобальных переменных сервера MySQL (эти переменные могут быть отменены сессионными переменными)
SESSION_VARIABLES	SHOW SESSION VARIABLES	Содержит информацию о сессионных переменных сервера MySQL

Рассмотрим несколько наиболее интересных представлений из информационной схемы.

15.1. Список баз данных

Представление SCHEMATA содержит список и характеристики баз данных, доступных текущему пользователю. Аналогичную информацию выдает оператор SHOW DATABASES, но в отличие от представления SCHEMATA выдается только список баз данных без характеристик. Соответствие столбцов представления SCHEMATA и результирующей таблицы, возвращаемой оператором SHOW DATABASES, показано в табл. 15.2.

Таблица 15.2. *Представление SCHEMATA*

Имя столбца	SHOW-эквивалент	Замечание
CATALOG_NAME		Всегда NULL
SCHEMA_NAME	Database	
DEFAULT_CHARACTER_SET_NAME		
DEFAULT_COLLATION_NAME		
SQL_PATH		Всегда NULL

Замечание

Столбец DEFAULT_COLLATION_NAME добавлен в СУБД MySQL, начиная с версии 5.0.6.

В листинге 15.2 приведена результирующая таблица оператора DESCRIBE, примененного к представлению SCHEMATA.

Листинг 15.2. Список столбцов представления SCHEMATA

```
DESCRIBE INFORMATION_SCHEMA.SCHEMATA;
+----------------------------+--------------+------+-----+---------+
| Field                      | Type         | Null | Key | Default |
+----------------------------+--------------+------+-----+---------+
| CATALOG_NAME               | varchar(512) | YES  |     | NULL    |
| SCHEMA_NAME                | varchar(64)  | NO   |     |         |
| DEFAULT_CHARACTER_SET_NAME | varchar(64)  | NO   |     |         |
| DEFAULT_COLLATION_NAME     | varchar(64)  | NO   |     |         |
| SQL_PATH                   | varchar(512) | YES  |     | NULL    |
+----------------------------+--------------+------+-----+---------+
```

Каждая строка представления SCHEMATA соответствует одной базе данных. Столбцы представления имеют следующие значения:

- CATALOG_NAME — в СУБД MySQL это поле всегда содержит NULL, т. к. в MySQL отсутствует понятие каталога баз данных;
- SCHEMA_NAME — имя базы данных;
- DEFAULT_CHARACTER_SET_NAME — кодировка базы данных;
- DEFAULT_COLLATION_NAME — сортировка базы данных;
- SQL_PATH — как и поле CATALOG_NAME, в текущих версиях СУБД MySQL принимает значение NULL.

В листинге 15.3 приведена выборка из представления SCHEMATA с ограничением LIMIT.

Листинг 15.3. Выборка представления SCHEMATA

```
SELECT * FROM INFORMATION_SCHEMA.SCHEMATA LIMIT 2\G;
*************************** 1. row ***************************
              CATALOG_NAME: NULL
               SCHEMA_NAME: information_schema
DEFAULT_CHARACTER_SET_NAME: utf8
    DEFAULT_COLLATION_NAME: utf8_general_ci
                  SQL_PATH: NULL
```

```
*************************** 2. row ***************************
              CATALOG_NAME: NULL
               SCHEMA_NAME: mysql
DEFAULT_CHARACTER_SET_NAME: cp1251
    DEFAULT_COLLATION_NAME: cp1251_general_ci
                  SQL_PATH: NULL
```

15.2. Список таблиц

Представление TABLES содержит список таблиц и их характеристик. Информация в представлении дублирует результирующую таблицу оператора SHOW STATUS, но в отличие от данного оператора представление TABLES содержит информацию для всех таблиц на сервере, доступных текущему пользователю. Соответствие столбцов представления TABLES и результирующей таблицы, возвращаемой оператором SHOW TABLE STATUS, показано в табл. 15.3.

Таблица 15.3. *Представление TABLES*

Имя столбца	SHOW-эквивалент	Замечание
TABLE_CATALOG		Всегда NULL
TABLE_SCHEMA		
TABLE_NAME		
TABLE_TYPE		
ENGINE	Engine	Расширение MySQL
VERSION	Version	Расширение MySQL
ROW_FORMAT	Row_format	Расширение MySQL
TABLE_ROWS	Rows	Расширение MySQL
AVG_ROW_LENGTH	Avg_row_length	Расширение MySQL
DATA_LENGTH	Data_length	Расширение MySQL
MAX_DATA_LENGTH	Max_data_length	Расширение MySQL
INDEX_LENGTH	Index_length	Расширение MySQL
DATA_FREE	Data_free	Расширение MySQL
AUTO_INCREMENT	Auto_increment	Расширение MySQL
CREATE_TIME	Create_time	Расширение MySQL
UPDATE_TIME	Update_time	Расширение MySQL
CHECK_TIME	Check_time	Расширение MySQL
TABLE_COLLATION	Collation	Расширение MySQL

Таблица 15.3 *(окончание)*

Имя столбца	SHOW-эквивалент	Замечание
CHECKSUM	Checksum	Расширение MySQL
CREATE_OPTIONS	Create_options	Расширение MySQL
TABLE_COMMENT	Comment	Расширение MySQL

Замечание

Столбцы, помеченные как "Расширение MySQL", не являются стандартными и свойственны только СУБД MySQL.

В листинге 15.4 приведена результирующая таблица оператора DESCRIBE, примененного к представлению TABLES.

Листинг 15.4. Список столбцов представления TABLES

```
DESCRIBE INFORMATION_SCHEMA.TABLES;
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| TABLE_CATALOG   | varchar(512) | YES  |     | NULL    |       |
| TABLE_SCHEMA    | varchar(64)  | NO   |     |         |       |
| TABLE_NAME      | varchar(64)  | NO   |     |         |       |
| TABLE_TYPE      | varchar(64)  | NO   |     |         |       |
| ENGINE          | varchar(64)  | YES  |     | NULL    |       |
| VERSION         | bigint(21)   | YES  |     | NULL    |       |
| ROW_FORMAT      | varchar(10)  | YES  |     | NULL    |       |
| TABLE_ROWS      | bigint(21)   | YES  |     | NULL    |       |
| AVG_ROW_LENGTH  | bigint(21)   | YES  |     | NULL    |       |
| DATA_LENGTH     | bigint(21)   | YES  |     | NULL    |       |
| MAX_DATA_LENGTH | bigint(21)   | YES  |     | NULL    |       |
| INDEX_LENGTH    | bigint(21)   | YES  |     | NULL    |       |
| DATA_FREE       | bigint(21)   | YES  |     | NULL    |       |
| AUTO_INCREMENT  | bigint(21)   | YES  |     | NULL    |       |
| CREATE_TIME     | datetime     | YES  |     | NULL    |       |
| UPDATE_TIME     | datetime     | YES  |     | NULL    |       |
| CHECK_TIME      | datetime     | YES  |     | NULL    |       |
| TABLE_COLLATION | varchar(64)  | YES  |     | NULL    |       |
| CHECKSUM        | bigint(21)   | YES  |     | NULL    |       |
| CREATE_OPTIONS  | varchar(255) | YES  |     | NULL    |       |
| TABLE_COMMENT   | varchar(80)  | NO   |     |         |       |
+-----------------+--------------+------+-----+---------+-------+
```

Каждая строка представления TABLES соответствует одной таблице, доступной текущему пользователю. Столбцы представления имеют следующие значения:

- TABLE_CATALOG — в СУБД MySQL это поле всегда содержит NULL, т. к. в MySQL отсутствует понятие каталога баз данных;
- TABLE_SCHEMA — имя базы данных, в которой расположена таблица;
- TABLE_NAME — имя таблицы;
- TABLE_TYPE — тип таблицы, поле может принимать значения BASE TABLE, TEMPORARY или VIEW для обычных таблиц, временных таблиц и представлений соответственно;
- ENGINE — тип таблицы;
- VERSION — номер версии файла с расширением frm;
- ROW_FORMAT — формат хранения строки (Fixed, Dynamic или Compressed);
- TABLE_ROWS — количество записей в таблице;
- AVG_ROW_LENGTH — средняя длина записи;
- DATA_LENGTH — длина файла данных;
- MAX_DATA_LENGTH — максимальная длина файла данных. Для строк с фиксированной длиной — максимальное количество строк в таблице. Для строк с динамическим форматом — общее число байтов данных, которое может поместиться в таблице;
- INDEX_LENGTH — длина индексного файла;
- DATA_FREE — количество выделенных, но не используемых байтов;
- AUTO_INCREMENT — следующее значение AUTO_INCREMENT;
- CREATE_TIME — дата и время создания таблицы;
- UPDATE_TIME — дата и время обновления файла данных;
- CHECK_TIME — время, когда таблица проверялась в последний раз;
- TABLE_COLLATION — сортировка, которая используется в таблице;
- CHECKSUM — контрольная сумма (если включено ее автоматическое вычисление);
- CREATE_OPTIONS — дополнительные опции, использованные при создании таблицы оператором CREATE TABLE;
- TABLE_COMMENT — комментарий, введенный при создании таблицы.

В листинге 15.5 приведена выборка из представления TABLES для таблицы products.

Листинг 15.5. Выборка представления TABLE_PRIVILEGES

```
SELECT * FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_NAME = 'products'\G;
*************************** 1. row ***************************
  TABLE_CATALOG: NULL
   TABLE_SCHEMA: shop
     TABLE_NAME: products
     TABLE_TYPE: BASE TABLE
         ENGINE: MyISAM
        VERSION: 10
     ROW_FORMAT: Dynamic
     TABLE_ROWS: 31
 AVG_ROW_LENGTH: 123
    DATA_LENGTH: 3836
MAX_DATA_LENGTH: 281474976710655
   INDEX_LENGTH: 9216
      DATA_FREE: 0
 AUTO_INCREMENT: 32
    CREATE_TIME: 2005-07-23 20:13:16
    UPDATE_TIME: 2005-07-23 20:13:16
     CHECK_TIME: NULL
TABLE_COLLATION: cp1251_general_ci
       CHECKSUM: 1706424754
 CREATE_OPTIONS: checksum=1
  TABLE_COMMENT:
```

15.3. Список столбцов

Представление COLUMNS содержит информацию о доступных текущему пользователю столбцах во всех таблицах всех баз данных. Информация в представлении COLUMNS отличается от результирующей таблицы оператора SHOW COLUMNS, в частности, последний выводит информацию строго для определенной таблицы и содержит меньшее количество столбцов. Соответствие столбцов представления COLUMNS и результирующей таблицы, возвращаемой оператором SHOW COLUMNS, приведено в табл. 15.4.

Замечание

Столбцы, помеченные "Расширение MySQL", не являются стандартными и свойственны только СУБД MySQL.

В листинге 15.6 приведена результирующая таблица оператора DESCRIBE, примененного к представлению COLUMNS.

Таблица 15.4. *Представление* `COLUMNS`

Имя столбца	SHOW-эквивалент	Замечание
TABLE_CATALOG		Всегда NULL
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME	Field	
ORDINAL_POSITION		
COLUMN_DEFAULT	Default	
IS_NULLABLE	Null	
DATA_TYPE	Type	
CHARACTER_MAXIMUM_LENGTH	Type	
CHARACTER_OCTET_LENGTH		
NUMERIC_PRECISION	Type	
NUMERIC_SCALE	Type	
CHARACTER_SET_NAME		
COLLATION_NAME	Collation	
COLUMN_TYPE		
COLUMN_KEY	Key	Расширение MySQL
EXTRA	Extra	Расширение MySQL
PRIVILEGES		
COLUMN_COMMENT	Comment	Расширение MySQL

Листинг 15.6. Список столбцов представления `COLUMNS`

```
DESCRIBE INFORMATION_SCHEMA.COLUMNS;
+-------------------------+--------------+------+-----+---------+------+
| Field                   | Type         | Null | Key | Default | Extra|
+-------------------------+--------------+------+-----+---------+------+
| TABLE_CATALOG           | varchar(512) | YES  |     | NULL    |      |
| TABLE_SCHEMA            | varchar(64)  | NO   |     |         |      |
| TABLE_NAME              | varchar(64)  | NO   |     |         |      |
| COLUMN_NAME             | varchar(64)  | NO   |     |         |      |
| ORDINAL_POSITION        | bigint(21)   | NO   |     | 0       |      |
| COLUMN_DEFAULT          | varchar(64)  | YES  |     | NULL    |      |
| IS_NULLABLE             | varchar(3)   | NO   |     |         |      |
| DATA_TYPE               | varchar(64)  | NO   |     |         |      |
```

```
| CHARACTER_MAXIMUM_LENGTH | bigint(21)   | YES  |     | NULL    |     |
| CHARACTER_OCTET_LENGTH   | bigint(21)   | YES  |     | NULL    |     |
| NUMERIC_PRECISION        | bigint(21)   | YES  |     | NULL    |     |
| NUMERIC_SCALE            | bigint(21)   | YES  |     | NULL    |     |
| CHARACTER_SET_NAME       | varchar(64)  | YES  |     | NULL    |     |
| COLLATION_NAME           | varchar(64)  | YES  |     | NULL    |     |
| COLUMN_TYPE              | longtext     | NO   |     |         |     |
| COLUMN_KEY               | varchar(3)   | NO   |     |         |     |
| EXTRA                    | varchar(20)  | NO   |     |         |     |
| PRIVILEGES               | varchar(80)  | NO   |     |         |     |
| COLUMN_COMMENT           | varchar(255) | NO   |     |         |     |
+--------------------------+--------------+------+-----+---------+-----+
```

Каждая строка представления `COLUMNS` соответствует одному из столбцов. Причем представление содержит информацию обо всех столбцах во всех таблицах во всех базах данных, к которым имеет доступ текущий пользователь. Столбцы представления `COLUMNS` имеют следующие значения:

- `TABLE_CATALOG` — данное поле всегда принимает значение `NULL`, т. к. СУБД MySQL не имеет каталогов баз данных;
- `TABLE_SCHEMA` — имя базы данных, содержащей таблицу, в которой находится столбец с именем `COLUMN_NAME`;
- `TABLE_NAME` — имя таблицы, содержащей столбец с именем `COLUMN_NAME`;
- `COLUMN_NAME` — имя столбца;
- `ORDINAL_POSITION` — порядковый номер столбца в таблице, которой он принадлежит;
- `COLUMN_DEFAULT` — значение столбца по умолчанию;
- `IS_NULLABLE` — поле отображает, может ли столбец принимать значение `NULL`. В случае положительного ответа принимает значение `YES`, в противном случае — `NO`;
- `DATA_TYPE` — тип столбца (только ключевое слово);
- `CHARACTER_MAXIMUM_LENGTH` — максимальное число символов, отводимых под значение в столбце (для многобайтных кодировок это значение не совпадает со значением `CHARACTER_OCTET_LENGTH`);
- `CHARACTER_OCTET_LENGTH` — максимальное число байтов, отводимых под значение в столбце (для многобайтных кодировок это значение не совпадает со значением `CHARACTER_MAXIMUM_LENGTH`);
- `NUMERIC_PRECISION` — количество позиций, отводимых под число, если поле не является числовым, принимает значение `NULL`;
- `NUMERIC_SCALE` — количество позиций, отводимых под дробную часть числа;

- CHARACTER_SET_NAME — для текстовых столбцов отображается кодировка, для нетекстовых столбцов — NULL;
- COLLATION_NAME — для текстовых столбцов отображается сортировка, для нетекстовых столбцов — NULL;
- COLUMN_TYPE — полный тип столбца;
- COLUMN_KEY — поле отражает, является ли столбец индексированным или нет;
- EXTRA — дополнительная информация о столбце, например, снабжено ли поле атрибутом AUTO_INCREMENT;
- PRIVILEGES — привилегии, доступные текущему пользователю по отношению к данному столбцу;
- COLUMN_COMMENT — комментарий к столбцу.

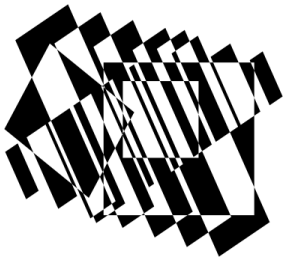
Выборка представления INFORMATION_SCHEMA.COLUMNS без ограничений (особенно для учетной записи root) приводит к выводу огромного числа информации, поэтому оператор SELECT, осуществляющий выборку из данного представления, разумно снабжать жестким WHERE-условием (листинг 15.7).

Листинг 15.7. Выборка представления INFORMATION_SCHEMA.COLUMNS

```
SELECT * FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 'catalogs'\G;
*************************** 1. row ***************************
           TABLE_CATALOG: NULL
            TABLE_SCHEMA: shop
              TABLE_NAME: catalogs
             COLUMN_NAME: id_catalog
        ORDINAL_POSITION: 1
          COLUMN_DEFAULT: NULL
             IS_NULLABLE: NO
               DATA_TYPE: int
CHARACTER_MAXIMUM_LENGTH: NULL
  CHARACTER_OCTET_LENGTH: NULL
       NUMERIC_PRECISION: 11
           NUMERIC_SCALE: 0
      CHARACTER_SET_NAME: NULL
          COLLATION_NAME: NULL
             COLUMN_TYPE: int(11)
              COLUMN_KEY: PRI
                   EXTRA: auto_increment
              PRIVILEGES: select,insert,update,references
          COLUMN_COMMENT:
```

```
*************************** 2. row ***************************
           TABLE_CATALOG: NULL
            TABLE_SCHEMA: shop
              TABLE_NAME: catalogs
             COLUMN_NAME: name
        ORDINAL_POSITION: 2
          COLUMN_DEFAULT:
             IS_NULLABLE: NO
               DATA_TYPE: tinytext
CHARACTER_MAXIMUM_LENGTH: 255
  CHARACTER_OCTET_LENGTH: 255
       NUMERIC_PRECISION: NULL
           NUMERIC_SCALE: NULL
      CHARACTER_SET_NAME: cp1251
          COLLATION_NAME: cp1251_general_ci
             COLUMN_TYPE: tinytext
              COLUMN_KEY:
                   EXTRA:
              PRIVILEGES: select,insert,update,references
          COLUMN_COMMENT:
```

ГЛАВА 16



Планировщик заданий

Планировщик заданий позволяет выполнять один или несколько операторов SQL в строго определенное время. Это очень похоже на триггеры *(см. главу 13)*, только в качестве запускающего события выступает не выполнение операторов `INSERT`, `DELETE` и `UPDATE`, а наступление определенной даты или времени суток.

Замечание

В литературе часто вместо термина "планировщик заданий" используется термин "временные триггеры" (temporal triggers).

Замечание

Планировщик заданий и связанные с ним операторы введены в MySQL, начиная с версии 5.1.6. Для более ранних версий запуск SQL-инструкций необходимо было производить из внешней программы, запускаемой под управлением UNIX-демона cron (планировщик задач в операционной системе UNIX).

Планировщик заданий не является частью стандарта SQL и является расширением СУБД MySQL.

16.1. Управление планировщиком задач

Задания на сервере MySQL выполняются специальным резидентным потоком, который стартует и прекращает работу вместе с сервером.

Замечание

Такие же резидентные потоки ответственны за процесс репликации *(см. разд. 11.8)*, один поток, отправляющий новые записи бинарного журнала, выполняется на главном сервере и два (осуществляющих получение и разбор записей бинарного журнала) выполняются на подчиненном сервере.

По умолчанию планировщик заданий выключен и его процесс не видно в списке процессов, возвращаемых оператором SHOW PROCESSLIST или представлением PROCESSLIST информационной схемы. За состояние планировщика заданий несет ответственность системная переменная event_scheduler, которая может принимать три значения:

- OFF — планировщик заданий выключен;
- ON — планировщик заданий включен;
- DISABLED — планировщик заданий дезактивирован и не может быть запущен.

Замечание

Подробнее системные переменные рассматриваются в *разд. 9.3* и *приложении 1*.

Определить текущее состояние системной переменной event_scheduler можно при помощи оператора SHOW VARIABLES (листинг 16.1).

Листинг 16.1. Просмотр значения системной переменной event_scheduler

```
SHOW VARIABLES LIKE 'event_scheduler';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| event_scheduler | OFF   |
+-----------------+-------+
```

Для того чтобы включить планировщик заданий, необходимо изменить значение системной переменной event_scheduler с OFF на ON. Вместо констант OFF и ON можно также использовать числовые значения 0 и 1. В листинге 16.2 все четыре строки эквиваленты и приводят к запуску планировщика заданий.

Замечание

Ключевое слово GLOBAL в листинге 16.2 является обязательным, т. к. установить значение переменной event_scheduler для одного соединения нельзя — планировщик либо включен для всего сервера, либо выключен.

Листинг 16.2. Запуск планировщика заданий

```
SET GLOBAL event_scheduler = ON;
SET @@global.event_scheduler = ON;
SET GLOBAL event_scheduler = 1;
SET @@global.event_scheduler = 1;
```

```
SHOW VARIABLES LIKE 'event_scheduler';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| event_scheduler | ON    |
+-----------------+-------+
```

Теперь, выполнив оператор `SHOW PROCESSLIST`, в результирующей таблице можно увидеть процесс, соответствующий планировщику заданий. В поле `User` данного процесса указывается значение `event_scheduler` (листинг 16.3).

Листинг 16.3. Планировщик заданий в процессах

```
SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 1
   User: root
   Host: localhost:2242
     db: NULL
Command: Query
   Time: 0
  State: NULL
   Info: SHOW PROCESSLIST
*************************** 2. row ***************************
     Id: 2
   User: event_scheduler
   Host: localhost
     db: NULL
Command: Daemon
   Time: 396
  State: Waiting for next activation
   Info: NULL
```

Если системная переменная `event_scheduler` принимает значение `DISABLED`, изменить ее значение нельзя во время работы сервера MySQL. Точно так же невозможно присвоить значение `DISABLED` переменной `event_scheduler` при помощи оператора `SET` — сервер MySQL возвращает ошибку в ответ на подобную попытку (листинг 16.4).

Листинг 16.4. Неудачная попытка приводить значение `DISABLED` переменной `event_scheduler`

```
SET GLOBAL event_scheduler = DISABLED;
ERROR 1231 (42000): Variable 'event_scheduler' can't be set to the value of
'DISABLED'
```

Дезактивировать планировщик заданий можно либо при помощи параметра `--event-scheduler=DISABLED` при запуске MySQL-сервера, либо поместив в конфигурационный файл my.ini или my.cnf директиву `event-scheduler=DISABLED`.

16.2. Размещение задания

Поместить новое задание в планировщике задач можно при помощи оператора `CREATE EVENT`, который имеет следующий синтаксис:

```
CREATE EVENT [IF NOT EXISTS] event_name
    ON SCHEDULE schedule
    [ON COMPLETION [NOT] PRESERVE]
    [ENABLE | DISABLE]
    [COMMENT 'comment']
    DO sql_statement;
```

Оператор создает новое задание с именем *event_name*, которое должно быть уникально в пределах всего сервера. Если такое задание существует, сервер возвращает ошибку. Впрочем, возникновение ошибочной ситуации можно избежать, если указать необязательную конструкцию `IF NOT EXISTS` — при ее использовании оператор просто ничего не создает, если задание с таким именем уже существует.

Замечание

Начиная с версии MySQL 5.1.8, имена заданий не зависят от регистра.

Конструкция `ON SCHEDULE` задает, когда и как часто будет выполняться задание. Определение *schedule* имеет следующий синтаксис:

```
AT timestamp [+ INTERVAL interval]
| EVERY interval [STARTS timestamp] [ENDS timestamp]
```

Задания могут быть единовременными, т. е. выполняющимися один раз, и периодическими, срабатывающими через регулярные промежутки времени. Для периодических заданий можно указать дату и время начала и конца периода выполнения. Если эти параметры не указываются, то задание выполняется бессрочно, начиная с момента создания.

Время выполнения задания задается при помощи ключевого слова `AT`, после которого помещается время, и если требуется, к нему прибавляется интервал, который более подробно описывается в *разд. 6.2.1*. В листинге 16.5 приводится пример простейшего задания `drop_catalog`, который очистит таблицу `catalog` базы данных `test` через час после создания задания.

Замечание

Характерной особенностью заданий является тот факт, что в теле задания после ключевого слова DO необходимо использовать только расширенные имена, включающие имя базы данных и таблицы.

Листинг 16.5. Создание задания

```
CREATE EVENT drop_catalog
ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
DO
DROP TABLE test.catalog;
```

Вместо CURRENT_TIMESTAMP можно использовать фиксированное время так, как это представлено в листинге 16.6.

Листинг 16.6. Использование фиксированного времени

```
CREATE EVENT drop_catalog
ON SCHEDULE AT '2007-02-28 10:20:00'
DO
DROP TABLE test.catalog;
```

Задание не может быть создано со стартовым временем в прошлом. Кроме того, задания, которые назначены на время после UNIX-эпохи (позднее 2037 года) не выполняются. В листинге 16.7 демонстрируется попытка создания задания в прошлом.

Листинг 16.7. Задание в прошлом

```
SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2007-02-15 01:31:23 |
+---------------------+
CREATE EVENT drop_catalog
ON SCHEDULE AT '2007-02-15 01:00:00'
DO
DROP TABLE test.catalog;
ERROR 1531 (HY000): Activation (AT) time is in the past
```

Интервалы можно комбинировать, например, для задания даты, которая наступит спустя три недели и два дня, можно использовать конструкцию, представленную в листинге 16.8.

Листинг 16.8. Комбинированные интервалы

```
CREATE EVENT drop_catalog
ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 3 WEEK + INTERVAL 2 DAY
DO
DROP TABLE test.catalog;
```

Для создания периодических заданий используется ключевое слово `EVERY`, после которого указывается период выполнения задания. В листинге 16.9 приводится пример создания задания, которое каждую неделю удаляет из таблицы `test.catalog` все записи.

Замечание

При создании задания допускается использование либо ключевого слова `AT`, либо `EVERY` — совместное их использование не допускается.

Листинг 16.9. Периодическое задание

```
CREATE EVENT drop_catalog
ON SCHEDULE EVERY 1 WEEK
DO
DROP TABLE test.catalog;
```

После ключевого слова `EVERY` не допускается указание ключевого слова `INTERVAL`, однако разрешено использование сложных форматов времени *(см. разд. 6.2.1)*. В листинге 16.10 приводится задание, которое выполняется каждые пять с половиной минут.

Листинг 16.10. Задание, выполняющееся каждые пять с половиной минут

```
CREATE EVENT drop_catalog
ON SCHEDULE EVERY '5:30' MINUTE_SECOND
DO
DROP TABLE test.catalog;
```

После ключевого слова `EVERY` могут быть использованы ключевые слова `STARTS` и `ENDS`, которые задают начало и конец периодического задания. В отличие от `EVERY` во временных значениях, следующих за конструкциями `STARTS` и `ENDS`, допускается использование ключевого слова `INTERVAL`. В листинге 16.11 приводится пример ежечасного задания, которое начинает выполняться через три недели от текущего момента, а заканчивается через два месяца с начала выполнения.

Замечание

Не обязательно использовать одновременно STARTS и ENDS, каждое из этих ключевых слов может быть использовано без другого. Если не указывается STARTS, считается, что задание начинает выполняться с момента создания задания, если не используется ENDS, задание выполняется до тех пор, пока оно не будет удалено вручную.

Листинг 16.11. Использование ключевых слов STARTS и ENDS

```
CREATE EVENT drop_catalog
ON SCHEDULE EVERY 1 HOUR
STARTS CURRENT_TIMESTAMP + INTERVAL 3 WEEK
ENDS CURRENT_TIMESTAMP + INTERVAL 3 WEEK + INTERVAL 2 MONTH
DO
DROP TABLE test.catalog;
```

Точно так же, как и триггеры, задания представляют собой своеобразные хранимые процедуры, в блоке BEGIN...END которых содержатся операторы, предназначенные для выполнения. В листинге 16.12 приводится пример задания, которое удаляет записи из таблиц catalog и product базы данных test.

Замечание

Синтаксис блока BEGIN...END более подробно рассматривается в *главе 12*.

Листинг 16.12. Использование блока BEGIN...END

```
DELIMITER //
CREATE EVENT drop_catalog
ON SCHEDULE EVERY 1 HOUR
DO
BEGIN
  DROP TABLE test.catalog;
  DROP TABLE test.product;
END//
```

В отличие от хранимых процедур в задание невозможно передать параметры, однако это ограничение можно обойти, вызывая хранимую процедуру из задания (листинг 16.13).

Листинг 16.13. Вызов хранимой процедуры из задания

```
CREATE EVENT call_proc
    ON SCHEDULE
      AT CURRENT_TIMESTAMP + 1 DAY
    DO CALL myproc(5, 27);
```

По умолчанию задание сразу после создания является активным и, если наступает время выполнения, срабатывает. Можно изменить такое поведение, используя ключевое слово DISABLE (листинг 16.14).

Листинг 16.14. Использование ключевого слова DISABLE

```
CREATE EVENT drop_catalog
ON SCHEDULE EVERY '5:30' MINUTE_SECOND
DISABLE
DO
DROP TABLE test.catalog;
```

До тех пор, пока статус задания не будет изменен на ENABLE (например, при помощи оператора ALTER EVENT, который рассматривается в *разд. 16.4*), задание не будет выполняться. При создании нового задания статус ENABLE обычно не указывают, т. к. этот статус присваивается по умолчанию.

При помощи ключевого слова COMMENT можно задать комментарий к заданию (листинг 16.15). Данный комментарий можно прочитать в результирующей таблице оператора SHOW CREATE EVENT *(см. разд. 16.4)*.

Листинг 16.15. Использование ключевого слова COMMENT

```
CREATE EVENT drop_catalog
ON SCHEDULE AT '2007-12-31 23:59:59'
COMMENT 'Очистка каталога в конце года'
DO
DROP TABLE test.catalog;
```

16.3. Удаление задания

Обычно задание удаляется сразу по истечении его срока действия, однако такое поведение можно исправить, указав в определении задания конструкцию ON COMPLETION PRESERVE (листинг 16.16).

Листинг 16.16. Использование конструкции ON COMPLETION PRESERVE

```
CREATE EVENT drop_catalog
ON SCHEDULE AT '2007-02-15 01:00:00'
ON COMPLETION PRESERVE
DO
DROP TABLE test.catalog;
```

Теперь, даже после того, как задание будет выполнено и не сработает больше никогда, оно не будет удаляться. Это может быть удобным, если задание является сложным, и в дальнейшем потребуется его повторное использование — достаточно будет изменить время выполнения задания (например, при помощи оператора `ALTER EVENT`, который рассматривается в *разд. 16.4*).

Поведение обратное, рассмотренному выше, можно задать при помощи конструкции `ON COMPLETION NOT PRESERVE` (листинг 16.17). Однако эту директиву часто опускают, т. к. задания уничтожаются по истечении срока действия по умолчанию, без явного указания конструкции `ON COMPLETION NOT PRESERVE`.

Листинг 16.17. Использование конструкции `ON COMPLETION NOT PRESERVE`

```
CREATE EVENT drop_catalog
ON SCHEDULE AT '2007-02-15 01:00:00'
ON COMPLETION NOT PRESERVE
DO
DROP TABLE test.catalog;
```

Выше рассмотрены способы влияния на неявное удаление задания по истечении его срока действия. Однако существует явный способ уничтожения задания при помощи оператора `DROP EVENT`, который имеет следующий синтаксис:

```
DROP EVENT [IF EXISTS] event_name
```

Оператор уничтожает задание с именем *`event_name`*. Если такое задание не существует, возвращается соответствующая ошибка. Генерацию ошибки можно подавить, если использовать ключевое слово `IF EXISTS`. В листинге 16.18 приводится пример уничтожения события `drop_catalog`.

Листинг 16.18. Использование оператора `DROP EVENT`

```
DROP EVENT drop_catalog;
```

16.4. Редактирование задания

Просмотреть структуру задания можно при помощи оператора `SHOW CREATE EVENT` (листинг 16.19).

Листинг 16.19. Использование оператора `SHOW CREATE EVENT`

```
SHOW CREATE EVENT drop_catalog\G
*************************** 1. row ***************************
       Event: drop_catalog
```

```
    sql_mode:
Create Event: CREATE EVENT `drop_catalog`
ON SCHEDULE AT '2007-12-31 20:59:59'
ON COMPLETION
NOT PRESERVE
ENABLE
COMMENT 'Очистка каталога в конце года'
DO DROP TABLE test.catalog
```

Результирующая таблица оператора `SHOW CREATE EVENT` имеет три поля:

- `Event` — название задания;
- `sql_mode` — SQL-режим, в котором выполняется задание;
- `Create Event` — оператор `CREATE EVENT`, который воссоздает событие.

Уже существующее задание можно поправить при помощи оператора `ALTER EVENT`, который имеет следующий синтаксис:

```
ALTER EVENT event_name
    [ON SCHEDULE schedule]
    [RENAME TO new_event_name]
    [ON COMPLETION [NOT] PRESERVE]
    [ENABLE | DISABLE]
    [COMMENT 'comment']
    [DO sql_statement]
```

Оператор `ALTER EVENT` редактирует один или более параметров события *event_name*. Синтаксис конструкций `ON SCHEDULE`, `ON COMPLETION`, `ENABLE`/`DISABLE`, `COMMENT` и `DO` аналогичен соответствующим конструкциям оператора `CREATE EVENT`.

В листинге 16.20 событие `drop_catalog` сначала создается при помощи оператора `CREATE EVENT`, а затем редактируется при помощи `ALTER EVENT`.

Замечание

Оператор `ALTER EVENT` допускает использование в теле задания `DO` вложенных операторов `ALTER EVENT` — однако выполнение такого задания заканчивается неудачей.

Листинг 16.20. Использование оператора `ALTER EVENT`

```
CREATE EVENT IF NOT EXISTS drop_catalog
ON SCHEDULE AT '2007-02-15 01:00:00'
DO
DROP TABLE test.catalog;
```

```
ALTER EVENT drop_catalog
ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
```

Переименовать событие можно, воспользовавшись ключевым словом RENAME TO. В листинге 16.21 задание drop_catalog переименовывается в DropCatalog.

Листинг 16.21. Переименование задания

```
ALTER EVENT drop_catalog
RENAME TO DropCatalog;
```

Ключевое слово RENAME TO допускает не только переименование события, но также перемещение его из одного каталога в другой. В листинге 16.22 приводится пример переноса события drop_catalog из базы данных test в базу данных shop.

Листинг 16.22. Перенос каталога из базы данных test в базу данных shop

```
ALTER EVENT test.drop_catalog
RENAME TO shop.drop_catalog;
DO
DROP TABLE shop.catalog;
```

16.5. Список заданий

Просмотреть список созданных ранее заданий можно при помощи оператора SHOW EVENTS, который имеет следующий синтаксис:

SHOW EVENTS [FROM *schema_name*] [LIKE *pattern*]

Оператор позволяет указать необязательное ключевое слово FROM, задающее базу данных, из которой следует извлечь события. Еще более сузить результаты поиска можно при помощи ключевого слова LIKE, которое имеет тот же синтаксис, что и ключевое слово LIKE в операторе SELECT *(см. главу 5)*.

В листинге 16.23 приводится пример использования оператора SHOW EVENTS.

Листинг 16.23. Использование оператора SHOW EVENTS

```
SHOW EVENTS FROM test\G
*************************** 1. row ***************************
            Db: test
          Name: drop_catalog
       Definer: root@localhost
```

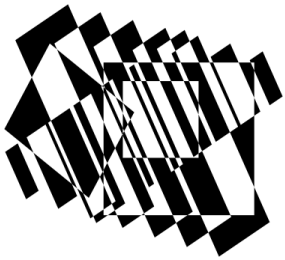
```
          Type: ONE TIME
    Execute at: 2007-12-31 20:59:59
Interval value: NULL
Interval field: NULL
        Starts: NULL
          Ends: NULL
        Status: ENABLED
```

Как видно из листинга 16.23, база данных `test` содержит одно задание `drop_catalog`. Результирующая таблица оператора `SHOW EVENTS` имеет следующие столбцы:

- `Db` — имя базы данных, к которой привязано задание;
- `Name` — имя задания;
- `Definer` — пользователь MySQL, который создал задание;
- `Type` — данное поле принимает два значения: `ONE TIME` для единовременных заданий и `RECURRING` для периодических;
- `Execute at` — время выполнения единовременного задания в формате `DATETIME`, для периодических заданий поле принимает значение `NULL`;
- `Interval value` — количество временных единиц для интервала периодического задания, для единовременных заданий поле принимает значение `NULL`;
- `Interval field` — единицы измерения интервала периодического задания, для единовременных заданий поле принимает значение `NULL`;
- `Starts` — время начала выполнения периодического задания в формате `DATETIME`, для единовременных заданий поле принимает значение `NULL`;
- `Ends` — время окончания выполнения периодического задания в формате `DATETIME`, для единовременных заданий поле принимает значение `NULL`;
- `Status` — статус задания, может принимать два значения: `ENABLED`, если задание активно и выполняется, и `DISABLED`, если выполнение задания приостановлено.

Помимо оператора `SHOW EVENTS` для получения информации о событиях можно воспользоваться представлением `EVENTS` информационной схемы *(см. главу 15)*.

ГЛАВА 17



Сегментирование

Сегментирование позволяет разбить таблицы на части и расположить эти части в различных областях файловой системы. Это может быть удобно для увеличения производительности. Не секрет, что доступ к файлам, расположенным на нескольких дисках, более высок (особенно если файлы находятся в начале диска). Да и оперировать несколькими небольшими файлами получается гораздо быстрее, чем одним объемным. Кроме того, ряд файловых систем не поддерживает файлы больших объемов, например, больше 2 Гбайт — сегментирование позволяет обойти это ограничение.

Замечание

Процесс разбиения таблицы на несколько разделов называют *сегментированием* или *секционированием*. Данный прием впервые был использован в СУБД Oracle. Сегментирование введено в MySQL, начиная с версии 5.1.

Различают *горизонтальное сегментирование*, при котором разные строки находятся в разных физических разделах, и *вертикальное сегментирование*, при котором в разных разделах находятся столбцы таблицы. В настоящий момент СУБД MySQL поддерживает только горизонтальное сегментирование.

17.1. Включено ли сегментирование?

Проверить, поддерживает ли текущий дистрибутив сегментирование и включено ли оно, можно, обратившись к системной переменной `have_partitioning` (листинг 17.1).

Листинг 17.1. Проверка, включено ли сегментирование

```
SHOW VARIABLES LIKE '%partition%';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| have_partitioning | YES   |
+-------------------+-------+
```

Если переменная `have_partitioning` отсутствует, или ее значение отлично от `YES`, это означает, что дистрибутив не поддерживает сегментирование.

17.2. Создание сегментированной таблицы

Сегментированные таблицы, как и любые другие таблицы, создаются при помощи оператора `CREATE TABLE`, параметры сегментирования указываются в конце таблицы в секции параметров.

В сегментации может участвовать большинство типов таблиц (кроме MERGE и CSV), поддерживаемых СУБД MySQL, однако все сегменты должны быть одного типа, т. е. не допускается, чтобы одна часть таблицы была типа MyISAM, а другая, например, InnoDB.

Указать тип, как и для обычных таблиц, можно при помощи ключевого слова `ENGINE`. Однако следует иметь в виду, что ключевое слово `ENGINE` должно предварять все ключевые слова, связанные с сегментированием. В листинге 17.2 приводится пример создания сегментированной таблицы хэш-типа, состоящей из шести частей.

Замечание

Типы сегментирования подробно рассматриваются в *разд. 17.3*.

Листинг 17.2. Создание сегментированной таблицы

```
CREATE TABLE tbl (id INT, amount DECIMAL(7,2), tr_date DATE)
    ENGINE=MYISAM
    PARTITION BY HASH( MONTH(tr_date) )
    PARTITIONS 6;
```

Как видно из листинга 17.2, тип таблицы задается до конструкций `PARTITION BY` и `PARTITIONS`.

Разбиение таблицы на части происходит при помощи *функции сегментирования*, в листинге 17.2 в качестве такой функции выступает функция `HASH()`.

В качестве аргумента любая функция принимает либо целочисленный столбец, либо функцию, которая возвращает целочисленное значение.

Замечание

Максимальное количество частей, на которые может быть разбита таблица, не должно превышать 1024.

Ключевое слово `PARTITIONS` в листинге 17.2 задает количество частей, на которые будет разбита таблица.

17.3. Типы сегментирования таблиц

Различают несколько типов сегментации, которые определяются функцией сегментирования, указываемой после ключевого слова `PARTITION BY`. Типы сегментации включают:

- `RANGE` — линейная сегментация или сегментация по диапазону. Попадание записи в тот или иной раздел определяется тем, в какой интервал попадает аргумент функции сегментирования `RANGE()`. В качестве примера диапазонов могут служить следующие интервалы 1—7, 8—16, 17—24, 25—32;
- `LIST` — списочная сегментация. Данный тип сегментации похож на линейную сегментацию, только выбор сегмента определяется дискретными списками. В качестве примера списков могут служить следующие наборы: (1, 3, 5), (10, 2, 8), (4, 6, 7, 9);
- `HASH` — хэш-сегментация. В случае хэш-сегментации записи разделяются между разделами равномерно, пользователю остается задать лишь количество разделов, на которые должна быть разбита таблица;
- `KEY` — индексная сегментация. Данный тип сегментации может использовать индекс по одному или нескольким столбцам и схож с хэш-сегментацией — пользователь задает лишь количество разделов.

17.3.1. Линейная сегментация *RANGE*

При использовании линейной сегментации записи таблицы делятся на части в зависимости от того, в какой диапазон попадает критерий разбиения таблицы. Пусть имеется таблица `catalogs`, которая содержит два столбца:

- `id_catalog` — играющий роль первичного ключа;
- `name` — название элемента каталога.

Разобьем таблицу `catalogs` на три части по первичному ключу `id_catalog`. Те записи, для которых первичный ключ принимает значения от 1 до 3, будут

попадать в первый раздел (MySQL нумерует его нулевым), от 4 до 6 — во второй, а от 7 до 9 — в третий (листинг 17.3).

Замечание

Интервалы, должны граничить друг с другом, но не должны пересекаться.

Листинг 17.3. Линейная сегментация

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  PRIMARY KEY (id_catalog)
)
ENGINE=MYISAM
PARTITION BY RANGE (id_catalog) (
    PARTITION p0 VALUES LESS THAN (4),
    PARTITION p1 VALUES LESS THAN (7),
    PARTITION p2 VALUES LESS THAN (10)
);
INSERT INTO catalogs VALUES (1, 'Процессоры');
INSERT INTO catalogs VALUES (2, 'Материнские платы');
INSERT INTO catalogs VALUES (3, 'Видеоадаптеры');
INSERT INTO catalogs VALUES (4, 'Жесткие диски');
INSERT INTO catalogs VALUES (5, 'Оперативная память');
INSERT INTO catalogs VALUES (6, 'Периферия');
INSERT INTO catalogs VALUES (7, 'Карты оплаты');
INSERT INTO catalogs VALUES (8, 'Сетевое оборудование');
```

Как видно из листинга 17.3, при помощи функции `RANGE()`, принимающей целочисленное поле `id_catalog`, таблица разбивается на три части: `p0`, `p1` и `p3`. В качестве аргумента функции сегментирования `RANGE()` нельзя использовать поле `name` — допускаются к применению только целочисленные положительные значения. Именование раздела осуществляется при помощи ключевого слова `PARTITION`, после которого следует название раздела (`p0`, `p1` и `p2`), а также диапазон, который задается при помощи конструкции `VALUES LESS THAN`. Следует отметить, что диапазоны объявляются от меньшего к большему, т. е. не допускается, чтобы раздел со значением `VALUES LESS THAN`, равным 10, предшествовал 4 или 7.

Замечание

Вместо `p0`, `p1` и `p2` можно использовать любые другие уникальные имена, которые бы отличались друг от друга. Имена разделов не зависят от регистра, поэтому имя `MyPartition` и `mypartition` являются одинаковыми.

Теперь, если заглянуть в подкаталог, соответствующий базе данных в каталоге данных, можно убедиться, что СУБД MySQL создала для таблицы `catalogs` восемь файлов:

- catalogs.frm — файл, содержащий структуру таблицы, информацию об именах и типах столбцов;
- catalogs.par — файл, содержащий описание и структуру разделов, на которые разбита таблица `catalogs`;
- catalogs#P#p0.MYD — файл данных первого раздела `p0`;
- catalogs#P#p0.MYI — файл индексов первого раздела `p0`;
- catalogs#P#p1.MYD — файл данных второго раздела `p1`;
- catalogs#P#p1.MYI — файл индексов второго раздела `p1`;
- catalogs#P#p2.MYD — файл данных третьего раздела `p2`;
- catalogs#P#p2.MYI — файл индексов третьего раздела `p2`.

Как видно из листинга 17.3, в таблицу `catalogs` заносится восемь записей. Записи с первичными ключами `id_catalog`, равными 1, 2 и 3, попадают в раздел `p0`, записи с `id_catalog`, равными 4, 5 и 6, попадают в раздел `p1`, а записи с `id_catalog`, равными 7 и 8, попадают в раздел `p2`.

Если теперь добавить в таблицу `catalogs` еще две записи (листинг 17.4), то первая запись будет успешно вставлена, в то время как вставка второй записи закончится неудачей и выводом сообщения об ошибке "Table has no partition for value 10" ("Таблица не имеет раздела для значения 10").

Листинг 17.4. Вставка дополнительных записей в таблицу `catalogs`

```
INSERT INTO catalogs VALUES (9, 'Источники бесперебойного питания');
INSERT INTO catalogs VALUES (10, 'Программное обеспечение');
ERROR 1513 (HY000): Table has no partition for value 10
```

Такое поведение обусловлено тем, что таблица `catalogs` имеет диапазоны для записей с первичным ключом `id_catalog` только лишь до значения 9 и, получив значение 10, сервер MySQL не знает, в какой из разделов следует поместить новую запись. Для того чтобы предотвратить возникновение такой ситуации, можно использовать конструкцию `VALUES LESS THAN MAXVALUE` (листинг 17.5).

Листинг 17.5. Использование конструкции `VALUES LESS THAN MAXVALUE`

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
```

```
  name TINYTEXT NOT NULL,
  PRIMARY KEY (id_catalog)
)
ENGINE=MYISAM
PARTITION BY RANGE (id_catalog) (
    PARTITION p0 VALUES LESS THAN (4),
    PARTITION p1 VALUES LESS THAN (7),
    PARTITION p2 VALUES LESS THAN (10),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
INSERT INTO catalogs VALUES (1, 'Процессоры');
INSERT INTO catalogs VALUES (2, 'Материнские платы');
INSERT INTO catalogs VALUES (3, 'Видеоадаптеры');
INSERT INTO catalogs VALUES (4, 'Жесткие диски');
INSERT INTO catalogs VALUES (5, 'Оперативная память');
INSERT INTO catalogs VALUES (6, 'Периферия');
INSERT INTO catalogs VALUES (7, 'Карты оплаты');
INSERT INTO catalogs VALUES (8, 'Сетевое оборудование');
INSERT INTO catalogs VALUES (9, 'Источники бесперебойного питания');
INSERT INTO catalogs VALUES (10, 'Программное обеспечение');
```

Ключевое слово `MAXVALUE` представляет максимально возможное целое число. Теперь все записи, для которых значение поля `id_catalog` превышает значение 9, будут попадать в раздел `p3`. Если в разделе накопится слишком большое количество записей по сравнению с другими разделами, таблицу можно всегда реорганизовать при помощи оператора `ALTER TABLE` *(см. разд. 17.6—17.8)*.

В качестве аргумента функции сегментирования `RANGE()` может выступать не только целочисленный столбец таблицы, но также и любое выражение, возвращающее целые положительные числа. Очень часто в качестве таких значений выступают функции даты и времени *(см. разд. 6.2)*. В листинге 17.6 приводится таблица `tbl`, которая содержит три столбца:

- `id` — первичный ключ таблицы;
- `fio` — фамилия, имя и отчество работника;
- `putdate` — дата рождения работника.

Таблица `tbl` разбивается по году рождения работника. Для этого в качестве аргумента функции `RANGE()` выступает выражение `YEAR(putdate)`, которое возвращает год рождения работника. В раздел `year0` попадают все работники, которые родились до 1980 года, в раздел `year1` — те, кто родился в период с 1980 до 2000 года, а в раздел `year2` — все, кто родился после 2000 года.

Листинг 17.6. Использование встроенной функции YEAR() совместно с функцией сегментирования RANGE()

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  fio TINYTEXT NOT NULL,
  putdate DATE NOT NULL
)
ENGINE=MYISAM
PARTITION BY RANGE (YEAR(putdate)) (
    PARTITION year0 VALUES LESS THAN (1981),
    PARTITION year1 VALUES LESS THAN (2001),
    PARTITION year2 VALUES LESS THAN MAXVALUE
);
INSERT INTO tbl VALUES (1, 'Тимирязев В.К.', '1972-12-02');
INSERT INTO tbl VALUES (2, 'Малышев Ю.Г.', '1976-06-24');
INSERT INTO tbl VALUES (3, 'Абрамов К.Т.', '1968-10-24');
INSERT INTO tbl VALUES (4, 'Ганюшкин В.В.', '1986-12-07');
```

17.3.2. Списочная сегментация *LIST*

Списочная сегментация позволяет распределять записи по разделам при помощи дискретных списков. Пусть каталоги, представленные в листинге 17.5, распределены по отделам магазина следующим образом:

- первый отдел (`first`) — "Процессоры" (1), "Материнские платы" (2), "Жесткие диски" (4);
- второй отдел (`second`) — "Видеоадаптеры" (3), "Оперативная память" (5), "Сетевое оборудование" (8);
- третий отдел (`third`) — "Периферия" (6), "Карты оплаты" (7), "Источники бесперебойного питания" (9), "Программное обеспечение" (10).

Тогда при помощи списочной сегментации можно распределить товарные позиции в таблице `products` по отделам магазина, так чтобы товарные позиции из каждого отдела находились в своем собственном разделе (листинг 17.7).

Замечание

Таблицы `catalogs` и `products` связаны друг с другом через поле `id_catalog`, товарная позиция принадлежит тому элементу каталога, чей первичный ключ содержится в ее поле `id_catalog`.

Листинг 17.7. Списочная сегментация

```
CREATE TABLE products (
  id_product INT(11) NOT NULL,
```

```
  name TINYTEXT NOT NULL,
  price TINYTEXT NOT NULL,
  id_catalog INT(11) NOT NULL
)
ENGINE=MYISAM
PARTITION BY LIST (id_catalog) (
    PARTITION first  VALUES IN (1, 2, 4),
    PARTITION second VALUES IN (3, 5, 8),
    PARTITION third  VALUES IN (6, 7, 9, 10)
);
```

В листинге 17.7 создается таблица `products`, которая разбивается на три раздела: `first`, `second` и `third` в зависимости от того, какое значение принимает поле `id_catalog`. Списки задаются при помощи конструкции `VALUES IN (...)`, которая следует после имени раздела в конструкции `PARTITION` функции сегментирования `LIST()`. Следует отметить, что если поле `id_product` проиндексировано как первичный ключ, то использовать `id_catalog` в качестве аргумента функции `LIST()` не получится.

Замечание

В отличие от линейной сегментации в списочной сегментации не имеет значение порядок следования списков и элементов внутри списка.

Точно так же, как и в случае других видов сегментации, функция `LIST()` может принимать только целочисленные положительные значения. Если аргумент не является числовым, его необходимо искусственно приводить к числовому значению. Пусть имеется таблица `users`, в которой содержится разнообразная информация о пользователях, включая их e-mail, телефон и URL (листинг 17.8).

Листинг 17.8. Таблица с личной информацией о пользователях

```
CREATE TABLE users (
  id_user INT(11) NOT,
  surname VARCHAR(30),
  patronymic VARCHAR(30),
  name VARCHAR(30),
  phone varchar(12) default null,
  email TINYTEXT,
  url TINYTEXT,
  `status` ENUM('active','passive','lock','gold') DEFAULT 'active'
);
INSERT INTO users VALUES(1, 'Иванов', 'Валерьевич', 'Александр', '58-98-78',
'ivanov@email.ru', NULL, 'active');
```

```
INSERT INTO users VALUES (2, 'Лосев', 'Иванович', 'Сергей', '9057777777',
'losev@email.ru', NULL, 'passive');
INSERT INTO users VALUES (3, 'Симдянов', 'Вячеславович', 'Игорь',
'9056666100', 'simdyanov@softtime.ru', 'http://www.softtime.ru/', 'active');
INSERT INTO users VALUES (4, 'Кузнецов', 'Валерьевич', 'Максим', NULL,
'kuznetsov@softtime.ru', 'http://www.softtime.ru', 'active');
INSERT INTO users VALUES (5, 'Нехорошев', 'Юрьевич', 'Анатолий', NULL, NULL,
NULL, 'lock');
INSERT INTO users VALUES (6, 'Корнеев', 'Александрович', 'Александр', '89-78-
36', 'korneev@domen.ru', NULL, 'gold');
```

Разобьем таблицу на две части так, чтобы в первую попали пользователи, чья фамилия начинается с букв, находящихся в диапазоне от А до Л, а во вторую — фамилии от М до Я. Использовать первый символ фамилии, к сожалению, нельзя, поэтому придется преобразовать его к ASCII-кодам при помощи функции `ASCII()` (листинг 17.9).

Замечание

Так как функция `ASCII()` преобразует лишь первый элемент строки, нет необходимости вырезать его при помощи функции `SUBSTRING()`. Впрочем, в текущей версии это все равно не удалось бы, т. к. вложенные функции в сегментационных функциях не допускаются.

Листинг 17.9. Использование встроенной функции `ASCII()` совместно с функцией сегментирования `LIST()`

```
CREATE TABLE users (
  id_user INT(11) NOT NULL,
  surname VARCHAR(30),
  patronymic VARCHAR(30),
  name VARCHAR(30),
  phone varchar(12) default null,
  email TINYTEXT,
  url TINYTEXT,
  `status` ENUM('active','passive','lock','gold') DEFAULT 'active'
)
ENGINE=MYISAM
PARTITION BY LIST (ASCII(surname)) (
    PARTITION a_l VALUES IN (192, 193, 194, 195, 196,
                             197, 168, 198, 199, 200, 201, 202, 203),
    PARTITION m_z VALUES IN (204, 205, 206, 207, 208, 209, 210, 211,
                             212, 213, 214, 215, 216, 217, 218, 219,
                             220, 221, 222, 223, NULL)
);
```

Следует обратить внимание на тот факт, что в раздел `m_z` добавлено значение `NULL`, т. к. передача функции `ASCII()` значения `NULL` приводит к тому, что возвращается `NULL`. Если это значение не добавить ни в один из разделов, добавление записи будет завершаться неудачей.

17.3.3. Хэш-сегментация *HASH*

В случае хэш-сегментации записи таблицы автоматически распределяются по разделам, пользователю остается лишь выбрать количество разделов, на которые будет разбита таблица. Создание разделов осуществляется при помощи функции сегментации `HASH()`, которая точно так же, как и другие функции сегментации, принимает в качестве аргумента либо целочисленный столбец таблицы, либо выражение, которое возвращает целое положительное число.

В листинге 17.10 приводится пример создания таблицы `catalogs`, разделенной на 10 частей.

Листинг 17.10. Хэш-сегментация

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  PRIMARY KEY (id_catalog)
)
ENGINE=MYISAM
PARTITION BY HASH (id_catalog)
PARTITIONS 10;
```

Количество разделов задается при помощи ключевого слова `PARTITIONS`. Однако вполне допускается создание таблицы с хэш-сегментацией без использования ключевого слова `PARTITIONS`, в этом случае таблица просто не разбивается на разделы.

Помимо регулярной хэш-сегментации, которая определяется функцией `HASH()`, СУБД MySQL поддерживает линейную хэш-сегментацию, которая определяется функцией `LINEAR HASH()`. В случае функции `HASH()` хэш вычисляется по остатку деления при помощи функции `MOD()`, т. е. номер раздела для записи, вставляемой в таблицу `catalogs`, определяется по формуле

`N = MOD(id_catalog, num)`

Здесь *num* — количество разделов, в случае таблицы `catalogs` из листинга 17.10 это значение равно 10. Если значение `id_catalog` делится без остатка на 10 разделов, запись попадает в первый раздел (нумерация начинается с 0), если остаток деления равен 1, то во второй раздел, и т. д.

В случае линейной хэш-сегментации хэш вычисляется по более сложному алгоритму:

```
V = POWER(2, CEILING(LOG(2, num)))
N = id_catalog & (V — 1)
если V > num
V = CEIL(V/2)
N = N & (V -1)
```

17.3.4. Индексная сегментация *KEY*

Индексная сегментация похожа на хэш-сегментацию, здесь пользователь может также указать только количество разделов, на которые должна быть разбита таблица. Распределение записей по разделам происходит в автоматическом режиме. Однако вместо функции `HASH()` используется функция `KEY()`, которая может разбивать таблицу по одному или большему количеству столбцов (листинг 17.11).

Листинг 17.11. Использование индексной сегментации

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
  name VARCHAR(30) NOT NULL,
  PRIMARY KEY (id_catalog, name)
)
ENGINE=MYISAM
PARTITION BY KEY (id_catalog, name)
PARTITIONS 10;
```

Как видно из листинга 17.11, функции сегментирования `KEY()` передается сразу два столбца, более того, один из столбцов не целочисленный, а является строкой. Единственным ограничением функции `KEY()` является то, что столбцы, передаваемые ей в качестве аргумента, должны входить в первичный ключ таблицы.

Вдобавок, если первичный ключ определен явно, можно не указывать столбцы, входящие в первичный ключ `id_catalog` и `name` в качестве аргументов функции `KEY()` — функция автоматически их распознает в таблице (листинг 17.12).

Листинг 17.12. Можно не указывать столбцы первичного ключа в качестве аргументов функции `KEY()`

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
```

```
  name VARCHAR(30) NOT NULL,
  PRIMARY KEY (id_catalog, name)
)
ENGINE=MYISAM
PARTITION BY KEY ()
PARTITIONS 10;
```

Для вычисления хэша в случае индексной сегментации используется функция `PASSWORD()`, именно поэтому допускается использование составных первичных ключей, содержащих в том числе и строковые столбцы.

17.4. Подразделы

Каждый из разделов, получаемых в результате сегментации, можно разбить на подразделы. Если тип разделов указывается при помощи конструкции `PARTITION BY`, то тип подраздела — при помощи `SUBPARTITION BY`. В листинге 17.13 приводится пример использования подразделов.

Листинг 17.13. Использование подразделов

```
CREATE TABLE users (
  id INT(11) NOT NULL,
  fio TINYTEXT NOT NULL,
  putdate DATE NOT NULL
)
ENGINE=MYISAM
PARTITION BY RANGE( YEAR(putdate) )
SUBPARTITION BY HASH( TO_DAYS(putdate) )
SUBPARTITIONS 2 (
  PARTITION p0 VALUES LESS THAN (1980),
  PARTITION p1 VALUES LESS THAN (2000),
  PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

Таблица `users` разбивается на три раздела (`p0`, `p1` и `p2`), каждый из которых в свою очередь разбивается на два подраздела (количество подразделов указывается при помощи ключевого слова `SUBPARTITIONS`). Таким образом, таблица разбивается на шесть ($2 \times 3 = 6$) подразделов. Следует обратить внимание, что типы сегментации разделов и подразделов могут не совпадать.

Теперь, если заглянуть в подкаталог, соответствующий базе данных в каталоге данных, можно убедиться, что СУБД MySQL создала для таблицы `users` четырнадцать файлов:

- users.frm — файл, содержащий структуру таблицы, информацию об именах и типах столбцов;
- users.par — файл, содержащий описание и структуру разделов, на которые разбита таблица `users`;
- users#P#p0#SP#p0sp0.MYD — файл данных первого раздела `p0` и первого подраздела `p0sp0`;
- users#P#p0#SP#p0sp0.MYI — файл индексов первого раздела `p0` и первого подраздела `p0sp0`;
- users#P#p0#SP#p0sp1.MYD — файл данных первого раздела `p0` и второго подраздела `p0sp1`;
- users#P#p0#SP#p0sp1.MYI — файл индексов первого раздела `p0` и второго подраздела `p0sp1`;
- users#P#p1#SP#p1sp0.MYD — файл данных второго раздела `p1` и первого подраздела `p1sp0`;
- users#P#p1#SP#p1sp0.MYI — файл индексов второго раздела `p1` и первого подраздела `p1sp0`;
- users#P#p1#SP#p1sp1.MYD — файл данных второго раздела `p1` и второго подраздела `p1sp1`;
- users#P#p1#SP#p1sp1.MYI — файл индексов второго раздела `p1` и второго подраздела `p1sp1`;
- users#P#p2#SP#p2sp0.MYD — файл данных третьего раздела `p2` и первого подраздела `p2sp0`;
- users#P#p2#SP#p2sp0.MYI — файл индексов третьего раздела `p2` и первого подраздела `p2sp0`;
- users#P#p2#SP#p2sp1.MYD — файл данных третьего раздела `p2` и второго подраздела `p2sp1`;
- users#P#p2#SP#p2sp1.MYI — файл индексов третьего раздела `p2` и второго подраздела `p2sp1`.

Если имена подразделов не указываются, им назначаются автоматически сгенерированные имена, которые формируются из имен разделов. Однако синтаксис объявления подразделов допускает назначение собственных имен. В листинге 17.14 приводится альтернативный способ объявления таблицы `users` из листинга 17.13.

Замечание

Разделы должны иметь одинаковое количество подразделов.

Листинг 17.14. Явное назначение имен подразделам

```
CREATE TABLE users (
  id INT(11) NOT NULL,
  fio TINYTEXT NOT NULL,
  putdate DATE NOT NULL
)
ENGINE=MYISAM
PARTITION BY RANGE( YEAR(putdate) )
SUBPARTITION BY HASH( TO_DAYS(putdate) ) (
  PARTITION p0 VALUES LESS THAN (1980) (
    SUBPARTITION s0,
    SUBPARTITION s1
  ),
  PARTITION p1 VALUES LESS THAN (2000) (
    SUBPARTITION s2,
    SUBPARTITION s3
  ),
  PARTITION p2 VALUES LESS THAN MAXVALUE (
    SUBPARTITION s4,
    SUBPARTITION s5
  )
);
```

Таблицы, создаваемые листингами 17.13 и 17.14, эквивалентны, только в случае листинга 17.14 подразделам назначаются не автоматически сгенерированные имена, а имена, определяемые пользователем (это отражается также на файлах данных и индексов подразделов).

Имена подразделов должны быть уникальны в пределах таблицы. Причем, если подразделам назначаются имена, они должны быть назначены каждому подразделу таблицы. В листинге 17.15 приводится некорректное определение таблицы `users`, в котором имена подразделам назначены только в первом и третьем разделах.

Листинг 17.15. Некорректное назначение имен подразделам

```
CREATE TABLE users (
  id INT(11) NOT NULL,
  fio TINYTEXT NOT NULL,
  putdate DATE NOT NULL
)
ENGINE=MYISAM
PARTITION BY RANGE( YEAR(putdate) )
SUBPARTITION BY HASH( TO_DAYS(putdate) ) (
```

```
    PARTITION p0 VALUES LESS THAN (1980) (
        SUBPARTITION s0,
        SUBPARTITION s1
    ),
    PARTITION p1 VALUES LESS THAN (2000),
    PARTITION p2 VALUES LESS THAN MAXVALUE (
        SUBPARTITION s2,
        SUBPARTITION s3
    )
);
ERROR 1064 (42000): Wrong number of subpartitions defined, mismatch with
previous setting near '
    PARTITION p2 VALUES LESS THAN MAXVALUE (
        SUBPARTITION s2,
        S' at line 13
```

17.5. Распределение подразделов по разным папкам и дискам

Подразделы можно расположить в разных папках и даже на разных дисках. Пусть таблица `users` разбивается на три раздела, каждый из которых имеет три подраздела. Расположим подразделы каждого из разделов на отдельном диске (листинг 17.16).

Замечание

В листинге 17.16 демонстрируется распределение подразделов по разным дискам в СУБД MySQL, работающей на компьютере под управлением операционной системы Windows. В UNIX-подобных операционных системах жесткие диски монтируются в один из каталогов файловой системы. Поэтому в ключевых словах `DATA DIRECTORY` и `INDEX DIRECTORY` указывается абсолютный путь от корня `/`.

Листинг 17.16. Расположения разделов на разных дисках

```
CREATE TABLE users (
  id INT(11) NOT NULL,
  fio TINYTEXT NOT NULL,
  putdate DATE NOT NULL
)
ENGINE=MYISAM
PARTITION BY RANGE( YEAR(putdate) )
SUBPARTITION BY HASH( TO_DAYS(putdate) ) (
  PARTITION p0 VALUES LESS THAN (1980) (
    SUBPARTITION s0
```

```
      DATA DIRECTORY = 'C:/mysql'
      INDEX DIRECTORY = 'C:/mysql',
    SUBPARTITION s1
      DATA DIRECTORY = 'C:/mysql'
      INDEX DIRECTORY = 'C:/mysql'
  ),
  PARTITION p1 VALUES LESS THAN (2000) (
    SUBPARTITION s2
      DATA DIRECTORY = 'D:/mysql'
      INDEX DIRECTORY = 'D:/mysql',
    SUBPARTITION s3
     DATA DIRECTORY = 'D:/mysql'
     INDEX DIRECTORY = 'D:/mysql'
  ),
  PARTITION p2 VALUES LESS THAN MAXVALUE (
    SUBPARTITION s4
      DATA DIRECTORY = 'E:/mysql'
      INDEX DIRECTORY = 'E:/mysql',
    SUBPARTITION s5
      DATA DIRECTORY = 'E:/mysql'
      INDEX DIRECTORY = 'E:/mysql'
  )
);
```

Как видно из листинга 17.16, раздел `p0` располагается на диске C:\, раздел `p1` — на диске D:\, а раздел `p2` — на диске E:\. Каждому из подразделов при помощи директив `DATA DIRECTORY` и `INDEX DIRECTORY` можно указать папку, в которой будут расположены файлы данных и индексов соответственно. В листинге 17.17 приводится пример таблицы `users`, в которой файлы данных сохраняются на диске C:\, а файлы индексов — на диске D:\.

Листинг 17.17. Расположение файлов данных и индексов на разных дисках

```
CREATE TABLE users (
  id INT(11) NOT NULL,
  fio TINYTEXT NOT NULL,
  putdate DATE NOT NULL
)
ENGINE=MYISAM
PARTITION BY RANGE( YEAR(putdate) )
SUBPARTITION BY HASH( TO_DAYS(putdate) ) (
  PARTITION p0 VALUES LESS THAN (1980) (
    SUBPARTITION s0
      DATA DIRECTORY = 'C:/mysql'
      INDEX DIRECTORY = 'D:/mysql',
```

```
    SUBPARTITION s1
      DATA DIRECTORY = 'C:/mysql'
      INDEX DIRECTORY = 'D:/mysql'
  ),
  PARTITION p1 VALUES LESS THAN (2000) (
    SUBPARTITION s2
      DATA DIRECTORY = 'C:/mysql'
      INDEX DIRECTORY = 'D:/mysql',
    SUBPARTITION s3
     DATA DIRECTORY = 'C:/mysql'
     INDEX DIRECTORY = 'D:/mysql'
  ),
  PARTITION p2 VALUES LESS THAN MAXVALUE (
    SUBPARTITION s4
      DATA DIRECTORY = 'C:/mysql'
      INDEX DIRECTORY = 'D:/mysql',
    SUBPARTITION s5
      DATA DIRECTORY = 'C:/mysql'
      INDEX DIRECTORY = 'D:/mysql'
  )
);
```

17.6. Удаление разделов

Сегментированные таблицы очень удобно использовать для удаления старых блоков. Для этого используется опция `DELETE PARTITION` оператора `ALTER TABLE`. В листинге 17.18 создается таблица `users` с линейной сегментацией по дате рождения `putdate`. При помощи оператора `ALTER TABLE` уничтожается раздел `year0` с пользователями, которые родились до 1961 года.

Листинг 17.18. Уничтожение раздела

```
CREATE TABLE users (
  id INT(11) NOT NULL,
  fio TINYTEXT NOT NULL,
  putdate DATE NOT NULL
)
ENGINE=MYISAM
PARTITION BY RANGE (YEAR(putdate)) (
    PARTITION year0 VALUES LESS THAN (1961),
    PARTITION year1 VALUES LESS THAN (1981),
    PARTITION year2 VALUES LESS THAN (2001),
    PARTITION year3 VALUES LESS THAN MAXVALUE
);
```

```
ALTER TABLE users DROP PARTITION year0;
SHOW CREATE TABLE users\G
*************************** 1. row ***************************
       Table: users
Create Table: CREATE TABLE `users` (
  `id` int(11) NOT NULL,
  `fio` tinytext NOT NULL,
  `putdate` date NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=cp1251 /*!50100 PARTITION BY RANGE
(YEAR(putdate)) (PARTITION year1 VALUES LESS THAN (1981) ENGINE
= MyISAM, PARTITION year2 VALUES LESS THAN (2001) ENGINE = MyISAM, PARTITION
year3 VALUES LESS THAN MAXVALUE ENGINE = MyISAM) */
```

Такой способ удаления данных гораздо эффективнее, чем использование оператора `DELETE`.

Следует отметить, что все записи, которые входят в раздел при использовании оператора `ALTER TABLE...DROP PARTITION...`, уничтожаются. В случае линейной сегментации `RANGE` новые записи попадают в следующий раздел. Если уничтожается раздел списочной сегментации `LIST`, вставка записей, соответствующих этому разделу, приводит к ошибке.

При помощи опции `DROP PARTITION` оператора `ALTER TABLE` нельзя уничтожить раздел индексной или хэш-сегментации. В этом случае прибегают к опции `COALESCE PARTITION`, которая позволяет удалить лишние разделы (листинг 17.19).

Листинг 17.19. Объединение разделов

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  PRIMARY KEY (id_catalog)
)
ENGINE=MYISAM
PARTITION BY HASH (id_catalog)
PARTITIONS 10;
ALTER TABLE catalogs COALESCE PARTITION 4;
SHOW CREATE TABLE catalogs\G
*************************** 1. row ***************************
       Table: catalogs
Create Table: CREATE TABLE `catalogs` (
  `id_catalog` int(11) NOT NULL AUTO_INCREMENT,
  `name` tinytext NOT NULL,
  PRIMARY KEY (`id_catalog`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251 /*!50100 PARTITION BY HASH (id_catalog)
PARTITIONS 6  */
```

Как видно из листинга 17.19, при помощи оператора `ALTER TABLE... COALESCE PARTITION` количество разделов в таблице `catalogs` уменьшается с 10 до 6.

При помощи опции `DROP PARTITION` нельзя удалить все разделы таблицы, если производится попытка уничтожить больше разделов, чем содержит таблица, возвращается ошибка "Cannot remove all partitions, use DROP TABLE instead" ("Невозможно удалить все разделы таблицы, используйте оператор DROP TABLE") (листинг 17.20).

Листинг 17.20. Попытка удалить все разделы таблицы

```
mysql> ALTER TABLE catalogs COALESCE PARTITION 20;
ERROR 1496 (HY000): Cannot remove all partitions, use DROP TABLE instead
```

17.7. Добавление разделов

Помимо удаления разделов, при помощи оператора `ALTER TABLE` в таблицу можно добавить новый раздел. Однако добавление разделов для каждого из типов сегментации имеет определенные ограничения.

В случае линейной сегментации `RANGE` добавлять можно разделы только в сторону увеличения интервалов. В листинге 17.21 при помощи опции `ADD PARTITION` оператора `ALTER TABLE` таблице `users` добавляется новый раздел для пользователей, чья дата рождения находится в интервале от 1981 до 2000 года.

Листинг 17.21. Добавление раздела в случае линейной сегментации `RANGE`

```
CREATE TABLE users (
  id INT(11) NOT NULL,
  fio TINYTEXT NOT NULL,
  putdate DATE NOT NULL
)
ENGINE=MYISAM
PARTITION BY RANGE (YEAR(putdate)) (
    PARTITION year0 VALUES LESS THAN (1961),
    PARTITION year1 VALUES LESS THAN (1981)
);
ALTER TABLE users
ADD PARTITION (PARTITION year2 VALUES LESS THAN (2001));
SHOW CREATE TABLE users\G
*************************** 1. row ***************************
       Table: users
```

```
Create Table: CREATE TABLE `users` (
  `id` int(11) NOT NULL,
  `fio` tinytext NOT NULL,
  `putdate` date NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=cp1251 /*!50100 PARTITION BY RANGE
(YEAR(putdate)) (PARTITION year0 VALUES LESS THAN (1961) ENGINE
= MyISAM, PARTITION year1 VALUES LESS THAN (1981) ENGINE = MyISAM, PARTITION
year2 VALUES LESS THAN (2001) ENGINE = MyISAM) */
```

Однако добавить раздел для пользователей с датой рождения от 1941 до 1960 года уже не получится (листинг 17.22).

Листинг 17.22. Добавить раздел перед существующими разделами в линейной сегментации нельзя

```
ALTER TABLE users
ADD PARTITION (PARTITION year_1 VALUES LESS THAN (1941));
ERROR 1481 (HY000): VALUES LESS THAN value must be strictly increasing for
each partition
```

В случае списочной сегментации `LIST` ограничений, связанных с местоположением раздела, не существует. Однако список добавляемого раздела не должен включать в себя значения из других разделов. В листинге 17.23 приводится пример добавления нового списочного раздела.

Листинг 17.23. Добавление раздела в случае списочной сегментации `LIST`

```
CREATE TABLE products (
  id_product INT(11) NOT NULL,
  name TINYTEXT NOT NULL,
  price TINYTEXT NOT NULL,
  id_catalog INT(11) NOT NULL
)
ENGINE=MYISAM
PARTITION BY LIST (id_catalog) (
    PARTITION first VALUES IN (1, 2, 4),
    PARTITION second VALUES IN (3, 5, 8),
    PARTITION third VALUES IN (6, 7, 9, 10)
);
ALTER TABLE products
ADD PARTITION (PARTITION fourth VALUES IN (11, 12, 13));
```

В случае индексной `KEY` и хэш-сегментации `HASH` SQL-запрос, увеличивающий количество разделов в таблице, может выглядеть так, как это представлено в листинге 17.24.

Листинг 17.24. Добавление разделов в случае индексной KEY и хэш-сегментации HASH

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  PRIMARY KEY (id_catalog)
)
ENGINE=MYISAM
PARTITION BY HASH (id_catalog)
PARTITIONS 10;
ALTER TABLE catalogs ADD PARTITION PARTITIONS 6;
```

В листинге 17.24 количество разделов увеличивается с 10 до 16.

17.8. Перепланировка разделов

Главным неудобством во время удаления разделов при помощи опции DELETE PARTITION оператора ALTER TABLE является уничтожение данных, находящихся в разделе. Когда требуется реорганизовать разделы с сохранением данных, прибегают к опции REORGANIZE PARTITION оператора ALTER TABLE.

В листинге 17.25 приводится пример запроса, в котором раздел year0 разбивается на два новых раздела: year0s и year1s.

Листинг 17.25. Перепланировка разделов

```
CREATE TABLE users (
  id INT(11) NOT NULL,
  fio TINYTEXT NOT NULL,
  putdate DATE NOT NULL
)
ENGINE=MYISAM
PARTITION BY RANGE (YEAR(putdate)) (
    PARTITION year0 VALUES LESS THAN (1961),
    PARTITION year1 VALUES LESS THAN (1981),
    PARTITION year2 VALUES LESS THAN (2001),
    PARTITION year3 VALUES LESS THAN MAXVALUE
);
ALTER TABLE users REORGANIZE PARTITION year0 INTO (
    PARTITION year0s VALUES LESS THAN (1951),
    PARTITION year1s VALUES LESS THAN (1961)
);
```

```
SHOW CREATE TABLE users\G
*************************** 1. row ***************************
       Table: users
Create Table: CREATE TABLE `users` (
  `id` int(11) NOT NULL,
  `fio` tinytext NOT NULL,
  `putdate` date NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=cp1251
 /*!50100 PARTITION BY RANGE (YEAR(putdate)) (
  PARTITION year0s VALUES LESS THAN (1951) ENGINE = MyISAM,
  PARTITION year1s VALUES LESS THAN (1961) ENGINE = MyISAM,
  PARTITION year1 VALUES LESS THAN (1981) ENGINE = MyISAM,
  PARTITION year2 VALUES LESS THAN (2001) ENGINE = MyISAM,
  PARTITION year3 VALUES LESS THAN MAXVALUE ENGINE = MyISAM) */
```

В листинге 17.26 демонстрируется обратная операция — разделы `year0s` и `year1s` объединяются в один раздел `year0`.

Листинг 17.26. Объединение разделов в один

```
ALTER TABLE users REORGANIZE PARTITION year0s, year1s INTO (
    PARTITION year0 VALUES LESS THAN (1961)
);
SHOW CREATE TABLE users\G
*************************** 1. row ***************************
       Table: users
Create Table: CREATE TABLE `users` (
  `id` int(11) NOT NULL,
  `fio` tinytext NOT NULL,
  `putdate` date NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=cp1251
/*!50100 PARTITION BY RANGE (YEAR(putdate)) (
  PARTITION year0 VALUES LESS THAN (1961) ENGINE = MyISAM,
  PARTITION year1 VALUES LESS THAN (1981) ENGINE = MyISAM,
  PARTITION year2 VALUES LESS THAN (2001) ENGINE = MyISAM,
  PARTITION year3 VALUES LESS THAN MAXVALUE ENGINE = MyISAM) */
```

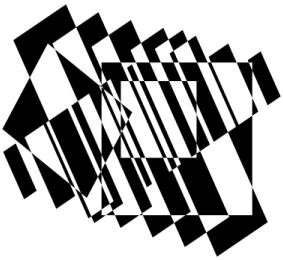
При перепланировке таблицы можно использовать все имеющиеся разделы и перераспределять их произвольным образом. В листинге 17.27 разделы `year0`, `year1`, `year2`, `year3` объединяются и вновь разбиваются с шагом 10 лет.

Листинг 17.27. Полное перепроектирование таблицы `users`

```
ALTER TABLE users REORGANIZE PARTITION year0, year1, year2, year3 INTO (
    PARTITION year0 VALUES LESS THAN (1961),
```

```
    PARTITION year1 VALUES LESS THAN (1971),
    PARTITION year2 VALUES LESS THAN (1981),
    PARTITION year3 VALUES LESS THAN (1991),
    PARTITION year4 VALUES LESS THAN (2001),
    PARTITION year5 VALUES LESS THAN MAXVALUE
);
SHOW CREATE TABLE users\G
*************************** 1. row ***************************
       Table: users
Create Table: CREATE TABLE `users` (
  `id` int(11) NOT NULL,
  `fio` tinytext NOT NULL,
  `putdate` date NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=cp1251
/*!50100 PARTITION BY RANGE (YEAR(putdate)) (
  PARTITION year0 VALUES LESS THAN (1961) ENGINE = MyISAM,
  PARTITION year1 VALUES LESS THAN (1971) ENGINE = MyISAM,
  PARTITION year2 VALUES LESS THAN (1981) ENGINE = MyISAM,
  PARTITION year3 VALUES LESS THAN (1991) ENGINE = MyISAM,
  PARTITION year4 VALUES LESS THAN (2001) ENGINE = MyISAM,
  PARTITION year5 VALUES LESS THAN MAXVALUE ENGINE = MyISAM) */
```

ГЛАВА 18



Взаимодействие с PHP

Не секрет, что СУБД MySQL очень популярна в Web-разработках, и на сегодняшний момент в подавляющем большинстве случаев используется совместно с языком программирования PHP, в котором последнее время приобретает популярность объектно-ориентированный подход в программировании. Использование объектно-ориентированного подхода с классическими реляционными базами данных затруднено тем, что объект приходится сериализовать или представлять не в объектной форме, чтобы поместить в базу данных, а при извлечении опять восстанавливать объект из записи таблицы. Для решения этой проблемы зачастую создаются разнообразные наборы классов, не совместимые друг с другом. Чтобы решить эту задачу, в PHP введено, помимо классического расширения для взаимодействия с PHP — php_mysql, новое улучшенное расширение php_mysqli, которое предоставляет разработчику объектно-ориентированный интерфейс для взаимодействия с MySQL.

В данной главе будут рассмотрены как классическое расширение php_mysql, так и приобретающее все большую популярность объектно-ориентированное расширение php_mysqli.

18.1. Расширение php_mysql

Библиотека php_mysql предоставляет классический файловый интерфейс к СУБД MySQL. На заре программирования, когда только устанавливались традиции современных программных интерфейсов, было принято каждый объект рассматривать как файл. Фрагмент памяти на жестком диске — файл, каталог — тоже файл (который хранит список других файлов), внешнее устройство или сетевая карта — это тоже файл, только специального типа. Это позволяло использовать для всех операций привычную каждому программисту схему взаимодействия:

1. Открыть ресурс.
2. Записать или прочитать из ресурса данные.
3. Закрыть ресурс.

Замечание

В языке программирования FORTRAN, одном из первых языков высокого уровня, даже переменные рассматриваются как файлы, например, преобразование строки в число выглядит как запись содержимого файла-строки в файл-число.

Одновременно может быть открыто несколько ресурсов (файлов). Чтобы отличать их друг от друга, используются специальные объекты — дескрипторы, при помощи которых можно получать доступ к тому или иному ресурсу. Благодаря дескрипторам можно одновременно работать сразу с несколькими открытыми источниками без конфликтов и опасности перепутать один ресурс с другим.

При работе с MySQL из PHP используются два вида дескриптора:

- дескриптор соединения — обозначает соединение с сервером, например, если необходимо в одном приложении открыть два соединения сразу с двумя различными серверами, то в приложении будут два разных дескриптора, которые нужно будет указывать в качестве параметра функциям, чтобы клиентская библиотека могла направить запрос по правильному адресу. Получить два дескриптора для одного и того же сервера с одинаковыми параметрами подключения нельзя — в этом случае библиотека вернет уже существующий дескриптор;
- дескриптор результирующей таблицы — взаимодействие PHP-приложения и сервера MySQL происходит по клиент-серверной технологии: посылается SQL-запрос, на который сервер возвращает результирующую таблицу. Результирующая таблица сама представляет собой ресурс, по которому можно перемещаться, выставляя курсор в начало таблицы, перемещаясь по записям. Именно для доступа к результату выполнения запроса предназначен дескриптор результирующей таблицы.

Рассмотрим кратко жизненный цикл программы, обращающейся к серверу MySQL. В первую очередь следует установить соединение с сервером при помощи функции `mysql_connect()`, которая имеет следующий синтаксис:

```
resource mysql_connect ([server [, username [, password [, new_link [,
                         client_flags]]]]])
```

Эта функция устанавливает соединение с сервером MySQL, сетевой адрес которого задается параметром *`server`*. Вторым и третьим аргументами этой функции являются имя пользователя базы данных *`username`* и его пароль *`password`* соответственно.

По умолчанию повторный вызов функции `mysql_connect()` с теми же аргументами не приводит к установлению нового соединения, вместо этого функция возвращает дескриптор уже существующего соединения. Если четвертому параметру *new_link* присвоить значение `TRUE`, будет открыто новое соединение с сервером. Параметр *client_flags* должен быть комбинацией из следующих констант:

- `MYSQL_CLIENT_COMPRESS` — предписывает использование протокола сжатия при обмене информацией между сервером и клиентом;
- `MYSQL_CLIENT_IGNORE_SPACE` — в SQL-запросах после имен функций разрешается использование пробелов;
- `MYSQL_CLIENT_INTERACTIVE` — ждать `interactive_timeout` секунд до закрытия соединения, если между сервером и клиентом не происходит обмен данными;
- `MYSQL_CLIENT_SSL` — использовать при соединении с сервером защищенный SSL-канал.

Замечание

Все аргументы функции являются необязательными. В случае их отсутствия, по умолчанию, для этой функции устанавливаются следующие параметры: *server* = `'localhost:3306'`, *username* принимает значение владельца процесса сервера, а *password* принимает пустую строку.

В случае успеха функция возвращает дескриптор соединения с сервером, при неудаче возвращает значение `FALSE`.

После того как соединение установлено, необходимо выбрать базу данных при помощи функции `mysql_select_db()`, которая эквивалентна вызову команды `USE` в консольном клиенте mysql. Функция имеет следующий синтаксис:

```
bool mysql_select_db (string database_name [, resource link_identifier])
```

Функция принимает в качестве аргументов название выбираемой базы данных *database_name* и дескриптор соединения *link_identifier*, который ранее был возвращен функцией `mysql_connect()`. Функция возвращает `TRUE` при успешном выполнении операции и `FALSE` в противном случае.

После того как соединение установлено и выбрана база данных, с которой будет осуществляться дальнейшая работа, можно отправлять серверу SQL-запросы. Данную операцию можно осуществить при помощи функции `mysql_query()`, которая имеет следующий синтаксис:

```
resource mysql_query (query [, link_identifier])
```

Первый аргумент функции представляет собой строку с запросом *query*, второй (*link_identifier*) — дескриптор соединения, возвращаемый функцией `mysql_connect()`.

Замечание

При передаче запроса функции `mysql_query()` точку с запятой в конце запроса, обязательную при работе с клиентом `mysql`, можно не ставить.

Функция возвращает дескриптор результирующей таблицы в случае успеха и `FALSE` в случае неудачного выполнения запроса. Важно понимать, что об ошибках, которые происходят на сервере MySQL, интерпретатор PHP не подозревает и не выводит никаких сообщений (как это происходит обычно с ошибками в PHP-коде). Поэтому каждый SQL-запрос следует обрабатывать на предмет ошибок. Получить сообщение об ошибке можно при помощи функции `mysql_error()`.

Дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`, используется далее для получения значений, возвращаемых СУБД. Обычно это осуществляется при помощи одной из пяти функций: `mysql_result()`, `mysql_fetch_row()`, `mysql_fetch_assoc()`, `mysql_fetch_array()` и `mysql_fetch_object()`.

Закрытие соединения производится при помощи функции `mysql_close()`. Ее синтаксис представлен ниже:

```
bool mysql_close ([link_identifier])
```

В качестве необязательного параметра функция принимает дескриптор открытого соединения *`link_identifier`*. Если этот параметр не указан, закрывается последнее открытое соединение. Функция возвращает `TRUE` в случае успеха и `FALSE` при возникновении ошибки.

В PHP можно не закрывать соединение, т. к. скрипты работают короткое время, а в конце работы скрипта соединение закрывается автоматически.

Соберем в единое приложение все рассмотренные выше функции. Операция установки соединения с СУБД MySQL будет использоваться очень часто, поэтому выделим ее в отдельный PHP-файл config.php (листинг 18.1).

Листинг 18.1. Установка соединения с MySQL-сервером (config.php)

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных на хостинге или локальной машине
  $dbname = "test";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";
```

```
  // Устанавливаем соединение с базой данных
  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
  if (!$dbcnx) {
   exit( "<P>В настоящий момент сервер базы данных не доступен,
            поэтому корректное отображение страницы невозможно.</P>" );
  }
  // Выбираем базу данных
  if (! @mysql_select_db($dbname, $dbcnx) ) {
    exit( "<P>В настоящий момент база данных не доступна,
             поэтому корректное отображение страницы невозможно.</P>" );
  }

  // Устанавливаем кодировку соединения. Следует выбрать ту кодировку,
  // в которой данные будут отправляться MySQL-серверу
  @mysql_query("SET NAMES 'cp1251'");
?>
```

Как видно из листинга 18.1, после установки соединения при помощи функции `mysql_connect()` и выбора базы данных при помощи функции `mysql_select_db()` MySQL-серверу отправляется запрос `"SET NAMES 'cp1251'"`. Данный запрос сообщает, что все передаваемые на сервер данные будут в кодировке Windows-1251, и MySQL-сервер, в свою очередь, в какой бы кодировке не хранились данные в таблицах, должен предоставлять результат в кодировке Windows-1251.

В листинге 18.2 приводится пример, возвращающий версию MySQL-сервера. Скрипт включает при помощи конструкции `require_once()` файл config.php.

Листинг 18.2. Получение версии MySQL-сервера

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Формируем и выполняем SQL-запрос
  $query = "SELECT VERSION()";
  $ver = mysql_query($query);

  // Проверяем правильность выполнения запроса
  if(!$ver) exit("Ошибка выполнения запроса ".mysql_error());

  // Получаем результат выполнения запроса
  echo mysql_result($ver, 0);
?>
```

Помимо рассмотренных выше функций, расширение PHP для работы с СУБД MySQL включает большое количество других функций, представленных в табл. 18.1.

Таблица 18.1. *Функции PHP для взаимодействия с MySQL*

Функция	**Описание**
mysql_affected_rows ([*link_identifier*])	В случае успеха, функция возвращает число затронутых операторами DELETE, UPDATE и REPLACE записей, в случае неудачи возвращается –1
mysql_change_user(*user*, *password*[, *database*[, *link_identifier*]])	Позволяет изменить параметры соединения (имя MySQL-пользователя *user*, его пароль *password*, название текущей базы данных *database*). В случае успеха возвращает TRUE, в случае неудачи — FALSE
mysql_client_encoding ([*link_identifier*])	Возвращает текущую кодировку соединения, которую извлекает из системной переменной MySQL — *character_set*
mysql_close ([*link_identifier*])	Закрывает соединение с сервером MySQL
mysql_connect([*server*[, *username*[, password[, *new_link*[, *client_flags*]]]]])	Открывает соединение с сервером MySQL и возвращает дескриптор соединения в случае успеха, в противном случае возвращается FALSE
mysql_data_seek (*result*, *row_number*)	Функция перемещает курсор результирующей таблицы *result* в позицию *row_number* (позиции начинаются с 0). Возвращает TRUE в случае успеха и FALSE в случае неудачи
mysql_db_name(*result*, *row*[, *field*])	Возвращает имя базы данных с номером *row* при обработке результата выполнения функции mysql_list_dbs(), которая возвращает список доступных баз данных
mysql_db_query(*database*, *query*[, *link_identifier*])	Переключается на базу данных *database* и выполняет запрос *query*. Возвращает дескриптор результирующей таблицы в случае успеха и FALSE в случае неудачи
mysql_errno ([*link_identifier*])	Возвращает номер ошибки для последней операции
mysql_error ([*link_identifier*])	Возвращает текстовое сообщение об ошибке для последней операции
mysql_escape_string (*unescaped_string*)	Экранирует специальные символы для SQL-запроса, который передается функции mysql_query(). Функция mysql_escape_string() является устаревшей, вместо нее рекомендуется использовать mysql_real_escape_string()

Таблица 18.1 *(продолжение)*

Функция	Описание
mysql_fetch_array(*result*[, *result_type*])	Функция возвращает текущую запись результирующей таблицы *result* в виде ассоциативного массива, числового массива или смешанного массива в зависимости от значения *result_type*
mysql_fetch_assoc(*result*)	Функция возвращает текущую запись результирующей таблицы *result* в виде ассоциативного массива
mysql_fetch_field(*result*[, *field_offset*])	Возвращает информацию о столбце в результирующей таблице *result* в виде объекта. При первом вызове возвращается информация о первом столбце, при последующих — о следующих по порядку столбцах. Можно извлечь информацию о произвольном столбце, если воспользоваться смещением *field_offset* (нумерация столбцов начинается с 0)
mysql_fetch_lengths (*result*)	Возвращает длину каждого поля в результирующей таблице *result* в виде массива
mysql_fetch_object(*result*[, *class_name*[, *params*]]	Функция возвращает текущую запись результирующей таблицы *result* в виде объекта
mysql_fetch_row(*result*)	Возвращает текущую запись результирующей таблицы *result* в виде числового массива
mysql_field_flags(*result*, *field_offset*)	Возвращает параметры поля в результирующей таблице *result*, смещенного от начала на позицию *field_offset* (нумерация полей начинается с 0)
mysql_field_len(*result*, *field_offset*)	Возвращает длину поля в результирующей таблице *result*, смещенного от начала на позицию *field_offset* (нумерация полей начинается с 0)
mysql_field_name(*result*, *field_offset*)	Возвращает имя поля в результирующей таблице *result*, смещенного от начала на позицию *field_offset* (нумерация полей начинается с 0)
mysql_field_seek(*result*, *field_offset*)	Функция устанавливает курсор результирующей таблицы на поле, указанное в смещении *field_offset* (нумерация полей начинается с 0)
mysql_field_table(*result*, *field_offset*)	Возвращает имя таблицы, которому принадлежит поле со смещением *field_offset* (нумерация полей начинается с 0). Функция предназначена для разбора результирующей таблицы многотабличного запроса
mysql_field_type(*result*, *field_offset*)	Возвращает тип данных (INT, CHAR, TEXT и т. п.) для поля в результирующей таблице *result*, смещенного относительно начала на позицию *field_offset* (нумерация полей начинается с 0)

Таблица 18.1 *(продолжение)*

Функция	Описание
mysql_free_result(*result*)	Освобождает память, выделенную под результирующую таблицу *result*. После вызова функции результирующая таблица не доступна
mysql_get_client_info()	Возвращает строку с версией клиентской библиотеки
mysql_get_host_info ([*link_identifier*])	Возвращает адрес MySQL-сервера и тип соединения
mysql_get_proto_info ([*link_identifier*])	Возвращает версию протокола взаимодействия клиента и MySQL-сервера
mysql_get_server_info ([*link_identifier*])	Возвращает версию MySQL-сервера
mysql_info ([*link_identifier*])	Возвращает информацию о последнем запросе, количество затронутых столбцов, предупреждений и т. п.
mysql_insert_id ([*link_identifier*])	Возвращает первичный ключ, сгенерированный при последнем INSERT-запросе по механизму AUTO_INCREMENT
mysql_list_dbs ([*link_identifier*])	Возвращает список баз данных, доступных текущему клиенту
mysql_list_fields (*database_name*, *table_name*[, *link_identifier*])	Возвращает список полей таблицы *table_name* базы данных *database_name*
mysql_list_processes ([*link_identifier*])	Возвращает список процессов MySQL, доступных текущему клиенту
mysql_list_tables(*database* [,*link_identifier*])	Возвращает список таблиц в базе данных *database*
mysql_num_fields(*result*)	Возвращает количество полей в результирующей таблице *result*
mysql_num_rows(*result*)	Возвращает количество строк в результирующей таблице *result*
mysql_pconnect([*server*[, *username*[, *password*[, *client_flags*]]]])	Открывает постоянное соединение с сервером MySQL и возвращает дескриптор соединения в случае успеха, в противном случае возвращается FALSE
mysql_ping ([*link_identifier*])	Проверяет соединение с сервером, и, если оно отсутствует, осуществляется повторная попытка соединения
mysql_query(*query*[, *link_identifier*])	Функция отсылает SQL-запрос серверу

Таблица 18.1 *(окончание)*

Функция	Описание
mysql_real_escape_string (*unescaped_string*[, *link_identifier*])	Экранирует специальные символы для SQL-запроса, который передается функции mysql_query()
mysql_result(*result*, *row*[, *field*])	Возвращает значение поля с позицией *row* в текущей записи результирующей таблицы *result*
mysql_select_db (*database_name*[, *link_identifier*])	Функция позволяет выбрать текущую базу данных *database_name*
mysql_stat ([*link_identifier*])	Возвращает массив, содержащий информацию о состоянии MySQL-сервера
mysql_tablename(*result*, *i*)	Возвращает имя таблицы с номером *i* из списка, который возвращает функция mysql_list_tables()
mysql_thread_id ([*link_identifier*])	Возвращает уникальный идентификатор текущего потока
mysql_unbuffered_query(*query*)	Отсылает SQL-запрос MySQL-серверу без автообработки результата и ее буферизации

18.2. Расширение php_mysqli

Расширение php_mysqli является новым объектно-ориентированным интерфейсом для доступа к СУБД MySQL из PHP-кода и доступен при работе с СУБД MySQL версии выше 4.1. Интерфейс включает в себя классы для доступа к СУБД MySQL, отложенных запросов и класса результирующей таблицы. Библиотека не заменяет собой функции из расширения php_mysql, наряду с объектно-ориентированным расширением доступен процедурный стиль программирования, рассмотренный в *разд. 18.1*.

Замечание

В отличие от старого расширения php_mysql в новом расширении php_mysqli функции начинаются с префикса mysqli, а не с mysql.

Главным классом является mysqli, при помощи его осуществляется установка соединения и выполнения запросов. В листинге 18.3 приводится пример, который устанавливает соединение с сервером MySQL и выводит его версию.

Листинг 18.3. Версия MySQL-сервера

```
<?php
  // Объявляем объект класса mysqli, который устанавливает
  // соединение с сервером
  $mysqli = new mysqli("localhost", "root", "", "test");
  if (mysqli_connect_errno())  exit("Ошибка установки соединения");

  // Выводим версию сервера
  echo $mysqli->server_info;

  // Закрываем соединение с сервером
  $mysqli->close();
?>
```

Методы класса `mysqli` повторяют по действиям функции расширения php_mysql и представлены в табл. 18.2.

Таблица 18.2. *Методы класса* `mysqli`

Метод	**Описание**
`autocommit(`*mode*`)`	Метод включает или отключает режим автозавершения транзакции. По умолчанию автозавершение включено, и каждый оператор рассматривается как отдельная транзакция. Отключение режима автозавершения требует явного завершения транзакции при помощи методов `commit()` и `rollback()`. Параметр *mode* принимает значение `TRUE` или `FALSE`
`change_user(`*user*, *password*, *database*`)`	Метод изменяет параметры соединения для текущего объекта `mysqli`, устанавливая нового пользователя *user*, его пароль *password* и текущую базу данных *database*
`character_set_name()`	Возвращает текущую кодировку соединения, которую извлекает из системной переменной MySQL — `character_set`
`close()`	Закрывает соединение с сервером MySQL
`commit()`	Подтверждение успешного завершения транзакции — все изменения записываются на жесткий диск
`get_client_info()`	Возвращает версию клиентской библиотеки
`kill(`*processid*`)`	Уничтожает процесс с идентификатором *processid*
`multi_query(`*query*`)`	Выполняет SQL-запросы в строке *query*, в отличие от метода `query()` в строке *query* допускается несколько SQL-запросов, разделенных точкой с запятой

Таблица 18.2 *(продолжение)*

Метод	Описание
more_results()	Метод проверяет, имеются ли записи в результирующей таблице, полученной в результате выполнения метода multi_query(), и возвращает TRUE, если записи имеются, и FALSE в противном случае
next_result()	Получение следующей записи в результирующей таблице, возвращенной в результате выполнения метода multi_query()
options(*option*, *value*)	Метод устанавливает для параметра *option* значение *value*
ping()	Проверяет соединение с сервером, и, если оно отсутствует, осуществляется повторная попытка соединения
prepare(*query*)	Метод создает отложенный запрос *query*, возвращая дескриптор отложенного запроса. При помощи метода execute() отложенного запроса его можно выполнить в любой удобный момент
query(*query*)	Выполняет SQL-запрос в строке *query*, в отличие от метода multi_query() возможно выполнение только одного запроса
real_connect([*hostname*[, *username*[, *passwd*[, *dbname*[, *port*[, *socket*[, *flags*]]]]]]])	Если соединение устанавливается не сразу при помощи конструктора класса mysqli, а объект получается при помощи функции mysqli_init(), то установить соединение с сервером можно при помощи этой функции
mysqli_real_escape_string (*escapestr*)	Экранирует специальные символы для SQL-запроса, который передается методу query() или multi_query()
rollback()	Откат транзакции, состояние СУБД изменяется на то, которое было до момента начала транзакции
select_db(*dbname*)	Позволяет выбрать текущую базу данных *dbname*
set_charset(*charset*)	Устанавливает текущую кодировку соединения *charset*
ssl_set(*key*, *cert*, *ca*, *capath*, *cipher*)	Устанавливает безопасное SSL соединение с MySQL-сервером
stat()	Возвращает строку, содержащую информацию о состоянии MySQL-сервера
stmt_init()	Возвращает объект отложенного запроса, является альтернативой методу prepare()
store_result()	Возвращает дескриптор результирующей таблицы последнего запроса

Таблица 18.2 *(окончание)*

Метод	Описание
use_result()	Инициирует результирующую таблицу последнего запроса
thread_safe()	Возвращает TRUE, если клиентская библиотека скомпилирована в безопасном режиме, и FALSE в противном случае

Помимо методов, класс mysqli имеет ряд открытых членов, список которых представлен в табл. 18.3.

Таблица 18.3. *Члены класса* mysqli

Член	Описание
affected_rows	Возвращает число затронутых операторами DELETE, UPDATE и REPLACE записей
client_info	Возвращает версию клиентской библиотеки в виде строки
client_version	Возвращает версию клиентской библиотеки в виде числа
errno	Возвращает номер ошибки для последней операции
error	Возвращает текстовое сообщение об ошибке для последней операции
field_count	Возвращает количество полей результирующей таблицы из последнего запроса
host_info	Возвращает адрес MySQL-сервера и тип соединения
info	Возвращает информацию о последнем запросе, количество затронутых столбцов, предупреждений и т. п.
insert_id	Возвращает первичный ключ, сгенерированный при последнем INSERT-запросе по механизму AUTO_INCREMENT
protocol_version	Возвращает версию протокола взаимодействия клиента и MySQL-сервера
server_info	Возвращает версию MySQL-сервера в виде строки
server_version	Возвращает версию MySQL-сервера в виде числа
sqlstate	Возвращает номер и текстовое сообщение о последней ошибке
thread_id	Возвращает уникальный идентификатор текущего потока
warning_count	Количество предупреждений, которое возвратил MySQL-сервер для последнего SQL-запроса

Второй класс — mysqli_stmt — применяется для формирования и обработки отложенных запросов. В листинге 18.4 приводится наиболее типичное использование отложенного запроса.

Листинг 18.4. Использование класса mysqli_stmt

```
<?php
  // Устанавливаем соединение с сервером MySQL
  $mysqli = new mysqli("localhost", "root", "", "test");
  if (mysqli_connect_errno())  exit("Ошибка установки соединения");

  // Формируем подготовленный запрос
  $query = "SELECT name, family FROM users ORDER BY name";
  if ($stmt = $mysqli->prepare($query))
  {
    // Выполняем запрос
    $stmt->execute();

    // Связываем переменные $name и $family со столбцами
    // name и family таблицы users
    $stmt->bind_result('ss', $name, $family);

    // Сохраняем результирующую таблицу
    $stmt->store_result();

    // Перемещаемся на вторую строку (нумерация начинается с 0)
    $stmt->data_seek(1);

    // Помещаем данные в переменные $name и $family
    $stmt->fetch();

    // Выводим результат
    echo "Имя : $name<br>";
    echo "Фамилия : $family<br>";

    // Освобождаем дескриптор подготовленного запроса
    $stmt->close();
  }

  // Закрываем соединение с сервером
  $mysqli->close();
?>
```

В табл. 18.4 приводится список методов класса mysqli_stmt.

Таблица 18.4. *Методы класса* `mysqli_stmt`

Метод	Описание
`bind_param(`*`types`*`, &`*`var1`*` [, &...])`	Связывает переменные, передаваемые в качестве аргумента, с переменными в SQL-запросе, которые обозначаются символом `?`. Аргумент *`types`* задает тип данных и может принимать значения `'i'`, `'d'`, `'s'` и `'b'` для целочисленных, вещественных, строковых и BLOB-полей
`bind_result(&`*`var1`*` [, &...] )`	Связывает переменные, передаваемые в качестве аргумента, со столбцами результирующей таблицы
`close()`	Освобождает дескриптор отложенного запроса
`data_seek(`*`offset`*`)`	Перемещается на строку *`offset`* в результирующей таблице (нумерация строк начинается с 0)
`execute()`	Выполняет отложенный запрос
`fetch()`	Помещает данные из текущей строки результирующей таблицы в связанные переменные, назначенные ранее при помощи метода `bind_result()`
`free_result()`	Освобождает память, выделенную под результирующую таблицу, после вызова метода данные из результирующей таблицы не доступны
`result_metadata()`	Возвращает метаданные отложенного запроса
`prepare(`*`query`*`)`	Создает отложенный запрос
`send_long_data(`*`param_nr`*`, `*`data`*`)`	Параметру *`param_nr`* присваивается значение *`data`*. Функция используется для передачи значений объемных BLOB-значений, которые превышают значение `max_allowed_packet`. Параметр, назначенный данной функцией, будет передан на сервер за несколько приемов
`reset()`	Сбрасывает отложенный запрос
`store_result()`	Сохраняет результирующую таблицу

Помимо методов, класс `mysqli_stmt` имеет ряд открытых членов, список которых представлен в табл. 18.5.

Таблица 18.5. *Члены класса* `mysqli_stmt`

Член	Описание
`affected_rows`	Возвращает число затронутых операторами `DELETE`, `UPDATE` и `REPLACE` записей

Таблица 18.5 *(окончание)*

Член	Описание
errno	Возвращает номер ошибки для последней операции
error	Возвращает текстовое сообщение об ошибке для последней операции
field_count	Возвращает количество полей результирующей таблицы из последнего запроса
id	Возвращает идентификатор запроса
insert_id	Возвращает первичный ключ, сгенерированный при последнем INSERT-запросе по механизму AUTO_INCREMENT
num_rows	Возвращает количество записей в результирующей таблице
param_count	Возвращает количество параметров в отложенном запросе
sqlstate	Возвращает номер и текстовое сообщение о последней ошибке

Последний класс mysqli_result предназначен для работы с результирующей таблицей. Методы класса представлены в табл. 18.6.

Таблица 18.6. *Методы класса* mysqli_result

Метод	Описание
close()	Освобождает память, выделенную под результирующую таблицу
data_seek(*offset*)	Перемещается на строку *offset* в результирующей таблице (нумерация строк начинается с 0)
fetch_field()	Возвращает информацию в виде объекта о столбцах результирующей таблицы
fetch_fields()	Возвращает массив объектов с информацией о столбцах результирующей таблицы
fetch_field_direct (*fieldnr*)	Возвращает объект с информацией о столбце номер *fieldnr*. Нумерация столбцов в результирующей таблице начинается с 0
fetch_array(*resulttype*)	Возвращает текущую запись результирующей таблицы в виде массива. Элементы массива соответствуют полям записи. Параметр *resulttype* определяет тип возвращаемого массива: MYSQLI_NUM — числовой массив, MYSQLI_ASSOC — ассоциативный массив, MYSQLI_BOTH — возвращает и числовой и ассоциативный массив
fetch_assoc()	Возвращает текущую запись результирующей таблицы в виде ассоциативного массива. Элементы массива соответствуют полям записи

Таблица 18.6 *(окончание)*

Метод	Описание
`fetch_object()`	Возвращает текущую запись результирующей таблицы в виде объекта
`fetch_row()`	Возвращает текущую запись в виде обычного массива
`free_result()`	Освобождает память, выделенную под результирующую таблицу
`field_seek(`*`fieldnr`*`)`	Устанавливает курсор результирующей таблицы на запись номер *`fieldnr`* (нумерация начинается с 0)

Помимо методов, класс `mysqli_result` имеет ряд открытых членов, список которых представлен в табл. 18.7.

Таблица 18.7. *Члены класса* `mysqli_result`

Член	Описание
`current_field`	Возвращает номер текущего столбца (нумерация с 0) при извлечении информации методом `fetch_field()`
`field_count`	Возвращает количество столбцов в результирующей таблице
`lengths`	Возвращает длину текущего столбца при извлечении информации методом `fetch_field()`
`num_rows`	Возвращает количество записей в результирующей таблице
`type`	Возвращает одну из констант: `MYSQLI_STORE_RESULT` или `MYSQLI_USE_RESULT`, в зависимости от того, сохранен результат или нет

В листинге 18.1 приведен пример файла config.php, осуществляющего соединение с СУБД MySQL, а в листинге 18.5 представлен аналогичный файл, созданный в процедурном стиле.

Листинг 18.5. Установка соединения с СУБД MySQL

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных на хостинге или локальной машине
  $dbname = "test";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";
```

```
  // Устанавливаем соединение с сервером MySQL
  $mysqli = new mysqli($dblocation, $dbuser, $dbpasswd, $dbname);
  if (mysqli_connect_errno()) exit("Ошибка установки соединения");

  // Устанавливаем кодировку соединения. Следует выбрать ту кодировку,
  // в которой данные будут отправляться MySQL-серверу
  $mysqli->query("SET NAMES 'cp1251'");?>
```

В следующих разделах будут рассмотрены наиболее типичные задачи, которые возникают при взаимодействии PHP-скриптов и СУБД MySQL. Большинство задач будут рассмотрены как в процедурном стиле (с использованием расширения php_mysql), так и в объектно-ориентированном стиле (с применением расширения php_mysqli).

18.3. Создание базы данных

В ранних версиях программного интерфейса СУБД MySQL имелась специальная функция для создания базы данных. Однако она была исключена, и создание новых баз данных в настоящий момент осуществляется путем выполнения оператора `CREATE DATABASE`. В листинге 18.6 база данных `dbase` создается с использованием процедурного стиля программирования.

Листинг 18.6. Создание базы данных (процедурный стиль)

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем и выполняем запрос
  $query = "CREATE DATABASE dbase";
  if(!mysql_query($query))
  {
    exit("База данных не создана ".mysql_error());
  }
?>
```

В листинге 18.7 представлен альтернативный, объектно-ориентированный вариант создания базы данных `dbase`.

Листинг 18.7. Создание базы данных (объектно-ориентированный стиль)

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");
```

```
  // Формируем и выполняем запрос
  $query = "CREATE DATABASE dbase";
  if(!$mysqli->query($query))
  {
    exit("База данных не создана ".$mysqli->error);
  }
?>
```

Функция `mysql_query()` и метод `query()` объекта класса `mysqli` в случае успеха возвращают либо `TRUE` (если запрос не возвращает результирующей таблицы), либо дескриптор результирующей таблицы, который в контексте оператора `if` также воспринимается как истинное значение (`TRUE`). Если запрос заканчивается неудачей, возвращается `FALSE`.

Следует всегда проверять, не возвращает ли СУБД MySQL ошибку выполнения последнего запроса, т. к. интерпретатор PHP (который выводит предупреждения в случае возникновения ошибки в PHP-коде) никак не сигнализирует об ошибках синтаксиса MySQL, поскольку PHP-скрипт и SQL-запросы выполняются в разных потоках и PHP-интерпретатор не имеет сведений об успешности или неудаче выполнения SQL-запроса.

18.4. Создание и заполнение таблицы

Процесс создания и заполнения таблицы схож с рассмотренным в предыдущем разделе процессом создания базы данных. Отличие заключается лишь в том, что потребуется выполнить несколько SQL-запросов.

Создадим таблицу `catalogs`, состоящую из трех полей:

- `id_catalog` — первичный ключ таблицы, снабженный атрибутом `AUTO_INCREMENT`;
- `name` — название элемента каталога;
- `putdate` — дата последней модификации каталога.

В листинге 18.8 приводится пример развертывания таблицы `catalogs` с использованием процедурного стиля.

Листинг 18.8. Создание таблицы `catalogs` (процедурный стиль)

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем запросы
  $query[] = "CREATE TABLE catalogs (
              id_catalog int(11) NOT NULL AUTO_INCREMENT,
```

```
              name tinytext NOT NULL,
              putdate datetime NOT NULL,
              PRIMARY KEY  (id_catalog)
              ) ENGINE=MyISAM";
  $query[] = "INSERT INTO catalogs
              VALUES (NULL, 'Процессоры', '2007-01-10 12:47:00')";
  $query[] = "INSERT INTO catalogs
              VALUES (NULL, 'Материнские платы', '2006-12-28 18:32:41')";
  $query[] = "INSERT INTO catalogs
              VALUES (NULL, 'Видеоадаптеры', '2007-01-10 16:48:05')";
  $query[] = "INSERT INTO catalogs
              VALUES (NULL, 'Жесткие диски', '2007-01-05 20:01:58')";
  $query[] = "INSERT INTO catalogs
             VALUES (NULL, 'Оперативная память', '2006-12-20 10:00:13')";
  // Выполняем запросы в цикле
  foreach($query as $line)
  {
    if(!mysql_query($line)) exit(mysql_error());
  }
?>
```

Как видно из листинга 18.8, для удобства все SQL-запросы помещаются в массив `$query`. Элементы массива обходятся в цикле и все друг за другом передаются функции `mysql_query()` на выполнение.

Аналогичным образом можно поступить и при использовании класса `mysqli`. Однако, как видно из табл. 18.2, класс имеет в своем составе метод `multi_query()`, который позволяет за один раз выполнить сразу несколько запросов, отделенных друг от друга точкой с запятой.

Листинг 18.9. Создание таблицы `catalogs` (объектно-ориентированный стиль)

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // В цикле выполняем запросы
  // Формируем запросы
  $query = "CREATE TABLE catalogs (
            id_catalog int(11) NOT NULL AUTO_INCREMENT,
            name tinytext NOT NULL,
            putdate datetime NOT NULL,
            PRIMARY KEY  (id_catalog)
            ) ENGINE=MyISAM;
```

```
            INSERT INTO catalogs
            VALUES (NULL, 'Процессоры', '2007-01-10 12:47:00');
            INSERT INTO catalogs
            VALUES (NULL, 'Материнские платы', '2006-12-28 18:32:41');
            INSERT INTO catalogs
            VALUES (NULL, 'Видеоадаптеры', '2007-01-10 16:48:05');
            INSERT INTO catalogs
            VALUES (NULL, 'Жесткие диски', '2007-01-05 20:01:58');
            INSERT INTO catalogs
            VALUES (NULL, 'Оперативная память', '2006-12-20 10:00:13')";
  if(!$mysqli->multi_query($query))
  {
    exit("Таблицу не удалось развернуть ".$mysqli->error);
  }
?>
```

Выполнение нескольких запросов за один прием имеет большое значение при развертывании базы данных из текстового дампа. Для того чтобы развернуть такой дамп при помощи функции `mysql_query()` или метода `query()` объекта класса `mysqli`, необходимо предварительно разбить содержимое дампа на отдельные запросы, что может быть затруднительно, если точка с запятой встречается в строковых значениях таблицы.

18.5. Заполнение связанных таблиц

Еще один интересный случай представляет собой заполнение связанных таблиц. Пусть имеется таблица `news`, предназначенная для хранения новостных позиций и состоящая из трех полей:

- `id_news` — первичный ключ, снабженный атрибутом `AUTO_INCREMENT`;
- `name` — название новостной позиции;
- `putdate` — дата размещения новости.

С таблицей `news` связана таблица `news_contents`, предназначенная для хранения текста новости и также состоящая из трех полей:

- `id_content` — первичный ключ, снабженный атрибутом `AUTO_INCREMENT`;
- `content` — содержимое новостной позиции;
- `id_news` — вторичный ключ, содержит значение полей `id_news` из таблицы `news`, связывая новостной блок и текст новости.

В листинге 18.10 представлены операторы `CREATE TABLE`, которые создают таблицы `news` и `news_contents`.

Листинг 18.10. Таблицы `news` и `news_contents`

```
CREATE TABLE news (
  id_news INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  putdate DATETIME NOT NULL,
  PRIMARY KEY (id_news)
);
CREATE TABLE news_contents (
  id_content INT(11) NOT NULL AUTO_INCREMENT,
  content TEXT NOT NULL,
  id_news INT(11) NOT NULL,
  PRIMARY KEY (id_content)
);
```

На рис. 18.1 представлена HTML-форма, которая позволяет добавлять название и текст новостной позиции в базу данных.

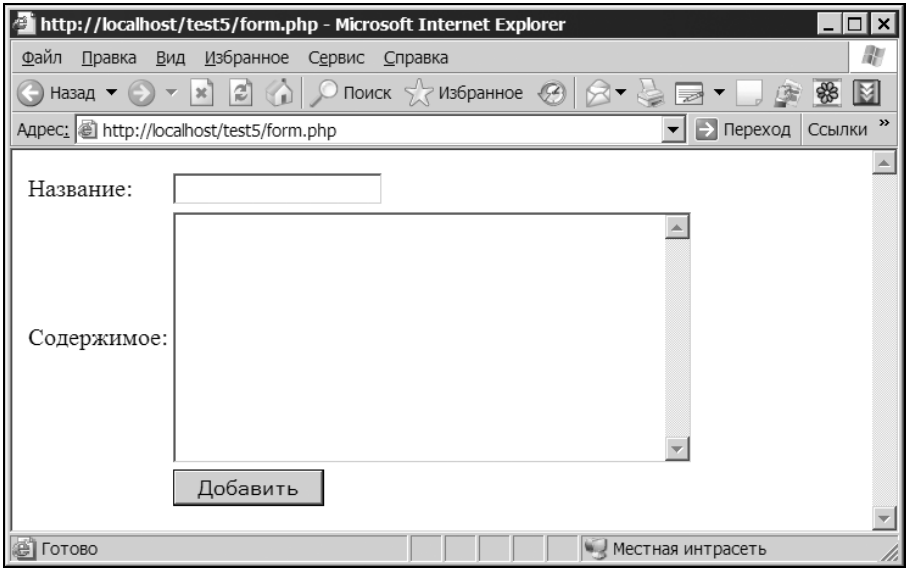


Рис. 18.1. HTML-форма для добавления новостной позиции в базу данных

HTML-код, воссоздающий форму, представленную на рис. 18.1, приведен в листинге 18.11.

Листинг 18.11. HTML-форма для добавления новостной позиции в базу данных

```
<table>
<form action=addnews.php method=POST>
```

```
  <tr>
    <td>Название:</td>
    <td><input type=text name=name></td>
  </tr>
  <tr>
    <td>Содержимое:</td>
    <td><textarea name=content rows=10 cols=40></textarea></td>
  </tr>
  <tr>
    <td></td>
    <td><input type=submit value=Добавить></td>
  </tr>
</form>
</table>
```

Как видно из листинга 18.11, в качестве обработчика HTML-формы назначен файл addnews.php. В листинге 18.12 представлен обработчик HTML-формы, который осуществляет вставку новостной позиции в таблицы news и news_contents.

Листинг 18.12. Добавление новостной позиции в базу данных (процедурный вариант)

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Проверяем, заполнены ли поля HTML-формы
  if(empty($_POST['name'])) exit('Не заполнено поле "Название"');
  if(empty($_POST['content'])) exit('Не заполнено поле "Содержимое"');

  // Экранируем специальные символы
  if (!get_magic_quotes_gpc())
  {
    $_POST['name'] = mysql_real_escape_string($_POST['name']);
    $_POST['content'] = mysql_real_escape_string($_POST['content']);
  }

  // Добавляем новостную позицию в таблицу news
  $query = "INSERT INTO news VALUES (NULL, '$_POST[name]', NOW())";
  if(!mysql_query($query))
  {
    exit("Ошибка добавления новостной позиции ".mysql_error());
  }
```

```
  // Получаем только что сгенерированный идентификатор id_news
  $id_news = mysql_insert_id();

  // Вставляем содержимое новостной позиции в таблицу news_contents.
  // Формируем запросы
  $query = "INSERT INTO news_contents
            VALUES (NULL, '$_POST[content]', $id_news)";
  if(!mysql_query($query))
  {
    exit("Ошибка добавления новостной позиции ".mysql_error());
  }

  // Осуществляем переадресацию на главную страницу
  echo "<HTML><HEAD>
        <META HTTP-EQUIV='Refresh' CONTENT='0; URL=form.php'>
        </HEAD></HTML>";
?>
```

Так как данные передаются методом `POST`, то содержимое полей `name` и `content` попадает в элементы суперглобального массива `$_POST['name']` и `$_POST['name']` соответственно. В начале обработчика производится проверка на правильность заполнения полей формы — если хоть одно из полей остается незаполненным, процесс добавления информации приостанавливается с выдачей соответствующего предупреждения. Содержит ли строка хоть один символ, можно проверить при помощи функции `empty()`, которая возвращает `TRUE`, если строка пустая, и `FALSE`, если строка содержит хоть один символ.

После этого при помощи функции `mysql_real_escape_string()` экранируются специальные символы в строковых переменных. Если строка содержит символ `'`, то синтаксис SQL-оператора может быть нарушен. Такой символ следует заменить на последовательность `\'`. Именно для этого и предназначена функция `mysql_real_escape_string()`.

Замечание

На ряде серверов включен режим "магических кавычек". В этом режиме экранирование специальных символов в `POST`-, `GET`- и `COOKIE`-данных осуществляется автоматически, и повторно экранировать данные не требуется. Выяснить, включен режим "магических кавычек" или нет, можно при помощи функции `get_magic_quotes_gpc()`, которая возвращает `TRUE`, если режим включен, и `FALSE` в противном случае.

Основной особенностью добавления данных в связанные таблицы заключается в том, что для добавления записи во вторую таблицу `news_content` необходимо знать первичный ключ `id_news`, который был сгенерирован по механиз-

му AUTO_INCREMENT для только что созданной записи в таблице news. Для этого предназначена функция mysql_insert_id().

После того как записи добавлены, следует вернуться на главную страницу HTML-формы или страницу, где выводятся новостные позиции.

Рассмотренная выше схема разделяет HTML-форму и обработчик, используя для их представления два разных файла. Такой способ удобен для восприятия, но не всегда удобен в реализации. Главным недостатком такого подхода является то, что обработка данных осуществляется на второй странице, где отсутствует сама HTML-форма. Поэтому часто прибегают к приему, когда HTML-форма и обработчик находятся в одном файле (листинг 18.13).

Листинг 18.13. Размещение HTML-формы и обработчика в одном файле

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Обработчик HTML-формы
  $error = array();
  if(!empty($_POST))
  {
    // Проверяем, заполнены ли поля HTML-формы
    if(empty($_POST['name'])) $error[] = 'Не заполнено поле "Название"';
    if(empty($_POST['content'])) $error[] = 'Не заполнено поле "Содержимое"';

    // Если ошибок нет — добавляем запись
    if(empty($error))
    {
      // Экранируем специальные символы
      if (!get_magic_quotes_gpc())
      {
        $_POST['name'] = mysql_real_escape_string($_POST['name']);
        $_POST['content'] = mysql_real_escape_string($_POST['content']);
      }

      // Добавляем новостную позицию в таблицу news
      $query = "INSERT INTO news VALUES (NULL, '$_POST[name]', NOW())";
      if(!mysql_query($query))
      {
        exit("Ошибка добавления новостной позиции ".mysql_error());
      }

      // Получаем только что сгенерированный идентификатор id_news
      $id_news = mysql_insert_id();
```

```
      // Вставляем содержимое новостной позиции в таблицу news_contents.
      // Формируем запросы
      $query = "INSERT INTO news_contents
                VALUES (NULL, '$_POST[content]', $id_news)";
      if(!mysql_query($query))
      {
        exit("Ошибка добавления новостной позиции ".mysql_error());
      }

      // Осуществляем переадресацию на главную страницу
      echo "<HTML><HEAD>
            <META HTTP-EQUIV='Refresh' CONTENT='0; URL=form.php'>
            </HEAD></HTML>";
   }
 }
?>
<table>
<form method=POST>
  <tr>
    <td></td>
    <td>
      <span style="color:red">
      <?php
        // Выводим ошибки, если они имеются
        if(!empty($error))
        {
          foreach($error as $line)
          {
            echo $line."<br>";
          }
        }
      ?>
      </span>
    </td>
  </tr>
  <tr>
    <td>Название:</td>
    <td><input type=text name=name value='<?= htmlspecialchars($_POST['name'],
ENT_QUOTES); ?>'></td>
  </tr>
  <tr>
    <td>Содержимое:</td>
    <td><textarea name=content rows=10 cols=40><?=
htmlspecialchars($_POST['content'], ENT_QUOTES); ?></textarea></td>
  </tr>
```

```
  <tr>
    <td></td>
    <td><input type=submit value=Добавить></td>
  </tr>
</form>
</table>
```

В листинге 18.13 сообщения об ошибках помещаются в массив `$error`: если он пуст — обработчик начинает работу, если массив не пуст, выводятся HTML-форма с ранее введенными полями и содержимое массива `$error`. При выводе информации в HTML-форму необходимо использовать функцию `htmlspecialchars()`, которая заменит угловые скобки и другие символы эквивалентами. Это предотвратит искажение данных.

В листинге 18.14 представлен обработчик addnews.php в объектно-ориентированной форме.

Листинг 18.14. Добавление новостной позиции в базу данных (объектно-ориентированный вариант)

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Проверяем, заполнены ли поля HTML-формы
  if(empty($_POST['name'])) exit('Не заполнено поле "Название"');
  if(empty($_POST['content'])) exit('Не заполнено поле "Содержимое"');

  // Экранируем специальные символы
  if (!get_magic_quotes_gpc())
  {
    $_POST['name'] = $mysqli->real_escape_string($_POST['name']);
    $_POST['content'] = $mysqli->real_escape_string($_POST['content']);
  }

  // Добавляем новостную позицию в таблицу news
  $query = "INSERT INTO news VALUES (NULL, '$_POST[name]', NOW())";
  if(!$mysqli->query($query))
  {
    exit("Ошибка добавления новостной позиции ".$mysqli->error);
  }

  // Получаем только что сгенерированный идентификатор id_news
  $id_news = $mysqli->insert_id;
```

```
  // Вставляем содержимое новостной позиции в таблицу news_contents.
  // Формируем запросы
  $query = "INSERT INTO news_contents
            VALUES (NULL, '$_POST[content]', $id_news)";
  if(!$mysqli->query($query))
  {
    exit("Ошибка добавления новостной позиции ".$mysqli->error);
  }

  // Осуществляем переадресацию на главную страницу
  echo "<HTML><HEAD>
        <META HTTP-EQUIV='Refresh' CONTENT='0; URL=form.php'>
        </HEAD></HTML>";
?>
```

Как видно, структура листинга 18.14 полностью совпадает со структурой листинга 18.12, только вместо API-функций PHP для доступа и управления СУБД MySQL используется объект класса `mysqli`.

18.6. Вывод данных

Еще одной часто встречающейся задачей является вывод содержимого таблицы из базы данных. Пусть имеется таблица `catalogs`, содержимое которой представлено в листинге 18.15.

Листинг 18.15. Таблица `catalogs`

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  PRIMARY KEY (id_catalog)
);
INSERT INTO catalogs VALUES (NULL, 'Процессоры'),
                            (NULL, 'Материнские платы'),
                            (NULL, 'Видеоадаптеры'),
                            (NULL, 'Жесткие диски'),
                            (NULL, 'Оперативная память');
```

В листинге 18.16 представлен скрипт, выводящий содержимое таблицы в окно браузера.

Листинг 18.16. Вывод содержимого таблицы `catalogs` (процедурный вариант)

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");
```

```
  // Формируем SELECT-запрос
  $query = "SELECT * FROM catalogs";
  $cat = mysql_query($query);
  if(!$cat) exit("Ошибка обращения к каталогу");

  // Если имеется хотя бы одна запись, выводим список
  if(mysql_num_rows($cat))
  {
    while($catalog = mysql_fetch_array($cat))
    {
      echo $catalog['name']."<br>";
    }
  }
?>
```

Как видно из листинга 18.16, функция `mysql_query()` возвращает дескриптор результирующей таблицы `$cat`, который используется для доступа к данным, возвращаемым `SELECT`-запросом. Функция `mysql_num_rows()` возвращает количество записей в результирующей таблице, если это значение больше нуля, содержимое таблицы извлекается при помощи функции `mysql_fetch_array()`. За один раз функция извлекает из результирующей таблицы одну строку. Повторный вызов функции `mysql_fetch_array()` приводит к извлечению следующего столбца и т. д. Поэтому вызов функции в цикле `while()` приводит к последовательному извлечению строк результирующей таблицы до тех пор, пока они не будут исчерпаны.

Функция `mysql_fetch_array()` имеет следующий синтаксис:

```
array mysql_fetch_array(resource result [, int result_type])
```

В качестве первого аргумента *`result`* функция принимает дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`. Второй необязательный параметр может принимать три значения:

- `MYSQL_ASSOC` — возврат результата работы в виде ассоциативного массива;
- `MYSQL_NUM` — возврат результата работы в виде численного массива;
- `MYSQL_BOTH` — возврат результата работы в виде массива, содержащего как численные индексы, так и ассоциативные индексы.

Замечание

По умолчанию второй аргумент принимает значение `MYSQL_BOTH`.

Замечание

Режимы работы функции `mysql_fetch_array()`, принимающей в качестве второго аргумента константы `MYSQL_ASSOC` и `MYSQL_NUM`, аналогичны функциям `mysql_fetch_assoc()` и `mysql_fetch_row()` соответственно.

При возврате ассоциативного массива в качестве индексов выступают имена столбцов результирующей таблицы, из которой производится выборка.

В листинге 18.17 приводится альтернативный, объектно-ориентированный вариант вывода содержимого таблицы `catalogs`.

Листинг 18.17. Вывод содержимого таблицы `catalogs` (объектно-ориентированный вариант)

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем SELECT-запрос
  $query = "SELECT * FROM catalogs";
  $cat = $mysqli->query($query);
  if(!$cat) exit("Ошибка обращения к каталогу");

  // Если имеется хотя бы одна запись, выводим список
  if($cat->num_rows)
  {
    while($catalog = $cat->fetch_array())
    {
      echo $catalog['name']."<br>";
    }
  }
?>
```

Как видно из листинга 18.17, метод `query()` объекта `mysqli` возвращает объект результирующей таблицы, порядок работы с которым аналогичен процедурному варианту, за исключением того факта, что вместо независимых функций вызываются методы объекта.

18.7. Повторное чтение результирующей таблицы

Следует отметить, что вызов функции `mysql_fetch_array()` или метода `fetch_array()` в цикле `while` приводит к тому, что курсор останавливается на последней записи результирующей таблицы и повторная попытка извлечения записей заканчивается неудачей. Для того чтобы можно было успешно прочитать результирующую таблицу повторно, не посылая `SELECT`-запрос серверу MySQL, необходимо вернуть курсор в начало результирующей таблицы. Для

этого предназначена функция `mysql_data_seek()`, которая имеет следующий синтаксис:

```
bool mysql_data_seek(result, row_number)
```

Функция для результирующей таблицы *result* устанавливает внутренний курсор на строку *row_number* (нумерация строк начинается с 0). В случае успеха функция возвращает `TRUE` и `FALSE` в случае неудачи.

Вернемся к задаче вывода содержимого таблицы `catalogs`, которая была рассмотрена в предыдущем разделе. Пусть необходимо подсчитать количество символов в выводимых строках, после чего вывести содержимое результирующей таблицы (листинг 18.18).

Листинг 18.18. Повторное считывание результирующей таблицы (процедурный вариант)

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем SELECT-запрос
  $query = "SELECT * FROM catalogs";

  $cat = mysql_query($query);
  if(!$cat) exit("Ошибка обращения к каталогу");
  // Если имеется хотя бы одна запись, выводим список
  if(mysql_num_rows($cat))
  {
    // Подсчитываем количество символов во
    // всех строках таблицы
    $count = 0;
    while($catalog = mysql_fetch_array($cat))
    {
      $count += strlen($catalog['name']);
    }
    echo "Общая длина всех строк ".$count."<br>";

    // Устанавливаем внутренний курсор в
    // начало результирующей таблицы
    mysql_data_seek($cat, 0);

    // Повторно считываем результирующую таблицу
    while($catalog = mysql_fetch_array($cat))
    {
      echo $catalog['name']."<br>";
    }
  }
?>
```

Как видно из листинга 18.18, для того чтобы осуществить повторное считывание результирующей таблицы, необходимо установить курсор на нулевую запись при помощи функции `mysql_data_seek()`.

В листинге 18.19 представлен объектно-ориентированный вариант скрипта из листинга 18.18.

Листинг 18.19. Повторное считывание результирующей таблицы (объектно-ориентированный вариант)

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем SELECT-запрос
  $query = "SELECT * FROM catalogs";

  $cat = $mysqli->query($query);
  if(!$cat) exit("Ошибка обращения к каталогу");
  // Если имеется хотя бы одна запись, выводим список
  if($cat->num_rows)
  {
    // Подсчитываем количество символов во
    // всех строках таблицы
    $count = 0;
    while($catalog = $cat->fetch_array())
    {
      $count += strlen($catalog['name']);
    }
    echo "Общая длина всех строк ".$count."<br>";

    // Устанавливаем внутренний курсор в
    // начало результирующей таблицы
    $cat->data_seek(0);

    // Повторно считываем результирующую таблицу
    while($catalog = $cat->fetch_array())
    {
      echo $catalog['name']."<br>";
    }
  }
?>
```

Результатом работы скриптов из листингов 18.18 и 18.19 будут следующие строки:

```
Общая длина всех строк 71
Процессоры
Материнские платы
Видеоадаптеры
Жесткие диски
Оперативная память
```

18.8. Количество строк в таблице

В предыдущих разделах для определения количество строк в результирующей таблице использовалась функция `mysql_num_rows()` или свойство `num_rows` объекта результирующей таблицы. Если `SELECT`-выборка не имеет ограничений и таблица выбирается полностью, можно использовать данные способы для оценки строк в таблице. Однако зачастую на одну страницу выводится лишь часть таблицы, в то время как для организации навигации требуется знать общее количество строк во всей таблице. Если таблица объемная, полная выборка таблицы не приемлема ни по скорости, ни по объему памяти, которая необходима для результирующей таблицы. В этом случае разумнее прибегнуть к функции `COUNT(*)`, которая подсчитывает количество строк в таблице. В листинге 18.20 представлен скрипт, который подсчитывает количество строк в таблице `catalogs` (см. листинг 18.15).

Листинг 18.20. Подсчет количества строк в таблице (процедурный вариант)

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем SELECT-запрос
  $query = "SELECT COUNT(*) FROM catalogs";

  $cat = mysql_query($query);
  if(!$cat) exit("Ошибка обращения к каталогу");
  echo mysql_result($cat, 0);
?>
```

Так как запрос возвращает всего одно значение, для его извлечения из результирующей таблицы разумнее использовать функцию `mysql_result()`. Синтаксис функции представлен ниже:

```
mixed mysql_result(resource result, int row [, mixed field])
```

В качестве первого аргумента *`result`* функция принимает дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`. Второй аргумент

row представляет собой номер столбца, который необходимо вернуть. Третий необязательный параметр *field* — это имя поля таблицы.

В листинге 18.21 представлен объектно-ориентированный вариант, выводящий количество строк таблицы catalogs.

Листинг 18.21. Подсчет количества строк в таблице (объектно-ориентированный вариант)

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем SELECT-запрос
  $query = "SELECT COUNT(*) AS total FROM catalogs";

  $cat = $mysqli->query($query);
  if(!$cat) exit("Ошибка обращения к каталогу");

  $result = $cat->fetch_object();
  echo $result->total;
?>
```

Объектно-ориентированного аналога для функции mysql_result() в объекте результирующей таблице нет. В листинге 18.21 при помощи метода fetch_object() строка результирующей таблицы возвращается в виде объекта, имена членов которого совпадают с именами полей в результирующей таблице. Для удобства единственный столбец в результирующей таблице переименовывается в total при помощи оператора AS.

18.9. Вывод результатов в несколько столбцов

Пусть имеется таблица catalogs (см. листинг 18.15), элементы которой необходимо вывести в таблице с заранее заданным количеством столбцов. При формировании таблицы допустимы два варианта вывода элементов: горизонтальный (рис. 18.2) и вертикальный (рис. 18.3).

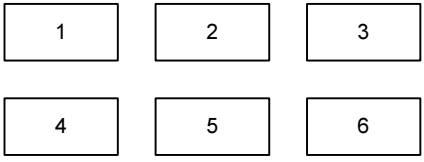


Рис. 18.2. Горизонтальное расположение элементов таблицы

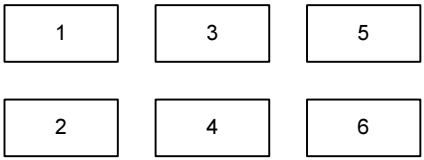


Рис. 18.3. Вертикальное расположение элементов таблицы

В листинге 18.22 приводится возможная реализация горизонтального вывода содержимого таблицы `catalogs`.

Листинг 18.22. Горизонтальный вывод элементов таблицы

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
  // Количество столбцов в таблице
  $row_count = 3;

  // Формируем и выполняем SQL-запрос
  $query = "SELECT name FROM catalogs ORDER BY name";
  $tbl = mysql_query($query);
  if(!$tbl) exit("Ошибка обращения к таблице — ".mysql_error());
  // Если в результирующей таблице имеется хотя бы
  // один элемент — выводим его
  if(mysql_num_rows($tbl))
  {
    // Количество строк таблицы
    $total = (int)(mysql_num_rows($tbl)/$row_count);
    if(mysql_num_rows($tbl)%$row_count) $total++;
    // Выводим HTML-таблицу
    echo "<table border=1 cellpadding=4>";
    for($i = 0; $i < $total; $i++)
    {
      // Выводим ряд
      echo "<tr>";
      for($j = 0; $j < $row_count; $j++)
      {
        // Выводим ячейку
        echo "<td>".@mysql_result($tbl, $j + $i*$total)." </td>";
      }
      echo "</tr>";
    }
    echo "</table>";
  }
?>
```

В листинге 18.23 представлен скрипт, реализующий вертикальный вывод элементов таблицы `catalogs`.

Листинг 18.23. Вертикальный вывод элементов таблицы

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
  // Количество столбцов в таблице
  $row_count = 3;

  // Формируем и выполняем SQL-запрос
  $query = "SELECT name FROM catalogs ORDER BY name";
  $tbl = mysql_query($query);
  if(!$tbl) exit("Ошибка обращения к таблице — ".mysql_error());
  // Если в результирующей таблице имеется хотя бы
  // один элемент — выводим его
  if(mysql_num_rows($tbl))
  {
    // Количество ячеек в одном столбце
    $total = (int)(mysql_num_rows($tbl)/$row_count);
    if(mysql_num_rows($tbl)%$row_count) $total++;
    // Выводим HTML-таблицу
    echo "<table border=1>";
    for($i = 0; $i < $total; $i++)
    {
      // Выводим ряд
      echo "<tr>";
      for($j = 0; $j < $row_count; $j++)
      {
        // Выводим ячейку
        echo "<td>".@mysql_result($tbl, $j*$total + $i)." </td>";
      }
      echo "</tr>";
    }
    echo "</table>";
  }
?>
```

Как видно из листингов 18.22 и 18.23, перед функцией `mysql_result()` указан символ подавления вывода предупреждения `@`. Данный символ тут необходим, т. к. в большинстве случаев количество элементов в таблице MySQL не будет точно соответствовать количеству элементов в HTML-таблице. В результате этого функция `mysql_result()` будет обращаться к несуществующим

элементам результирующей таблицы, что не является критичным при программировании на PHP, однако приводит к выводу предупреждений, которые искажают таблицу.

18.10. Постраничная навигация

Одним из наиболее часто применяемых приемов при программировании приложений с использованием баз данных является постраничная навигация. Таблицы, как правило, хранят большое количество записей, и вывод всех их на одной странице не удобен для пользователя, которому проще анализировать небольшой объем данных, и сильно нагружает сервер баз данных, которому при каждом обращении клиента необходимо переправлять большой объем данных. Поэтому на одной странице выводится ограниченное количество записей с предоставлением ссылок на последующие и предыдущие записи (рис. 18.4).

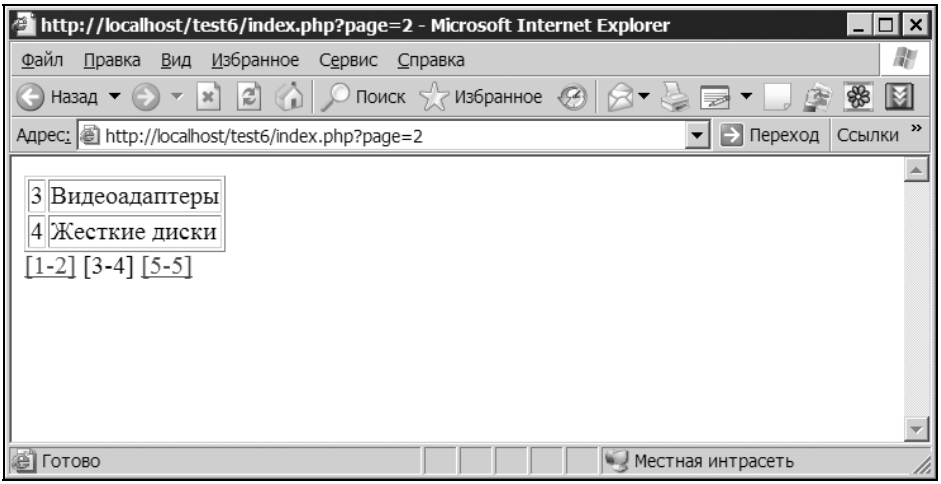


Рис. 18.4. Постраничная навигация

Реализацию постраничной навигации средствами MySQL удобнее реализовывать при помощи конструкции `LIMIT N, M`, позволяющей извлечь *M* записей, начиная с позиции *N* (листинг 18.24).

Листинг 18.24. Постраничная навигация (процедурный вариант)

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
```

```
// Число позиций на странице
$pnumber = 2;

// Проверяем, передан ли номер текущей страницы
if(isset($_GET['page'])) $page = $_GET['page'];
else $page = 1;

// Начальная позиция
$start = ($page — 1)*$pnumber;

// Формируем запрос на извлечение списка
// элементов каталога
$query = "SELECT * FROM catalogs
          LIMIT $start, $pnumber";
$cat = mysql_query($query);
if(!$cat) exit(mysql_error());
if(mysql_num_rows($cat))
{
  echo "<table border=1>";
  while($catalog = mysql_fetch_array($cat))
  {
    // Формируем строку таблицы
    echo "<tr><td>$catalog[id_catalog]</td><td>$catalog[name]</td>";
  }
  echo "</table>";
}

// Количество страниц
$query = "SELECT COUNT(*) FROM catalogs";
$tot = mysql_query($query);
if(!$tot) exit(mysql_error());
$total = mysql_result($tot,0);
$number = (int)($total/$pnumber);
if((float)($total/$pnumber) — $number != 0) $number++;

// Постраничная навигация
for($i = 1; $i <= $number; $i++)
{
  if($i != $number)
  {
    if($page == $i)
    {
      echo "[".(($i — 1)*$pnumber + 1)."-".$i*$pnumber."] ";
    }
```

```
      else
      {
        echo "<a href=$_SERVER[PHP_SELF]?page=".$i.">[".
                      (($i — 1)*$pnumber + 1)."-".
                      $i*$pnumber."]</a> ";
      }
    }
    else
    {
      if($page == $i)
      {
        echo "[".(($i — 1)*$pnumber + 1)."-".($total — 1)."] ";
      }
      else
      {
        echo "<a href=$_SERVER[PHP_SELF]?page=".$i.">[".
                      (($i — 1)*$pnumber + 1)."-".
                      $total."]</a> ";
      }
    }
  }
?>
```

Как видно из листинга 18.24, номер текущей страницы определяется `GET`-параметром `page`. С помощью номера текущей страницы `$_GET['page']` и количества позиций на одной странице `$pnumber` вычисляется начало интервала выборки `$start`. Конструкция `LIMIT` формируется динамически, используя вычисляемое значение `$start` в качестве первого параметра и количество позиций на странице `$pnumber` в качестве второго параметра.

В листинге 18.25 представлен объектно-ориентированный вариант постраничной навигации.

Листинг 18.25. Постраничная навигация (объектно-ориентированный вариант)

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
  // Число позиций на странице
  $pnumber = 2;

  // Проверяем, передан ли номер текущей страницы
  if(isset($_GET['page'])) $page = $_GET['page'];
  else $page = 1;
```

```
// Начальная позиция
$start = ($page — 1)*$pnumber;

// Формируем запрос на извлечение списка
// элементов каталога
$query = "SELECT * FROM catalogs
          LIMIT $start, $pnumber";
$cat = $mysqli->query($query);
if(!$cat) exit($mysqli->error);
if($cat->num_rows)
{
  echo "<table border=1>";
  while($catalog = $cat->fetch_array())
  {
    // Формируем строку таблицы
    echo "<tr><td>$catalog[id_catalog]</td><td>$catalog[name]</td>";
  }
  echo "</table>";
}

// Количество страниц
$query = "SELECT COUNT(*) AS total FROM catalogs";
$tot = $mysqli->query($query);
if(!$tot) exit($mysqli->error);
$result = $tot->fetch_array();
$total = $result['total'];
$number = (int)($total/$pnumber);
if((float)($total/$pnumber) — $number != 0) $number++;

// Постраничная навигация
for($i = 1; $i <= $number; $i++)
{
  if($i != $number)
  {
    if($page == $i)
    {
      echo "[".(($i — 1)*$pnumber + 1)."-".$i*$pnumber."] ";
    }
    else
    {
      echo "<a href=$_SERVER[PHP_SELF]?page=".$i.">[".
                    (($i — 1)*$pnumber + 1)."-".
                    $i*$pnumber."]</a> ";
    }
  }
```

```
    else
    {
      if($page == $i)
      {
        echo "[".(($i - 1)*$pnumber + 1)."-".($total - 1)."] ";
      }
      else
      {
        echo "<a href=$_SERVER[PHP_SELF]?page=".$i.">[".
                      (($i - 1)*$pnumber + 1)."-".
                      $total."]</a> ";
      }
    }
  }
?>
```

18.11. Удаление данных

При удалении данных часто прибегают к флажкам — можно отметить сразу несколько позиций, которые подвергаются удалению (рис. 18.5).

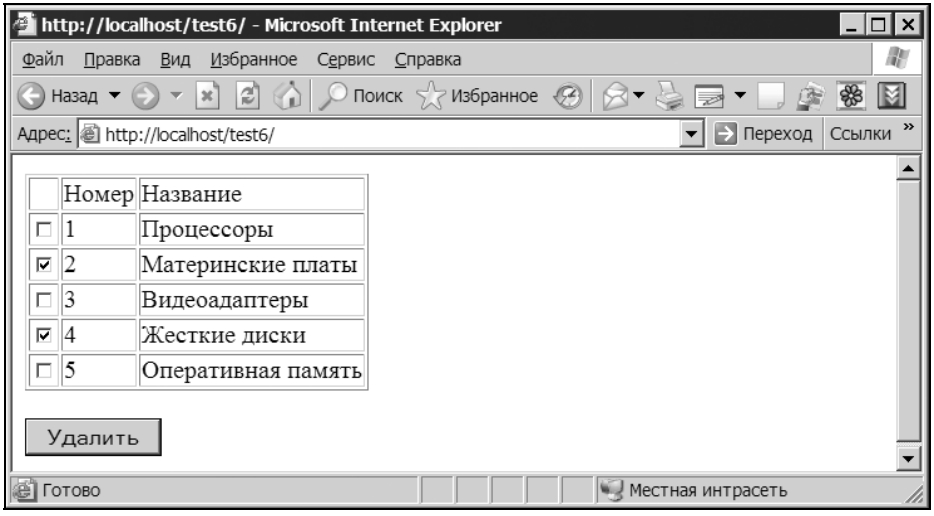


Рис. 18.5. Web-интерфейс для удаления нескольких записей таблицы

В листинге 18.26 представлен скрипт, позволяющий удалять сразу несколько записей, которые пользователь отмечает флажками.

Листинг 18.26. Удаление нескольких записей таблицы (процедурный вариант)

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Если суперглобальный массив $_POST не пуст,
  // производим обработку запроса
  if(!empty($_POST))
  {
    // Для удаления позиций необходимо сформировать запрос вида
    // DELETE FROM catalogs WHERE id_catalog IN (4, 6, 8, ..., 20),
    // где цифры в скобках являются элементами
    // массива $_POST['catalog'][]
    $temp = array();
    foreach($_POST['catalog'] as $id_catalog)
    {
      // Проверяем, является ли переменная $id_catalog числом
      if(preg_match("|^[\d]+$|",$id_catalog))
      {
        $temp[] = $id_catalog;
      }
    }
    // Формируем и выполняем запрос на удаление
    // нескольких позиций
    $query = "DELETE FROM catalogs
              WHERE id_catalog IN (".implode(",",$temp).")";
    if(mysql_query($query))
    {
      echo "<HTML><HEAD>
          <META HTTP-EQUIV='Refresh' CONTENT='0; URL=$_SERVER[PHP_SELF]'>
            </HEAD></HTML>";
      exit();
    }
  }

  // Выводим список элементов каталога
  $query = "SELECT * FROM catalogs
            ORDER BY id_catalog";
  $cat = mysql_query($query);
  if(!$cat) exit(mysql_error());
  // Если имеется хотя бы одна позиция — выводим ее
  if(mysql_num_rows($cat) > 0)
  {
    echo "<form method=post>";
```

```
    echo "<table border=1>
            <tr>
              <td> </td>
              <td>Номер</td>
              <td>Название</td>
            </tr>";
    $i = 0;
    while($catalog = mysql_fetch_array($cat))
    {
      echo "<tr>
              <td><input type=checkbox name=catalog[]
                         value=$catalog[id_catalog]></td>
              <td>$catalog[id_catalog]</td>
              <td>$catalog[name]</td>
            </tr>";
      $i++;
    }
    echo "</table>";
    echo "</br><input type=submit name=send value=Удалить>";
    echo "</form>";
  }
?>
```

В листинге 18.27 представлен объектно-ориентированный вариант удаления нескольких позиций из базы данных.

Листинг 18.27. Удаление нескольких записей таблицы (объектно-ориентированный вариант)

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Если суперглобальный массив $_POST не пуст,
  // производим обработку запроса
  if(!empty($_POST))
  {
    // Для удаления позиций необходимо сформировать запрос вида
    // DELETE FROM catalogs WHERE id_catalog IN (4, 6, 8, ..., 20),
    // где цифры в скобках являются элементами
    // массива $_POST['catalog'][]
    $temp = array();
    foreach($_POST['catalog'] as $id_catalog)
    {
      // Проверяем, является ли переменная $id_catalog числом
      if(preg_match("|^[\d]+$|",$id_catalog))
```

```
      {
        $temp[] = $id_catalog;
      }
    }
    // Формируем и выполняем запрос на удаление
    // нескольких позиций
    $query = "DELETE FROM catalogs
              WHERE id_catalog IN (".implode(",",$temp).")";
    if($mysqli->query($query))
    {
      echo "<HTML><HEAD>
          <META HTTP-EQUIV='Refresh' CONTENT='0; URL=$_SERVER[PHP_SELF]'>
            </HEAD></HTML>";
      exit();
    }
  }

  // Выводим список элементов каталога
  $query = "SELECT * FROM catalogs
            ORDER BY id_catalog";
  $cat = $mysqli->query($query);
  if(!$cat) exit($mysqli->error);
  // Если имеется хотя бы одна позиция — выводим ее
  if($cat->num_rows)
  {
    echo "<form method=post>";
    echo "<table border=1>
            <tr>
              <td> </td>
              <td>Номер</td>
              <td>Название</td>
            </tr>";
    $i = 0;
    while($catalog = $cat->fetch_array())
    {
      echo "<tr>
              <td><input type=checkbox name=catalog[]
                         value=$catalog[id_catalog]></td>
              <td>$catalog[id_catalog]</td>
              <td>$catalog[name]</td>
            </tr>";
      $i++;
    }
    echo "</table>";
    echo "</br><input type=submit name=send value=Удалить>";
    echo "</form>";
  }
?>
```

18.12. Сортировка

При выводе содержимого таблицы для удобства восприятия информации часто требуется организовывать сортировку по столбцам. Выведем содержимое таблицы `catalogs` (см. листинг 18.15) в таблицу, заголовки которой будут представлять собой гиперссылки, переход по которым осуществлял бы сортировку выбранного столбца. Повторный переход по гиперссылке должен приводить к сортировке данных в обратном порядке (рис. 18.6).

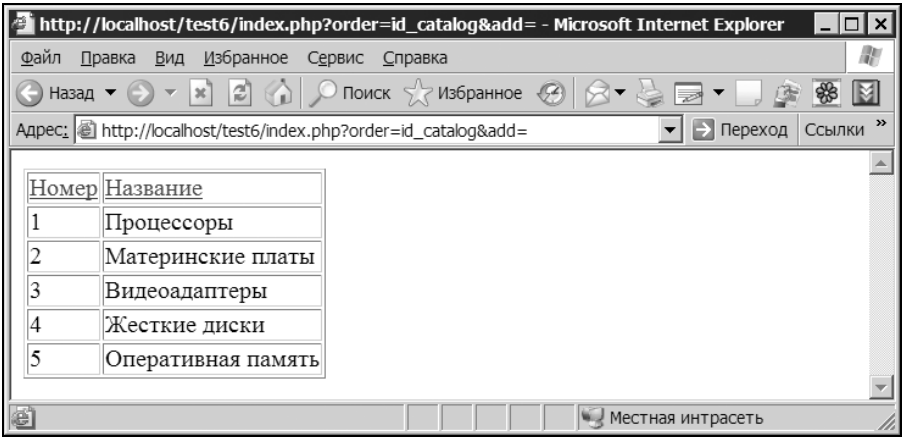


Рис. 18.6. Таблица с возможностью сортировки столбцов

Сортировка результатов SQL-запроса производится при помощи конструкции `ORDER BY`, после которой указывается имя столбца, подвергающегося сортировке. Если указать необязательное ключевое слово `DESC`, сортировка будет производиться в обратном порядке. Таким образом, задача сводится к формированию динамического SQL-запроса, в конструкцию `ORDER BY` которого подставлялись бы имена выбранных столбцов. Для этого будем использовать запрос вида:

```
SELECT * FROM catalogs
         ORDER BY $order $desc
```

Здесь переменная `$order` будет содержать имя столбца, а `$desc` принимать либо пустое значение, либо ключевое слово `DESC` в зависимости от того, какой вид сортировки выбран (листинг 18.28).

Листинг 18.28. Сортировка столбцов (процедурный вариант)

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
```

```
  // Проверяем параметры, переданные скрипту
  $order = "name";
  if($_GET['order'] == 'name')  $order = "name";
  if($_GET['order'] == 'id_catalog') $order = "id_catalog";
  if($_GET['add']   == 'desc')
  {
    $desc = "DESC";
    $add  = "";
  }
  else
  {
    $desc = "";
    $add  = "desc";
  }

  // Выводим элементы каталога
  $query = "SELECT * FROM catalogs
            ORDER BY $order $desc";
  $cat = mysql_query($query);
  if(!$cat) exit(mysql_error());
  if(mysql_num_rows($cat) > 0)
  {
    echo "<table border=1>
            <tr>
  <td><a href=$_SERVER[PHP_SELF]?order=id_catalog&add=$add>Номер</a></td>
  <td><a href=$_SERVER[PHP_SELF]?order=name&add=$add>Название</a></td>
            </tr>";
    while($catalog = mysql_fetch_array($cat))
    {
      echo "<tr>
              <td>$catalog[id_catalog]</td>
              <td>$catalog[name]</td>
            </tr>";
    }
    echo "</table>";
  }
?>
```

В начале скрипта `GET`-параметр `order` подвергается проверке, и если его значение равно одному из столбцов, присутствующих в таблице, в переменную `$order` помещается имя столбца. В противном случае переменная `$order` принимает значение по умолчанию, равное `"name"`. Такая громоздкая проверка необходима для того, чтобы избежать SQL-инъекции. При включенном режиме `register_globals` интерпретатор PHP автоматически создает для `GET`-параметров переменные. То есть при обращении к скрипту появляется воз-

можность передать через параметр `order` SQL-инъекцию. Поэтому при работе с методами `GET`, `POST` или `COOKIE` необходимо всегда явно проверять переменные, которые передаются от одной страницы к другой, т. к. в этих методах передачи одним из посредников выступает машина клиента, где передаваемые данные могут подвергнуться подделке.

В листинге 18.29 представлен объектно-ориентированный вариант скрипта из листинга 18.28.

Листинг 18.29. Сортировка столбцов (объектно-ориентированный вариант)

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Проверяем параметры, переданные скрипту
  $order = "name";
  if($_GET['order'] == 'name')  $order = "name";
  if($_GET['order'] == 'id_catalog') $order = "id_catalog";
  if($_GET['add']   == 'desc')
  {
    $desc = "DESC";
    $add  = "";
  }
  else
  {
    $desc = "";
    $add  = "desc";
  }

  // Выводим элементы каталога
  $query = "SELECT * FROM catalogs
            ORDER BY $order $desc";
  $cat = $mysqli->query($query);
  if(!$cat) exit($mysqli->error);
  if($cat->num_rows)
  {
    echo "<table border=1>
            <tr>
  <td><a href=$_SERVER[PHP_SELF]?order=id_catalog&add=$add>Номер</a></td>
  <td><a href=$_SERVER[PHP_SELF]?order=name&add=$add>Название</a></td>
            </tr>";
    while($catalog = $cat->fetch_array())
    {
      echo "<tr>
              <td>$catalog[id_catalog]</td>
```

```
            <td>$catalog[name]</td>
          </tr>";
    }
    echo "</table>";
  }
?>
```

18.13. Размер базы данных

Для того чтобы определить размер базы, удобно воспользоваться оператором `SHOW TABLE STATUS`, который в числе прочей информации в столбец `Data_length` возвращает размер, занимаемый таблицами базы данных. Остается только просуммировать значения столбца `Data_length`, чтобы получить общий объем базы данных. В листинге 18.30 приводится пример скрипта, который определяет размер базы данных `test`.

Листинг 18.30. Определение размера базы данных `test` (процедурный вариант)

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  $query = "SHOW TABLE STATUS FROM test";
  $sts = mysql_query($query);
  if(!$sts) exit("Ошибка определения занимаемого объема");
  $mysql_size = 0;
  if(mysql_num_rows($sts))
  {
    while($status = mysql_fetch_array($sts))
    {
      $mysql_size += $status['Data_length'] + $status['Index_length'];
    }
  }
  echo $mysql_size;
?>
```

В листинге 18.31 представлен объектно-ориентированный вариант скрипта из листинга 18.30.

Листинг 18.31. Определение размера базы данных `test` (объектно-ориентированный вариант)

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
```

```
  $query = "SHOW TABLE STATUS FROM test";
  $sts = $mysqli->query($query);
  if(!$sts) exit("Ошибка определения занимаемого объема");
  $mysql_size = 0;
  if($sts->num_rows)
  {
    while($status = $sts->fetch_array())
    {
      $mysql_size += $status['Data_length'] + $status['Index_length'];
    }
  }
  echo $mysql_size;
?>
```

Заключение

Чтобы не говорилось, чтение книги — это всегда в том или ином виде слушание авторского монолога, что, в общем-то, не очень хорошо, поскольку быстрая обратная связь в равной мере необходима как читателям, так и авторам. С этой целью, чтобы авторский монолог превратить в полноценный диалог с читателем, мы создали на своем сайте форум **http://www.softtime.ru/forum/index.php?id_forum=3**, на котором можно задать вопросы. Поэтому авторы искренне надеются, что после того как вы перелистнете эту страницу, мы с вами не расстанемся, а встретимся на нашем форуме.

Успехов вам и всего доброго!

ПРИЛОЖЕНИЕ 1

Системные переменные MySQL

Системные переменные позволяют осуществлять тонкую настройку сервера MySQL. Получить полный список системных переменных можно при помощи оператора `SHOW VARIABLES`, более подробно порядок работы с ними рассматривается в *главе 9*. В данном приложении более подробно рассматривается назначение каждой из системных переменных.

- `auto_increment_increment`. Системная переменная предназначена для установки приращения в механизме `AUTO_INCREMENT` *(см. разд. 4.1.7)*. Переменная может принимать значения от 1 до 65 535 и по умолчанию получает значение 1. Установка переменной в 0 приводит к тому, что она получает значение 1. Попытка присвоить переменной значение меньше 0 или больше 65 535 приводит к установке значения 65 535. Установка значения переменной, например, в 10 приводит к тому, что значение автоинкрементного счетчика увеличивается не на единицу, а сразу на 10 (листинг П1.1).

Замечание

Системная переменная `auto_increment_increment` введена в MySQL, начиная с версии 5.0.2.

Листинг П1.1. Влияние системной переменной `auto_increment_increment` на механизм `AUTO_INCREMENT`

```
SHOW VARIABLES LIKE 'auto_increment_increment';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| auto_increment_increment | 1     |
+-------------------------+-------+
```

```
CREATE TABLE autoinc1
(col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
SET @@auto_increment_increment=10;
SHOW VARIABLES LIKE 'auto_increment_increment';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
+--------------------------+-------+
INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
SELECT col FROM autoinc1;
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
+-----+
```

- `auto_increment_offset`. Системная переменная предназначена для установки начальной позиции в механизме `AUTO_INCREMENT` *(см. разд. 4.1.7)*. Переменная может принимать значения от 1 до 65 535 и по умолчанию получает значение 1. Установка переменной в 0 приводит к тому, что она получает значение 1. Попытка присвоить переменной значение меньше 0 или больше 65 535 приводит к установке значения 65 535. Установка значения переменной, например, в 5 приводит к тому, что значение автоинкрементного счетчика в новой таблице начинается не с 1, а с 5 (листинг П1.2).

Замечание

Системная переменная `auto_increment_offset` введена в MySQL, начиная с версии 5.0.2.

Листинг П1.2. Влияние системной переменной `auto_increment_offset` на механизм `AUTO_INCREMENT`

```
SET @@auto_increment_offset=5;
SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
```

```
CREATE TABLE autoinc2
(col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
INSERT INTO autoinc2 VALUES (NULL), (NULL), (NULL), (NULL);
SELECT col FROM autoinc2;
+-----+
| col |
+-----+
|   5 |
|  15 |
|  25 |
|  35 |
+-----+
```

- `back_log`. Данная переменная устанавливает количество ожидающих выполнения запросов на подключение к MySQL-серверу. Когда главный поток MySQL-сервера получает слишком много запросов на соединение за очень короткое время, ему требуется некоторое время (хотя и небольшое) на то, чтобы проверить соединение и запустить новый поток. Значение `back_log` указывает, сколько запросов может помещаться в стек на время этого короткого промежутка. При этом лишние соединения будут отбрасываться, а клиентам будет возвращаться сообщение о невозможности подключиться к серверу. Значение системной переменной `back_log` следует увеличивать только в том случае, если предвидится большое количество подключений за короткое время. Другими словами, данная переменная устанавливает длину очереди прослушивания входящих TCP/IP-соединений. Каждая операционная система устанавливает свой размер такой очереди, поэтому попытка установить значение переменной больше, чем разрешает операционная система, не приведет к результату.
- `basedir`. Переменная содержит путь к базовому каталогу установки MySQL. Данную переменную можно установить при помощи параметра `--basedir`.
- `bdb_cache_parts`. Переменная задает количество разделов в BDB-кэше. Переменная добавлена, начиная с MySQL версии 5.1.2.
- `bdb_cache_size`. Переменная задает размер буфера, выделяемого для индексов кэширования и строк BDB-таблиц. Если таблицы типа BDB использоваться не будут, значение переменной можно установить в 0, или запустить MySQL-сервер с параметром `--skip-bdb`, чтобы не тратить память на этот кэш.
- `bdb_home`. Переменная устанавливает путь к базовому каталогу таблиц типа BDB. По умолчанию переменной присваивается то же значение, что и переменной `datadir`, определяющей путь к каталогу данных.

- bdb_log_buffer_size. Переменная задает размер буфера, выделяемого для индексов кэширования и строк таблиц BDB. Если таблицы типа BDB использоваться не будут, значение переменной можно установить в 0 или запустить MySQL-сервер с параметром --skip-bdb, чтобы не тратить память на этот кэш.
- bdb_logdir. Переменная устанавливает путь к каталогу, где хранятся журнальные файлы BDB-таблиц. Данную переменную можно установить также при помощи параметра --bdb-logdir.
- bdb_max_lock. Переменная задает максимально допустимое число активных блокировок в таблице BDB (по умолчанию 10 000). Значение переменной следует увеличить, если возникают ошибки при длительных транзакциях или серверу MySQL приходится проверять большое количество строк в таблице. Обычно такие ошибки сопровождаются сообщениями вида:

```
bdb: Lock table is out of available locks
Got error 12 from ...
(bdb: Блокировка таблицы вышла за пределы доступных блокировок
Ошибка 12 из ...)
```

- bdb_region_size. Переменная задает размер области памяти BDB-окружения, определяющий размер пула памяти, используемого для сохранения имен файлов при транзакциях. Переменная введена, начиная с MySQL версии 5.1.2.
- bdb_shared_data. Переменная принимает значение ON, если MySQL-сервер запущен с параметром --bdb-shared-data для обработки таблиц типа BDB в многопроцессорном режиме.
- bdb_tmpdir. Переменная устанавливает путь к каталогу временных файлов механизма BDB. Установить значение переменной можно также при помощи параметра сервера MySQL --bdb-tmpdir.
- binlog_cache_size. Переменная определяет размер кэша, который будет хранить SQL-операторы для бинарного журнала регистраций во время транзакций. Кэш бинарного журнала выделяется для каждого клиента, если сервер поддерживает любую из транзакционных таблиц (BDB, InnoBD) и на сервере активизирован бинарный журнал регистрации (параметр --log-bin). Если часто используются объемные транзакции с множеством операторов в каждой из них, значение этой переменной стоит увеличить, чтобы повысить производительность MySQL-сервера.
- binlog_format. Системная переменная binlog_format определяет тип репликации и может принимать три значения: STATEMENT, ROW или MIXED. Значение STATEMENT определяет репликацию, основанную на анализе выполняемых запросов, ROW — на изменении записей таблиц, а MIXED — смешанный тип,

при котором по умолчанию используется STATEMENT-репликация с переключением в ROW-режим в ряде случаев. Установить значение переменной можно также при помощи параметра сервера MySQL --binlog-format. ROW-репликация введена, начиная с версии MySQL 5.1.5, MIXED, начиная с версии MySQL 5.1.8.

- bulk_insert_buffer_size. Таблицы типа MyISAM используют специальный древовидный кэш для ускорения групповых вставок данных для многострочного оператора INSERT, операторов INSERT...SELECT и LOAD DATA INFILE. Переменная bulk_insert_buffer_size ограничивает размеры дерева кэша на поток. При установке в 0 такая оптимизация будет отключена. По умолчанию переменная имеет значение, равное 8 Мбайт.
- character_sets. Переменная задает поддерживаемые кодировки.

Замечание

Начиная с версии 4.1.1, переменная character_sets удалена из MySQL и заменена различными переменными character_set_*xxx*.

- character_set_client. Переменная задает кодировку для операторов, поступающих со стороны клиента. Введена в MySQL, начиная с версии 4.1.1.
- character_set_connection. Переменная задает кодировку для соединения клиента с MySQL-сервером. Именно в эту кодировку перекодируются полученные от клиента операторы. Переменная введена в MySQL, начиная с версии 4.1.1.
- character_set_database. Переменная задает кодировку, используемую базой данных по умолчанию. MySQL-сервер устанавливает эту переменную всякий раз, когда изменяется текущая база данных. Если ни одна база данных не выбрана, то переменная принимает то значение, которое получает база данных, если при ее создании кодировка не указывается явно. Переменная введена в MySQL, начиная с версии 4.1.1.
- character_set_filesystem. Переменная задает кодировку, используемую для именования файлов в файловой системе. Данная переменная введена, начиная с версии 5.1.6.
- character_set_results. Переменная задает кодировку, используемую для представления возвращаемых клиенту результатов запроса. Введена в MySQL, начиная с версии 4.1.1.
- character_set_server. Переменная определяет кодировку MySQL-сервера по умолчанию. Введена в MySQL, начиная с версии 4.1.1.
- character_set_system. Кодировка, используемая сервером для сохранения идентификаторов. Всегда имеет значение UTF8. Переменная введена в MySQL, начиная с версии 4.1.1.

- character_sets_dir. Переменная устанавливает путь к каталогу с кодировками. Введена в MySQL, начиная с версии 4.1.2.
- collation_connection. Каждая кодировка может применяться для представления нескольких языков, однако для каждого из языков должен использоваться уникальный порядок сортировки. Данная переменная устанавливает сортировку для кодировки, определенной в переменной character_set_connection. Переменная введена в MySQL, начиная с версии 4.1.1.
- collation_database. Переменная устанавливает сортировку для кодировки, определенной в переменной character_set_database. Переменная введена в MySQL, начиная с версии 4.1.1.
- collation_server. Переменная устанавливает сортировку для кодировки, определенной в переменной character_set_server. Переменная введена в MySQL, начиная с версии 4.1.1.
- completion_type. Переменная определяет режим завершения транзакции. Если переменная равна 0, то операторы COMMIT и ROLLBACK не подвергаются изменению. Если переменная равна 1, то операторы COMMIT и ROLLBACK эквиваленты COMMIT AND CHAIN и ROLLBACK AND CHAIN соответственно. Если переменная принимает значение 2, то операторы COMMIT и ROLLBACK эквивалентны COMMIT RELEASE и ROLLBACK RELEASE соответственно. Подробнее транзакции обсуждаются в *главе 10*. Переменная введена в MySQL, начиная с версии 5.0.3.
- concurrent_insert. Если переменная принимает значение 1 (включена), то разрешается одновременное использование операторов INSERT и SELECT для таблиц, не содержащих внутри свободных блоков. Такое поведение можно отключить, запустив MySQL-сервер с параметром --safe или --skip-new, а также установив значение переменной в 0. Если переменная принимает значение 2, то вставка новых записей будет осуществляться в конец таблицы, если таблица одновременно используется вторым потоком, и в свободные блоки, если таблица используется только одним потоком. Последнее значение (2) было добавлено, начиная с версии MySQL 5.0.6.
- connect_timeout. Количество секунд, в течение которых MySQL-сервер ожидает пакет подключений, прежде чем ответить сообщением "Bad handshake" ("Сбойное квитирование").
- datadir. Переменная определяет каталог данных MySQL. Эту переменную можно установить при помощи параметра MySQL-сервера --datadir.
- date_format. Переменная не используется.
- datetime_format. Переменная не используется.
- default_week_format. Значение режима по умолчанию, используемое для функции WEEK() *(см. разд. 6.2)*. Переменная позволяет задать номер режи-

ма, который будет использоваться в том случае, если второй параметр функции не будет указан.

- delay_key_write. Эта переменная влияет только на таблицы типа MyISAM и на обработку ключевого слова DELAY_KEY_WRITE, используемого в операторе CREATE TABALE. Если переменная delay_key_write принимает значение OFF, ключевое слово DELAY_KEY_WRITE игнорируется. Если переменная принимает значение ON (это значение используется по умолчанию), ключевое слово DELAY_KEY_WRITE не игнорируется в операторе CREATE TABLE, если переменная принимает значение ALL, то все новые таблицы рассматриваются как таблицы, при создании которых использовалось ключевое слово DELAY_KEY_WRITE. Если при создании таблицы указывается ключевое слово DELAY_KEY_WRITE, то очистка буфера ключей для таблицы производится не при каждом обновлении индексов, а только когда таблица закрывается (это позволяет увеличить скорость выполнения запросов).
- delayed_insert_limit. После вставки delayed_insert_limit отложенных строк, обработчик INSERT DELAYED проверяет, имеются ли незавершенные операторы SELECT. Если таковые имеются, MySQL-сервер сначала выполняет их, а затем продолжает вставку отложенных строк. По умолчанию параметр принимает значение 100.
- delayed_insert_timeout. Переменная определяет, как долго поток обработчика INSERT DELAYED должен ожидать операторы INSERT перед завершением работы.
- delayed_queue_size. Переменная задает длину очереди обработчика INSERT DELAYED. Если очередь переполняется, все клиенты, которые выполнили запрос INSERT DELAYED, будут ожидать, пока в очереди появится свободное место.
- div_precision_increment. Переменная задает количество знаков после запятой, которые добавляются числу после применения целочисленной операции деления /. По умолчанию переменная имеет значение 4. Данное значение можно изменять в пределах от 0 до 30 (листинг П1.3).

Замечание

Переменная div_precision_increment была введена, начиная с версии 5.0.6.

Листинг П1.3. Использование переменной div_precision_increment

```
SELECT 1/7;
+--------+
| 1/7    |
+--------+
| 0.1429 |
+--------+
```

```
SET div_precision_increment = 30;
SELECT 1/7;
+---------------------------------+
| 1/7                             |
+---------------------------------+
| 0.142857142857142857142857142857 |
+---------------------------------+
```

- `event_scheduler`. Переменная `event_scheduler` включает или отключает планировщик заданий. По умолчанию планировщик заданий отключен. Данная переменная была добавлена, начиная с версии MySQL 5.1.6.
- `expire_logs_days`. Переменная задает число дней, по прошествии которых бинарные журнальные файлы автоматически уничтожаются. По умолчанию переменная принимает значение 0, которое служит сигналом для MySQL-сервера, что бинарные журнальные файлы не следует удалять. Данная переменная была добавлена, начиная с версии MySQL 4.1.0. Более подробно журнальные файлы обсуждаются в *разд. 1.12*.
- `flush`. Переменная принимает значение `ON`, если MySQL-сервер запущен с параметром `--flush`. Данный параметр требует от MySQL-сервера сбрасывать все изменения на жесткий диск после каждого SQL-оператора. Это уменьшает риск разрушения таблиц при сбое, но серьезно снижает производительность сервера. Обычно данный параметр используется при нестабильной работе сервера.
- `flush_time`. При ненулевом значении переменной все таблицы закрываются каждые `flush_time` секунд, чтобы освободить ресурсы и синхронизировать измененные данные на диск. Эта опция рекомендуется только для операционных систем Windows 9*x*, Windows ME, а также для операционных систем с минимальным количеством доступных ресурсов.
- `ft_boolean_syntax`. Список операций, поддерживаемых при полнотекстовом поиске (`IN BOOKEAN MODE`) в логическом режиме *(см. главу 7)*. Значение переменной по умолчанию соответствует строке `'+ -><()~*:""&|'`. Для изменения значения применяются следующие правила:
  - функция операции определяется позицией в строке;
  - значение замены должны состоять из 14 символов;
  - каждый символ должен представлять собой ASCII-символ, отличный от алфавитно-цифрового;
  - первый или второй символ — обязательно пробел;
  - дубликаты использовать нельзя, кроме операций кавычек в позициях 11 и 12. Эти два символа не обязательно должны быть одинаковыми, но только они могут быть таковыми;

- позиции 10, 13 и 14 (со значениями по умолчанию `:`, `&` и `|`) резервируются для будущих расширений.

- `ft_max_word_len`. Переменная определяет максимальную длину слова для включения в индекс `FULLTEXT`. По умолчанию принимает значение 84. После изменения значения данной переменной все существующие индексы `FULLTEXT` должны быть перекомпонованы.
- `ft_min_word_len`. Переменная определяет минимальную длину слова для включения в индекс `FULLTEXT`. По умолчанию принимает значение 4. После изменения значения данной переменной все существующие индексы `FULLTEXT` должны быть перекомпонованы.
- `ft_query_expansion_limit`. Переменная определяет максимальное количество точных совпадений, которое будет использоваться для полнотекстового поиска, выполняемого с помощью `WITH QUERY EXPANSION`. Переменная введена, начиная с версии MySQL 4.1.1.
- `ft_stopword_file`. Переменная определяет путь к файлу, из которого будет считываться список стоп-слов (общераспространенных слов) для полнотекстового поиска. Если установить эту переменную со значением пустой строки, фильтрация стоп-слов будет отключена. После изменения значения данной переменной все существующие индексы `FULLTEXT` должны быть перекомпонованы.
- `general_log`. Переменная `general_log` отображает, включен (`ON`) или выключен (`OFF`) журнал запросов. Состояние переменной зависит от того, используется параметр MySQL сервера `--log` или нет. Переменная введена, начиная с версии MySQL 5.1.12.
- `general_log_file`. Переменная определяет имя файла журнала запросов. Переменная может быть изменена при помощи параметра MySQL сервера `--log`. Переменная введена, начиная с версии MySQL 5.1.12.
- `group_concat_max_len`. Максимально допустимый размер для результата функции `GROUP_CONTACT()`. Переменная введена, начиная с версии MySQL 4.1.0.
- `have_archive`. Переменная `have_archive` принимает значение `YES`, если текущая версия MySQL поддерживает таблицы типа ARCHIVE, в противном случае переменная принимает значение `NO`. Переменная введена, начиная с версии MySQL 4.1.3.
- `have_bdb`. Переменная `have_bdb` принимает значение `YES`, если текущая версия MySQL поддерживает таблицы типа BDB, в противном случае принимает значение `NO`. Если поддержка BDB-таблиц отключается при помощи параметра `--skip-bdb`, переменная `have_bdb` принимает значение `DISABLED`.

- have_blackhole_engine. Переменная have_blackhole_engine принимает значение YES, если текущая версия MySQL поддерживает таблицы типа BLACKHOLE, в противном случае переменная принимает значение NO. Переменная введена, начиная с версии MySQL 4.1.11.
- have_compress. Переменная принимает значение YES, если MySQL-сервер скомпилирован с поддержкой библиотеки сжатия zlib. В противном случае переменная принимает значение NO. В последнем случае функции COMPRESS() и UNCOMPRESS() будут недоступны. Переменная введена, начиная с версии MySQL 4.1.1.
- have_crypt. Переменная have_crypt принимает значение YES, если операционная система поддерживает системный вызов crypt(), иначе переменная принимает значение NO. В последнем случае функция ENCRYPT() не будет доступна.
- have_csv. Переменная have_csv принимает значение YES, если текущая версия MySQL поддерживает таблицы типа CSV, в противном случае переменная принимает значение NO. Переменная введена, начиная с версии MySQL 4.1.4.
- have_dynamic_loading. Переменная принимает значение YES, если текущая версия MySQL поддерживает динамическую загрузку плагинов, в противном случае переменная принимает значение NO. Переменная введена, начиная с версии MySQL 5.1.10.
- have_example_engine. Переменная have_example_engine принимает значение YES, если MySQL-сервер поддерживает таблицы типа EXAMPLE, в противном случае переменная принимает значение NO. Переменная введена, начиная с версии MySQL 4.1.4.
- have_federated_engine. Переменная have_federated_engine принимает значение YES, если MySQL-сервер поддерживает таблицы типа FEDERATED, в противном случае переменная принимает значение NO. Переменная введена, начиная с версии MySQL 5.0.3.
- have_geometry. Переменная have_geometry принимает значение YES, если MySQL-сервер поддерживает пространственные расширения MySQL, в противном случае переменная принимает значение NO. Переменная введена, начиная с версии MySQL 4.1.3.
- have_innodb. Переменная have_innodb принимает значение YES, если текущая версия MySQL поддерживает таблицы типа InnoDB, в противном случае принимает значение NO. Если поддержка InnoDB-таблиц отключается при помощи параметра --skip-innodb, переменная have_innodb принимает значение DISABLED.

- `have_ndbcluster`. Переменная `have_ndbcluster` принимает значение `YES`, если текущая версия MySQL поддерживает таблицы типа NDB Cluster, в противном случае принимает значение `NO`. Если поддержка таблиц NDB Cluster отключается при помощи параметра `--skip-ndbcluster`, переменная `have_ndbcluster` принимает значение `DISABLED`.
- `have_partitioning`. Переменная принимает значение `YES`, если MySQL-сервер поддерживает сегментирование, в противном случае переменная принимает значение `NO`. Введена, начиная с версии MySQL 5.1.6.
- `have_openssl`. Переменная принимает значение `YES`, если MySQL-сервер поддерживает протокол SSL (шифрование данных), в противном случае она принимает значение `NO`.
- `have_query_cache`. Переменная принимает значение `YES`, если MySQL-сервер поддерживает кэш запросов, в противном случае она принимает значение `NO`.
- `have_row_based_replication`. Переменная принимает значение `YES`, если MySQL-сервер может выполнять репликацию с использованием построчного бинарного журнального файла. В противном случае переменная принимает значение `NO`. Переменная введена, начиная с версии MySQL 5.1.5.
- `have_rtree_keys`. Переменная принимает значение `YES`, если MySQL-сервер поддерживает `RTREE`-индексы, в противном случае она принимает значение `NO`. Переменная введена, начиная с версии MySQL 4.1.3.
- `have_symlink`. Переменная принимает значение `YES`, если MySQL-сервер поддерживает символические ссылки. В противном случае она принимает значение `NO`.
- `init_connect`. Переменная задает один или несколько SQL-запросов, которые выполняются сервером для каждого подключающегося клиента. При указании сразу нескольких операторов их разделяют точкой с запятой `;`. Например, при помощи данной переменной можно выставить кодировку по умолчанию, установив значение: `SET GLOBAL init_connect='SET NAMES cp1251'`. Переменная `init_connect` также может быть установлена в командной строке или в конфигурационных файлах my.ini и my.cnf (листинг П1.4).

Замечание

Переменная `init_connect` введена, начиная с версии MySQL 4.1.2.

Листинг П1.4. Использование переменной `init_connect`

```
[mysqld]
init_connect='SET NAMES cp1251'
```

- `init_file`. Переменная содержит путь к файлу, который задается при помощи параметра `--init-file` во время запуска сервера. Этот файл состоит из SQL-операторов, которые сервер должен выполнить при запуске. Каждый оператор должен занимать одну строку и не содержать комментарии.
- `init_slave`. Эта переменная подобна `init_file`, но представляет собой строку, выполняемую подчиненным сервером репликации при каждом запуске SQL-потока. Переменная введена, начиная с версии MySQL 4.1.2.
- `interactive_timeout`. Количество секунд, в течение которых сервер ожидает активности на интерактивном соединении, прежде чем закрыть его.
- `join_buffer_size`. Переменная определяет размер буфера, используемого при операциях объединения таблиц (если при объединении не используются индексы). Буфер устанавливается один раз во время каждой операции объединения двух таблиц.
- `key_buffer_size`. Индексные блоки таблиц типа MyISAM буферизованы. Все потоки используют общий буфер. Переменная `key_buffer_size` определяет размер буфера, используемого для индексных блоков. Буфер ключей также называют *кэшем ключей*. Данное значение необходимо увеличить для повышения эффективности обработки индексов (как для операций чтения, так и многочисленных операций записи). Обычно в качестве значения указывают 25% от общей памяти на машине, которая в основном работает с MySQL. Однако если значение окажется слишком большим (больше 50% от общей памяти), система может активировать страничный обмен и существенно замедлить работу.
- `key_cache_age_threshold`. Переменная управляет понижением статуса буферов с горячей подцепочки (hot sub-chain) кэша ключей до теплой подцепочки (warn sub-chain). Чем меньше значение, тем быстрее происходит это понижение. Минимальное значение — 100. Значение по умолчанию составляет 300. Переменная введена, начиная с версии MySQL 4.1.1.
- `key_cache_block_size`. Переменная определяет размер блока в байтах в кэше ключей. Значение по умолчанию — 1024. Переменная введена, начиная с версии MySQL 4.1.1.
- `key_cache_division_limit`. Переменная определяет разделительную точку между горячей и теплой подцепочками буферной цепочки кэша ключей. Переменная принимает в качестве значения процент буферной цепочки, который будет использоваться для теплой подцепочки. Допустимые значения лежат в диапазоне от 0 до 100. Значение по умолчанию — 100. Переменная введена, начиная с версии MySQL 4.1.1.
- `language`. Переменная определяет путь к каталогу языковой поддержки. Выбранный язык используется для вывода сообщений об ошибках.

- `large_file_support`. Переменная определяет, скомпилирован ли сервер MySQL с параметрами для поддержки больших файлов.
- `large_pages`. Переменная определяет, включена ли поддержка больших страниц памяти (больше 4 Кбайт). Переменная введена, начиная с версии MySQL 5.0.3.
- `lc_time_names`. Переменная позволяет установить локаль MySQL-сервера. Это дает возможность при использовании функций `DATE_FORMAT()`, `DAYNAME()` и `MONTHNAME()` выводить дни недели и месяцы на национальных языках. В листинге П1.5 устанавливается русская локаль.

Листинг П1.5. Установка русской локали

```
SELECT DATE_FORMAT(NOW(), '%e %M %Y (%W)');
+-------------------------------------+
| DATE_FORMAT(NOW(), '%e %M %Y (%W)') |
+-------------------------------------+
| 3 March 2007 (Saturday)             |
+-------------------------------------+
SET lc_time_names = 'ru_RU';
SELECT DATE_FORMAT(NOW(), '%e %M %Y (%W)');
+-------------------------------------+
| DATE_FORMAT(NOW(), '%e %M %Y (%W)') |
+-------------------------------------+
| 3 Марта 2007 (Суббота)              |
+-------------------------------------+
```

- `license`. Переменная содержит тип лицензии, по которой распространяется текущая версия сервера.
- `local_infile`. Переменная сообщает, поддерживает ли (`ON`) текущая версия MySQL ключевое слово `LOCAL` для оператора `LOAD DATA INFILE` или нет (`OFF`).
- `locked_in_memory`. Переменная сообщает, заблокирован ли сервер MySQL при помощи параметра `--memlock` или нет.
- `log`. Переменная сообщает, активизирована ли (`ON`) регистрация всех запросов в общем журнале запросов или нет (`OFF`).
- `log_bin`. Переменная сообщает, активизирована ли (`ON`) регистрация запросов в бинарный журнал или нет (`OFF`).
- `log_bin_trust_function_creators`. Переменная применяется в том случае, когда активизирована регистрация запросов в журнальные файлы. Эта переменная позволяет предотвращать попадание небезопасных хранимых процедур в журнальный файл. Если переменная установлена в 0, пользо-

вателям не разрешается создавать и изменять хранимые процедуры, если они не имеют привилегию SUPER дополнительно к привилегиям CREATE ROUTINE или ALTER ROUTINE. Установка переменной в 0 запрещает объявлять процедуры с характеристиками DETERMINISTIC, READS SQL DATA или NO SQL. Переменная введена, начиная с версии MySQL 5.0.16. Подробнее хранимые процедуры обсуждаются в *главе 12*.

- log_error. Переменная содержит путь к журнальному файлу ошибок.
- log_output. Системная переменная определяет характер хранения записей общего журнала запросов и журнала медленных запросов. Переменная может принимать одно или более следующих значений, разделенных запятой: TABLE (записи журнала хранятся в таблице), FILE (записи журнала хранятся в файле) или NONE (журналирование отключено). Переменная введена, начиная с версии MySQL 5.1.6.
- log_queries_not_using_indexes. Системная переменная определяет, помещаются ли запросы, которые не используют индексы, в журнал медленных запросов. Если такие запросы включаются, переменная принимает значение ON, в противном случае — OFF. Переменная введена, начиная с версии MySQL 5.1.11.
- log_slave_updates. Переменная определяет, должны ли обновления, которые подчиненный сервер получает от главного сервера в процессе репликации, регистрироваться в собственном бинарном журнале подчиненного сервера. Для того чтобы эта функция стала возможна, на подчиненном сервере потребуется активизировать бинарный журнал.
- log_slow_queries. Переменная определяет, должны ли регистрироваться медленные запросы в журнале медленных запросов. Запрос считается *медленным*, если он выполняется дольше, чем это определяет системная переменная long_query_time.
- log_warnings. Переменная определяет, должны ли выводиться дополнительные предупреждения во время старта и работы сервера MySQL. По умолчанию переменная принимает значение 1. До тех пор пока значение переменной больше единицы, ошибки не заносятся в журнальный файл ошибок.
- long_query_time. Если запрос занимает больше времени, чем указано в переменной long_query_time (в секундах), переменная состояния slow_queries увеличивается на единицу. При использовании параметра --log-slow-queries запрос заносится в журнал медленных запросов наряду с основными медленными запросами. При учете данного значения подсчитывается реальное, а не процессорное время, поэтому запрос со временем выполнения ниже пороговой величины в минимально загруженной системе может

обрабатываться дольше установленного порога на системе, которая загружена сильно.

- low_priority_updates. Если переменная установлена в значение 1, все операторы INSERT, UPDATE, DELETE и LOCK TABLE WRITE будут ожидать, пока на задействованной таблице не останется невыполненных операций SELECT или LOCK TABLE READ.
- lower_case_file_system. Переменная сообщает, зависят ли имена файлов и каталогов от регистра (ON) или нет (OFF).
- lower_case_table_names. Если значение переменной установлено в значение 0, сравнение имен таблиц осуществляется с учетом регистра. Если переменная установлена в значение 1, имена таблиц сохраняются на диск с использованием символов нижнего регистра, и сравнения имен таблиц к регистру не чувствительны. Если переменная установлена в значение 2, имена таблиц сохраняются во введенном пользователем регистре. Однако сравнения все равно производятся в нижнем регистре. Не следует присваивать переменной значение 0, если MySQL запускается в операционной системе, не поддерживающей имена файлов, чувствительных к регистру (Windows или Mac OS X).
- max_allowed_packet. Переменная определяет размер одного пакета, т. е. единичный SQL-запрос к MySQL-серверу не может быть больше по объему, чем max_allowed_packet. Первоначально буфер сообщений имеет значение net_buffer_length, однако оно при необходимости автоматически увеличивается до max_allowed_packet.
- max_binlog_cache_size. Если для транзакции, включающей несколько операторов, потребуется больше памяти, чем указано в max_binlog_cache_size, генерируется ошибка "Multi-statement transaction required more than 'max_binlog_chahe_size' bytes of storage" ("При выполнении транзакции, включающей несколько операторов, потребовалось больше, чем указано в 'max_binlog_cache_size' байтов для сохранения").
- max_binlog_size. Если текущий бинарный журнал превышает объем max_binlog_size, открывается новый файл бинарного журнала. Переменная принимает значение от 4096 байтов до 1 Гбайт.
- max_connect_errors. Если количество прерванных соединений с хоста превышает указанное в этой переменной число (по умолчанию 10), этот хост будет заблокирован. Все последующие попытки соединения с этого хоста будут игнорироваться. Разблокировать хост можно при помощи оператора FLUSH HOSTS.
- max_connections. Переменная устанавливает максимальное количество одновременных соединений сервера с клиентами.

- max_delayed_threads. Переменная ограничивает число потоков, запускаемых для обработки операторов INSERT DELAYED.
- max_error_count. Переменная определяет максимальное количество сообщений об ошибках и предупреждениях, сохраняемых для отображения с помощью операторов SHOW ERRORS и SHOW WARNINGS. Переменная введена, начиная с версии MySQL 4.1.0.
- max_heap_table_size. Переменная max_heap_table_size устанавливает максимальный размер для таблиц типа MEMORY (HEAP). Установка данной переменной никак не отражается на любых существующих таблицах MEMORY до тех пор, пока таблица не будет создана заново с помощью оператора CREATE TABLE.
- max_insert_delayed_threads. Данная переменная является синонимом для переменной max_delayed_threads.
- max_join_size. Сервер MySQL запрещает операторы SELECT, которым, возможно, понадобится проверить больше, чем указано в переменной max_join_size, комбинаций строк, или для которых, скорее всего, придется провести больше, чем задано в max_join_size, поисков на диске. Данная переменная позволяет блокировать операторы SELECT, в которых ключи используются некорректно или выполнение которых может занять много времени. Переменную удобно использовать, когда необходимо заблокировать длительные по времени операции объединения таблиц с пропущенными конструкциями WHERE или запросы, возвращающие миллионы строк.
- max_length_for_sort_data. Переменная определяет размер данных в байтах, который используется при оптимизации запросов ORDER BY с помощью неиндексированных столбцов.
- max_prepared_stmt_count. Переменная определяет максимальное количество динамических запросов, которые разрешены на сервере. По умолчанию принимает значение 16 382, значение переменной может быть изменено в интервале от 0 до 1 000 000. Переменная введена, начиная с версии MySQL 5.1.10.
- max_relay_log_size. Если запись, вносимая подчиненным сервером репликации в свой собственный журнал ретрансляции, превышает заданное в max_relay_log_size значение, для журнала ретрансляции будет создан новый файл. Эта переменная позволяет ограничить размер файлов ретрансляции. Переменная max_relay_log_size может принимать значения от 4096 байтов до 1 Гбайт.
- max_seeks_for_key. Переменная задает максимально допустимое число операций при поиске строк по ключам.

- `max_sort_length`. Переменная определяет количество байтов, используемых при сортировке значений `BLOB` или `TEXT`. Используются только первые `max_sort_length` байтов, остальные игнорируются.
- `max_tmp_tables`. Переменная определяет максимальное число одновременно открытых временных таблиц (в расчете на каждого клиента). В настоящий момент переменная не задействована и не учитывается.
- `max_user_connections`. Максимальное число одновременных соединений, разрешаемых любой указанной учетной записью MySQL. Значение 0 означает отсутствие ограничений.
- `max_write_lock_count`. Переменная определяет количество блокировок на запись, после которых разрешается выполнение незаконченных блокировок на чтение.
- `myisam_data_pointer_size`. Размер (в байтах) указателя по умолчанию, который будет использоваться оператором `CREATE TABLE` для таблиц типа MyISAM, когда ни один параметр `MAX_ROWS` не задан. Переменная может принимать значения от 2 до 8 (по умолчанию 4). Данная переменная удалена в версии MySQL 5.0.6.
- `myisam_max_sort_file_size`. Переменная задает размер временного файла, который используется для воссоздания индекса MyISAM (при использовании операторов `REPAIR TABLE`, `ALTER TABLE` или `LOAD DATA INFILE`). Если размер необходимого файла больше значения переменной `myisam_max_sort_file_size`, индекс будет создан с помощью кэша ключей, что намного медленнее.
- `myisam_recover_options`. В данную переменную помещается значение параметра `--myisam-rocover`.
- `myisam_repair_threads`. Если значение переменной больше 1, индексы таблицы MyISAM создаются параллельно (каждый в своем потоке).
- `myisam_sort_buffer_size`. Переменная задает размер буфера, выделяемого при сортировке индексов MyISAM посредством оператора `REPAIR TABLE` или при создании индексов с помощью операторов `CREATE TABLE` или `ALTER TABLE`.
- `myisam_stats_method`. Переменная определяет, как рассматривается значение `NULL` при построении индекса. Переменная может принимать два значения: `nulls_equal` и `nulls_unequal`. При значении `nulls_equal` все `NULL`-значения воспринимаются как эквивалентные записи, при значении `nulls_unequal` все `NULL`-значения воспринимаются как уникальные. Переменная введена, начиная с версии MySQL 5.0.14. Для более старых версий MySQL-сервер ведет себя так, как если бы значение переменной `myisam_stats_method` было установлено в значение `nulls_equal`.

- `myisam_use_mmap`. Если значение переменной установлено в `ON`, для чтения и записи MyISAM-таблиц используется механизм отображения в память. В противном случае переменная принимает значение `OFF`. Переменная введена, начиная с версии MySQL 5.1.4.
- `named_pipe`. Переменная принимает значение `ON`, если сервер поддерживает соединения через именованные каналы, и `OFF` в противном случае. Переменная определена только при работе MySQL-сервера в операционной системе Windows.
- `net_buffer_length`. Переменная `net_buffer_length` устанавливает размер коммуникационного буфера в байтах. Когда операторы, передаваемые MySQL-серверу, превышают по объему это значение, оно автоматически увеличивается до `max_allowed_packet` байтов.
- `net_read_timeout`. Переменная определяет количество секунд, в течение которых сервер ожидает поступление данных от клиента. Этот тайм-аут применяется только для TCP/IP-соединений и не используется для соединений по UNIX-сокету или именованному каналу.
- `net_retry_count`. Если считывание данных прервано, переменная задает число попыток восстановления соединения, прежде чем завершить задачу.
- `net_write_timeout`. Переменная определяет количество секунд, в течение которых сервер станет ожидать, пока блок будет записан в соединение, прежде чем операция записи будет отменена.
- `new`. Переменная принимает значение `ON`, если на MySQL-сервере версии 4.0 необходимо включить нововведения версии 4.1, в противном случае переменная принимает значение `OFF`. В версии выше 4.1 данная переменная всегда принимает значение `OFF`.
- `old_passwords`. Если данная переменная принимает значение `ON`, сервер MySQL использует старый формат пароля пользователя (применявшийся в MySQL версий старше 4.1), в противном случае, если применяется пароль нового формата, переменная принимает значение `OFF`. Переменная введена, начиная с версии MySQL 4.1.1.
- `open_files_limit`. Переменная определяет количество файлов, которые операционная система позволяет открывать серверу MySQL. Это реально разрешаемое системой значение, и оно может отличаться от значения, устанавливаемого администратором MySQL.
- `optimizer_prune_level`. Переменная позволяет контролировать эвристический механизм при оптимизации запросов. Установка переменной `optimizer_prune_level` в 0 отключает эвристический механизм оптимизации, вместо него используется всесторонний анализ. Установка переменной в 1 включает эвристический механизм. Переменная введена, начиная с версии MySQL 5.0.1.

- `optimizer_search_depth`. Переменная определяет максимальную глубину поиска, выполняемую оптимизатором запросов. Значение переменной большее, чем число связей таблиц в многотабличном SQL-запросе, лучше генерирует схему оптимизации, создание которой, однако, может занять значительное время. Значение переменной меньшее, чем число связей в многотабличном SQL-запросе, приводит к быстрому генерированию схемы оптимизации, однако скорость выполнения запроса по такой схеме может быть далека от оптимальной. Если переменная установлена в 0, система автоматически выбирает оптимальное значение. Переменная введена, начиная с версии MySQL 5.0.1.
- `pid_file`. Переменная содержит путь к файлу идентификатора процесса (PID). Эту переменную можно установить с помощью параметра `--pid-file`.
- `plugin_dir`. Переменная содержит путь к каталогу плагинов. Введена, начиная с версии MySQL 5.1.2.
- `port`. Переменная содержит номер порта, по которому MySQL-сервер ожидает соединения с клиентом (по умолчанию 3306, если данный параметр не указан, производится попытка установить соединение именно по данному порту).
- `preload_buffer_size`. Размер буфера, который резервируется для предзагрузки индексов. Переменная введена, начиная с версии MySQL 4.1.1.
- `prepared_stmt_count`. Переменная определяет текущее количество динамических запросов (максимальное количество определяется системной переменной `max_prepared_stmt_count`). Переменная введена, начиная с версии MySQL 5.1.10.
- `protocol_version`. Переменная содержит версию клиент-серверного протокола, используемого сервером MySQL.
- `query_alloc_block_size`. Размер распределения блоков памяти, выделяемых для создаваемых объектов во время анализа и выполнения запросов. При наличии проблем с фрагментацией памяти увеличение данного значения может в определенной степени помочь.
- `query_cache_limit`. MySQL-сервер не будет помещать в кэш запроса результаты, размер которых больше `query_cache_limit` байтов.
- `query_cache_min_res_unit`. Переменная определяет минимальный размер блоков, выделяемых кэшем запросов. Значение по умолчанию — 4 Кбайт. Переменная введена, начиная с версии MySQL 4.1.0.
- `query_cache_size`. Объем памяти, выделяемой для кэширования результатов запросов. Переменная принимает по умолчанию значение 0, при котором кэш запросов отключен.

- query_cache_type. Переменная устанавливает тип кэша запросов. Если переменная принимает значение OFF или 0, MySQL-сервер не будет кэшировать и извлекать результаты из кэша. Однако это не приводит к автоматическому освобождению буфера кэша запросов. Для этого необходимо установить переменную query_cache_size в значение 0. Если переменная query_cache_type принимает значение ON или 1, MySQL-сервер будет помещать в кэш запросов все результаты кроме тех, которые начинаются с SELECT SQL_NO_CACHE. Если переменная query_cache_type принимает значение DEMAND или 2, MySQL-сервер помещает в кэш результаты только тех запросов, которые начинаются с SELECT SQL_CACHE.
- query_cache_wlock_invalidate. Обычно когда один клиент запрашивает блокировку WRITE для таблицы типа MyISAM, остальные клиенты не блокируются на чтение (READ), если результаты запросов представлены в кэше запросов. Однако если переменная query_cache_wlock_invalidate принимает значение 1, блокировка по чтению WRITE приводит к игнорированию любых запросов, в том числе и на чтение. Пока блокировка будет действовать, всем остальным клиентам придется ждать, чтобы получить доступ к блокированной таблице.
- query_prealloc_size. Переменная задает размер постоянного буфера, который используется для синтаксического анализа и выполнения запросов. Этот буфер не освобождается между запросами. При работе со сложными запросами можно установить большее значение для этой переменной, т. к. при этом уменьшается необходимость осуществлять распределение памяти во время выполнения запроса.
- range_alloc_block_size. Переменная определяет размеры блоков, которые выделяются во время оптимизации области.
- read_buffer_size. Каждый поток, выполняющий полное сканирование, выделяет буфер размером read_buffer_size байтов для каждой сканируемой таблицы. При частом последовательном сканировании это значение лучше увеличить.
- read_only. Если на подчиненном сервере репликации эта переменная принимает значение ON, то подчиненный сервер будет обновляться только с главного сервера. Клиенты смогут изменить содержимое его таблиц, только если они обладают привилегией SUPER. Обычно такой режим работы сервера применяется в том, случае, когда необходимы гарантии того, что подчиненный сервер не принимает никаких обновлений от клиентов.
- relay_log_purge. Переменная принимает значение ON, если включен режим автоматического удаления старых журнальных файлов, которые больше не нужны, и OFF в противном случае. Переменная введена, начиная с версии MySQL 4.1.1.

- `read_rnd_buffer_size`. Переменная задает размер буфера, из которого читаются строки после сортировки, для того чтобы избежать повторного поиска на диске. Большое значение этой переменной может значительно увеличить эффективность конструкции `ORDER BY`. Однако этот буфер выделяется для каждого клиента, поэтому не следует задавать слишком большое значение.
- `rpl_recovery_rank`. Переменная в настоящий момент не используется.
- `secure_auth`. Если сервер MySQL запущен с параметром `--secure-auth`, он блокирует соединения со всех учетных записей, пароли которых сохранены в старом (до версии 4.1) формате. В этом случае значение данной переменной будет равно `ON` (включено), в противном случае переменная получит значение `OFF` (выключено).
- `server_id`. Переменная получает значение параметра `--server-id` и используется для присвоения серверам уникальных номеров в схеме репликации.
- `shared_memory`. Переменная принимает значение `ON`, если позволяется использовать общую память соединений, в противном случае переменная получает значение `OFF`. Переменная применяется только в среде Windows. Введена, начиная с версии MySQL 4.1.1.
- `shared_memory_base_name`. Переменная содержит имя совместно используемой памяти соединения. Обычно применяется для запуска сразу нескольких серверов на одном хосте. Значение переменной зависит от регистра и по умолчанию принимает значение `MySQL`. Переменная применяется только в среде Windows. Введена, начиная с версии MySQL 4.1.1.
- `skip_external_locking`. Переменная принимает значение `OFF`, если MySQL-сервер использует внешнюю блокировку, и `ON` в противном случае.
- `skip_networking`. Переменная принимает значение `ON`, если сервер принимает только локальные (не TCP/IP) соединения. Если разрешены сетевые обращения к серверу, переменная получает значение `OFF`.
- `skip_show_database`. Если переменная установлена в значение `ON`, то пользователям, не имеющим привилегии `SHOW DATABASES`, запрещается применять оператор `SHOW DATABASE`. Если переменная принимает значение `OFF`, оператор `SHOW DATABASES` могут применять все пользователи. Оператор будет отображать имена баз данных, только если у пользователя имеется привилегия `SHOW DATABASES` или какая-нибудь привилегия для доступа к базе данных. Если переменная принимает значение `ON` и пользователь имеет привилегию `SHOW DATABASES`, ему будут доступны имена всех баз данных сервера.
- `slave_compressed_protocol`. Если переменная принимает значение `ON`, данные, которыми обмениваются главный и подчиненный серверы, подвер-

гаются сжатию, в противном случае переменная принимает значение OFF. Использование протокола сжатия возможно только в том случае, если его поддерживают и главный, и подчиненный серверы.

- slave_load_tmpdir. Переменная содержит путь к каталогу, где подчиненный сервер создает временные файлы для репликации оператора LOAD DATA INFILE.
- slave_net_timeout. Переменная содержит количество секунд, в течение которых подчиненный сервер должен ожидать данные через соединение с главным сервером, прежде чем прекратить считывание.
- slave_skip_errors. Если переменная принимает значение ON, ошибки репликации должны игнорироваться подчиненным сервером.
- slow_launch_time. Если создание потока занимает больше slow_launch_time секунд, сервер увеличивает значение переменной состояния slow_lanch.
- slow_query_log. Переменная определяет, включен (ON) или выключен (OFF) журнал медленных запросов. Значение переменной зависит от параметра сервера MySQL --log-slow-queries. Переменная введена, начиная с версии MySQL 5.1.12.
- slow_query_log_file. Переменная определяет имя журнала медленных запросов. Установить имя можно при помощи параметра MySQL сервера --log-slow-queries. Переменная введена, начиная с версии MySQL 5.1.12.
- socket. В среде UNIX переменная содержит путь к файлу сокета, используемому для соединения с клиентами. В среде Windows это имя именованного канала, используемого для локальных соединений с клиентами.
- sort_buffer_size. Для каждого потока, которому необходимо выполнить сортировку, выделяется буфер в sort_buffer_size байтов. Следует увеличить данное значение, чтобы ускорить выполнение операций ORDER BY или GROUP BY.
- sql_mode. Переменная определяет режим SQL-сервера.
- sql_slave_skip_counter. Переменная определяет количество событий главного сервера, которые подчиненный сервер репликации должен проигнорировать.
- ssl_ca. Переменная определяет путь к файлу листа доверия SSL CA. Переменная введена, начиная с версии MySQL 5.1.11.
- ssl_capath. Переменная определяет путь к каталогу, в котором содержится сертификат SSL CA в PEM-формате. Введена, начиная с версии MySQL 5.1.11.
- ssl_cert. Переменная определяет имя SSL-сертификата, используемого для установки SSL-соединения. Введена, начиная с версии MySQL 5.1.11.

- `ssl_cipher`. Переменная определяет список разрешенных методов шифрования для SSL. Введена, начиная с версии MySQL 5.1.11.
- `ssl_key`. Переменная определяет имя файла ключей, используемого для установки SSL-соединения. Введена, начиная с версии MySQL 5.1.11.
- `storage_engine`. Переменная является синонимом `table_type`. Введена, начиная с версии MySQL 4.1.2.
- `sync_binlog`. Если значение этой переменной принимает положительное значение, MySQL-сервер синхронизирует информацию в оперативной памяти с жестким диском, после записи в бинарный журнал команды `sync_binlog`. Если переменная принимает значение 0 (а по умолчанию так и есть), синхронизация не производится. Если переменная принимает значение 1, вероятность потери данных значительно снижается, однако это несколько замедляет операции. Переменная введена, начиная с версии MySQL 4.1.3.
- `sync_frm`. Если значение данной переменной устанавливается в значение 1, при создании любого не временного файла информация в оперативной памяти синхронизируется с жестким диском. Такое поведение MySQL-сервера более безопасно, но несколько замедляет операции. Переменная введена, начиная с версии MySQL 4.1.3.
- `system_time_zone`. Переменная содержит часовой пояс сервера. Когда сервер стартует, он извлекает значение часового пояса из операционной системы. При необходимости время можно скорректировать при помощи данной переменной. Однако в последнем случае лучше воспользоваться переменной `time_zone`, которая устанавливает часовой пояс лишь для соединения с клиентом, а не для всего сервера в целом.
- `table_cache`. Это старое название для переменной `table_open_cache`. Начиная с версии MySQL 5.1.3, следует использовать переменную `table_open_cache` вместо `table_cache`.
- `table_definition_cache`. Переменная определяет количество идентификаторов, которые могут быть сохранены в кэше определений. Если используется большое количество таблиц, имеет смысл увеличить значение этой переменной для повышения производительности сервера MySQL. Переменная введена, начиная с версии MySQL 5.1.3.
- `table_lock_wait_timeout`. Переменная определяет время ожидания для блокировки таблицы в секундах. По умолчанию переменная принимает значение, равное 50 секундам. Переменная введена, начиная с версии MySQL 5.0.10.
- `table_open_cache`. Переменная определяет максимально допустимое количество открытых таблиц.

- `table_type`. Переменная содержит тип таблицы по умолчанию (MyISAM, InnoDB или BDB).
- `thread_cache_size`. Переменная определяет, сколько потоков сервер должен поместить в кэш для повторного использования. После отключения клиента его потоки помещаются в кэш, если только там уже не находятся другие потоки. Как только у сервера MySQL возникает потребность в новых потоках, он резервирует потоки из кэша. Новые потоки создаются только в том случае, если кэш пуст. При большом количестве новых соединений, чтобы повысить производительность, можно увеличить значение данной переменной. Однако если в операционной системе хорошая реализация потоков (Windows, Linux), заметного улучшения производительности не наблюдается.
- `thread_concurrency`. В некоторых операционных системах, например Solaris, при помощи данной переменной можно порекомендовать желаемое число одновременно запускаемых потоков.
- `thread_stack`. Переменная определяет размер стека для каждого потока.
- `time_format`. Переменная в настоящий момент не применяется.
- `time_zone`. Переменная определяет текущий часовой пояс.
- `tmp_table_size`. Если временная таблица превышает `tmp_table_size` байтов, она сбрасывается на жесткий диск. Если на сервере достаточно памяти, можно увеличить данное значение для повышения эффективности запросов с конструкцией `GROUP BY`.
- `tmpdir`. Переменная содержит путь к каталогу, который нужен для создания временных файлов и таблиц. Начиная с версии 4.1, эту переменную можно установить в качестве списка из нескольких путей, которые используются по циклическому принципу. Пути отделяются двоеточием (`:`) в операционной системе UNIX и точкой с запятой (`;`) в операционных системах Windows, NetWare и OS/2.
- `transaction_alloc_block_size`. Переменная определяет размер блоков памяти, выделяемых для сохранения запросов, являющихся частью транзакции, в бинарный журнал регистрации.
- `transaction_prealloc_size`. Размер постоянного буфера, который выделяется для сохранения запросов, являющихся частью транзакции, в бинарный журнал. Особенностью данного буфера является то, что он не освобождается между запросами.
- `tx_isolation`. Переменная определяет уровень локализации транзакций по умолчанию.
- `updatable_views_with_limit`. Данная переменная контролирует, могут ли применяться обновления при помощи оператора `UPDATE`, у которых отсут-

ствует `WHERE`-условие со ссылкой на первичный ключ или уникальный индекс. Такие обновления часто могут содержать только конструкцию `LIMIT`. Переменная введена, начиная с версии MySQL 5.0.2. Переменная может принимать два значения:

- 1 или `YES`, если такое обновление допускается, но при этом генерируется предупреждение;
- 0 или `NO`, если подобное обновление не допускается.

- `version`. Переменная содержит версию MySQL-сервера.
- `version_bdb`. Переменная содержит версию BDB-движка. Введена, начиная с версии MySQL 4.1.1.
- `version_comment`. Переменная содержит комментарий к текущей версии MySQL.
- `version_compile_machine`. Переменная содержит тип архитектуры, для которой скомпилирована текущая версия MySQL. Введена, начиная с версии MySQL 4.1.1.
- `version_compile_os`. Переменная содержит тип операционной системы, для которой скомпилирована текущая версия MySQL.
- `wait_timeout`. Переменная определяет количество секунд, в течение которых сервер ожидает активности от неинтерактивного соединения, прежде чем закрыть его.

ПРИЛОЖЕНИЕ 2

Описание компакт-диска

Компакт-диск, прикладываемый к данной книге, содержит данные и программы, перечисленные в табл. П2.1.

Таблица П2.1. *Содержимое компакт-диска*

Папка	Описание	Глава
\binaries	Бинарные дистрибутивы MySQL для Windows и Linux	1
\binaries\mysql50	Инсталлятор MySQL 5.0	1
\binaries\mysql51	Инсталлятор MySQL 5.1	1
\binaries\mysql52	Инсталлятор MySQL 5.2	1
\clients	Графические клиенты для СУБД MySQL	1
\code	PHP-примеры	18
\source	Исходные коды MySQL	1
\source\mysql50	Исходные коды MySQL 5.0	1
\source\mysql51	Исходные коды MySQL 5.1	1
\source\mysql52	Исходные коды MySQL 5.2	1

Предметный указатель

A

Advanced Encryption Standard (AES) 268

C

chcp 38

G

grep 5

I

IP-адрес 278, 279, 346

L

libmysql 3

M

MAC-адрес 283
Message-Digest Algorithm (MD5) 269

P

PHP:
◊ mysql_affected_rows() 494
◊ mysql_change_user() 494
◊ MYSQL_CLIENT_COMPRESS 491
◊ mysql_client_encoding() 494
◊ MYSQL_CLIENT_IGNORE_SPACE 491
◊ MYSQL_CLIENT_INTERACTIVE 491
◊ MYSQL_CLIENT_SSL 491
◊ mysql_close() 492, 494
◊ mysql_connect() 490, 494
◊ mysql_data_seek() 494, 518
◊ mysql_db_name() 494
◊ mysql_db_query() 494
◊ mysql_errno() 494
◊ mysql_error() 492, 494
◊ mysql_escape_string() 494
◊ mysql_fetch_array() 492, 495, 516
◊ mysql_fetch_assoc() 492, 495
◊ mysql_fetch_field() 495
◊ mysql_fetch_lengths() 495
◊ mysql_fetch_object() 492, 495
◊ mysql_fetch_row() 492, 495
◊ mysql_field_flags() 495
◊ mysql_field_len() 495
◊ mysql_field_name() 495
◊ mysql_field_seek() 495
◊ mysql_field_table() 495
◊ mysql_field_type() 495
◊ mysql_free_result() 496
◊ mysql_get_client_info() 496
◊ mysql_get_host_info() 496
◊ mysql_get_proto_info() 496
◊ mysql_get_server_info() 496
◊ mysql_info() 496
◊ mysql_insert_id() 496, 512
◊ mysql_list_dbs() 496
◊ mysql_list_fields() 496
◊ mysql_list_processes() 496
◊ mysql_list_tables() 496
◊ mysql_num_fields() 496
◊ mysql_num_rows() 496, 516, 520
◊ mysql_pconnect() 496

◊ mysql_ping() 496
◊ mysql_query() 491, 492, 496, 506, 507
◊ mysql_real_escape_string() 497, 511
◊ mysql_result() 492, 497, 520, 523
◊ mysql_select_db() 491, 497
◊ mysql_stat() 497
◊ mysql_tablename() 497
◊ mysql_thread_id() 497
◊ mysql_unbuffered_query() 497
◊ mysqli() 497
◊ mysqli_result 503
◊ mysqli_result->close() 503
◊ mysqli_result->current_field 504
◊ mysqli_result->data_seek() 503
◊ mysqli_result->fetch_array() 503
◊ mysqli_result->fetch_assoc() 503
◊ mysqli_result->fetch_field() 503
◊ mysqli_result->fetch_field_direct() 503
◊ mysqli_result->fetch_fields() 503
◊ mysqli_result->fetch_object() 504
◊ mysqli_result->fetch_row() 504
◊ mysqli_result->field_count 504
◊ mysqli_result->field_seek() 504
◊ mysqli_result->free_result() 504
◊ mysqli_result->lengths 504
◊ mysqli_result->num_rows 504, 520
◊ mysqli_result->type 504
◊ mysqli_stmt 501
◊ mysqli_stmt->affected_rows 502
◊ mysqli_stmt->bind_param() 502
◊ mysqli_stmt->bind_result() 502
◊ mysqli_stmt->data_seek() 502
◊ mysqli_stmt->errno 503
◊ mysqli_stmt->execute() 502
◊ mysqli_stmt->fetch() 502
◊ mysqli_stmt->free_result() 502
◊ mysqli_stmt->id 503
◊ mysqli_stmt->num_rows 503
◊ mysqli_stmt->param_count 503
◊ mysqli_stmt->prepare() 502
◊ mysqli_stmt->reset() 502
◊ mysqli_stmt->result_metadata() 502
◊ mysqli_stmt->send_long_data() 502
◊ mysqli_stmt->sqlstate 503
◊ mysqli_stmt->store_result() 502
◊ mysqli_stmt->close() 502
◊ mysqli->affected_rows 500
◊ mysqli->autocommit() 498
◊ mysqli->change_user() 498
◊ mysqli->character_set_name() 498
◊ mysqli->client_info 500
◊ mysqli->client_version 500
◊ mysqli->close() 498
◊ mysqli->commit() 498
◊ mysqli->errno 500
◊ mysqli->error 500, 503
◊ mysqli->field_count 500, 503
◊ mysqli->get_client_info() 498
◊ mysqli->host_info 500
◊ mysqli->info 500
◊ mysqli->insert_id 500, 503
◊ mysqli->kill() 498
◊ mysqli->more_results() 499
◊ mysqli->multi_query() 498, 507
◊ mysqli->mysqli_real_escape_string() 499
◊ mysqli->next_result() 499
◊ mysqli->options() 499
◊ mysqli->ping() 499
◊ mysqli->prepare() 499
◊ mysqli->protocol_version 500
◊ mysqli->query() 499, 506
◊ mysqli->real_connect() 499
◊ mysqli->rollback() 499
◊ mysqli->select_db() 499
◊ mysqli->server_info 500
◊ mysqli->server_version 500
◊ mysqli->set_charset() 499
◊ mysqli->sqlstate 500
◊ mysqli->ssl_set() 499
◊ mysqli->stat() 499
◊ mysqli->stmt_init() 499
◊ mysqli->store_result() 499
◊ mysqli->thread_id 500
◊ mysqli->thread_safe() 500
◊ mysqli->use_result() 500
◊ mysqli->warning_count 500
◊ дескриптор результирующей таблицы 490
◊ дескриптор соединения 490

S

Secure Hash Algorithm (SHA1) 268
SSL 357, 549

U

Universal Unique Identifier (UUID) 279, 282

А

Атрибут AUTO_INCREMENT 496, 500, 539, 540

Б

База данных:
◊ information_schema 76
◊ mysql 76, 399
◊ изменение 78
◊ имя 75
◊ привилегии 352
◊ создание 75, 505
◊ удаление 76
◊ целостность 333
Библиотека zlib 548
Блокировка:
◊ получение 278
◊ проверка 278, 281
◊ снятие 279, 280

В

Версия сервера 275, 276
Вертикальный режим вывода данных 46
Встроенный сервер 5

Д

Дамп 56, 102
Директива 29
Доменное имя 347

Ж

Журнал 68
◊ бинарный 542, 549, 551
  ▫ журнал регистрации 69
◊ запросов 547
  ▫ общий 69
◊ медленных запросов 72, 552, 560
◊ общий 551
◊ ошибок 68, 552
◊ ретрансляции 554

З

Запрос:
◊ вложенный 218
◊ выборка данных 171
◊ двухтабличный 209
◊ динамический 330, 554, 557
◊ коррелированный 230
◊ левое объединение 216
◊ многотабличный 208
◊ перекрестное объединение 210
◊ правое объединение 216
◊ самообъединение таблицы 214
◊ строчный 230

И

Именованное условие 409
Индекс:
◊ FOREIGN KEY 441
◊ FULLTEXT 295, 547
◊ FULLTEXT, многостолбцовый 296
◊ PRIMARY KEY 168, 441
◊ UNIQUE 136, 168, 441
◊ внешний ключ 95
◊ вторичный ключ 162
◊ обычный 127, 132, 136
◊ первичный ключ 95, 127—129, 144
◊ полнотекстовый 127
◊ сжатие 105
◊ создание 134, 135
◊ удаление 136
◊ уникальный 95, 127, 132
Информационная схема 439
Информационное сообщение 122

К

Каскадное удаление или обновление связанных записей 164
Каталог данных 75, 104, 111, 544
Квантификатор 292
Кодировка 38, 78, 95, 259, 543
Команда DELIMITER 43
Комментарий 89, 369
◊ в конфигурационном файле 29
Компьютерная сеть 111, 347
Конструкция:
◊ ADD FULLTEXT 135
◊ ADD INDEX 135
◊ ADD PRIMARY KEY 135
◊ ADD UNIQUE 135
◊ AFTER 115, 117
◊ AGAINST 298, 303, 305
◊ ALL 201, 206, 227

Конструкция *(прод.)*:
◊ ANY 226, 227
◊ AS 161, 191, 213, 232, 311
◊ AUTO_INCREMENT 94, 95, 102, 108, 109, 120, 131, 151—153
◊ BETWEEN 179, 186
◊ BINARY 182
◊ CHANGE 117
◊ CHARACTER SET 95, 103
◊ CHECKSUM 103
◊ COLLATE 95, 97
◊ COLUMN 115
◊ COMMENT 95, 98, 104, 112
◊ DATA DIRECTORY 104
◊ DEFAULT 91, 94, 95, 143
◊ DEFAULT CHARACTER SET 78, 103
◊ DELAYED 155
◊ DISTINCT 200—202, 207, 308, 314, 315, 317, 318
◊ DISTINCTROW 200
◊ DROP INDEX 136
◊ DROP PRIMARY KEY 136, 137
◊ ENCLOSED BY 61, 63
◊ ENGINE 101, 108, 110, 112
◊ ESCAPED BY 61, 63
◊ EXISTS 228
◊ FIELDS 62, 63
◊ FIRST 110, 115, 117
◊ FOREIGN KEY 95, 162, 163
  ▫ REFERENCES 162
  ▫ SET NULL 162
  ▫ CASCADE 162
  ▫ NO ACTION 162
  ▫ RESTRICT 163
  ▫ SET DEFAULT 163
◊ FROM 62, 92, 171, 197, 209, 215, 232
◊ GROUP BY 201—204, 213, 214, 307, 314, 560
◊ HAVING 203, 204, 223
◊ IF NOT EXISTS 77, 81
◊ IGNORE 59, 150
◊ IN 180, 224, 225
◊ IN BOOLEAN MODE 303
◊ IN NATURAL LANGUAGE MODE 302, 305
◊ IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION 303
◊ INDEX 95, 132, 133
◊ INDEX DIRECTORY 104
◊ INSERT_METHOD 110
◊ INTERVAL 146, 191, 245
◊ INTO OUTFILE 62
◊ INTO TABLE 59
◊ IS NOT NULL 178
◊ IS NULL 178
◊ ISNULL() 178
◊ KEY 95, 132, 133
◊ LAST 110
◊ LIKE 187
◊ LIMIT 158, 159, 168, 196, 277, 526
◊ LINES 60, 62, 63
◊ LOCAL 58
◊ MATCH 298
◊ MATCH (…) AGAINST (…) 298, 302
◊ MODIFY 116, 117
◊ NO 110
◊ NOT BETWEEN 179, 186
◊ NOT EXISTS 228
◊ NOT IN 181, 225
◊ NOT NULL 90, 94
◊ NULL 90, 94, 132
◊ ON 215, 223
◊ ORDER BY 2, 63, 96, 119, 159, 168, 192, 199, 214, 239, 311, 316, 319, 332, 342, 364, 417, 425, 438, 451, 464, 487, 532, 536, 537, 559, 560, 563, 565
◊ ORDER BY ... ASC 195
◊ ORDER BY ... DESC 193
◊ PACK_KEYS 105
◊ PRIMARY KEY 95, 128, 129
◊ RENAME 118
◊ REPLACE 59
◊ ROW 231
◊ ROW_FORMAT 105
◊ ROW_FORMAT=COMPRESSED 105
◊ ROW_FORMAT=DEFAULT 105
◊ ROW_FORMAT=DYNAMIC 105
◊ ROW_FORMATFIXED 105
◊ SEPARATOR 315
◊ SET 165
◊ SOME 226
◊ SQL_CALC_FOUND_ROWS 277
◊ STARTING BY 60, 63
◊ TERMINATED BY 60, 61, 63
◊ TO 65
◊ TYPE 101, 110
◊ UNION 110
◊ UNIQUE 95, 132
◊ UNIQUE INDEX 132
◊ UNIQUE KEY 132
◊ UNSIGNED 83, 85

◊ USING 217
◊ VALUES 154, 169
◊ WHERE 157, 158, 165, 173, 182, 202—204, 213—215, 223, 298, 313
◊ WHITH QUERY EXPANSION 305
◊ WITH QUERY EXPANSION 303, 306
◊ ZEROFILL 84
Коэффициент релевантности 300
Курсоры 411

Л

Лицензии 551
Логический режим 303, 305
◊ + 303

О

Обработчик ошибок 405
◊ CONTINUE 405
◊ EXIT 405
◊ NOT FOUND 406
◊ SQLEXCEPTION 406
◊ SQLSTATE 405
◊ SQLWARNING 406
◊ UNDO 405
Общеупотребительные слова 298
Оператор:
◊ != 174
◊ && 382
◊ := 322, 324
◊ || 382
◊ + 247
◊ < 174, 382
◊ <= 174, 382
◊ <=> 174, 177
◊ <> 174, 382
◊ = 174, 177, 324, 382
◊ > 174, 382
◊ >= 174, 382
◊ ALTER DATABASE 78, 338
◊ ALTER EVENT 461, 462
◊ ALTER FUNCTION 404
◊ ALTER PROCEDURE 404
◊ ALTER TABLE 73, 113—115, 117, 118, 135, 167, 297, 338, 481, 555
◊ ALTER VIEW 436
◊ ANALYZE TABLE 73
◊ AND 175, 302
◊ BACKUP TABLE 64
◊ BACKUP TABLES 65
◊ BEGIN 337, 338
◊ BEGIN … END 366, 373, 375, 382, 389, 420
◊ BEGIN WORK 337
◊ CALL 367, 380, 431
◊ CASE 385, 420
◊ CHANGE MASTER 350
◊ CHANGE MASTER TO 362
◊ CHECK TABLE:
  ▫ QUICK 138
  ▫ CHANGED 138
  ▫ EXTENDED 138
  ▫ FAST 138
  ▫ MEDIUM 138
◊ CHECKSUM TABLE:
  ▫ EXTENDED 124
  ▫ QUICK 124
◊ CLOSE 411, 412
◊ COMMIT 336, 338, 544
◊ CREATE DATABASE 75, 338, 505
◊ CREATE EVENT 456
  ▫ AT 456
  ▫ DISABLE 460
  ▫ ENABLE 460
  ▫ END 458
  ▫ EVERY 458
  ▫ IF NOT EXISTS 456
  ▫ ON COMPLETION PRESERVE 460
  ▫ ON SCHEDULE 456
  ▫ STARTS 458
◊ CREATE FUNCTION 365, 379, 402
  ▫ RETURNS 367
◊ CREATE INDEX 134, 135, 297, 338, 349
◊ CREATE PROCEDURE 365, 379, 402
  ▫ IN 365, 370
  ▫ INOUT 365, 371
  ▫ OUT 365, 371
◊ CREATE ROUTINE:
  ▫ DETERMINISTIC 552
  ▫ NO SQL 552
  ▫ READS SQL DATA 552
◊ CREATE TABLE 50, 58, 95, 101, 108, 110, 132, 208, 296, 338, 349, 466, 554, 555
  ▫ DELAY_KEY_WRITE 545
◊ CREATE TEMPORARY TABLE 105
◊ CREATE TRIGGER 419, 420
◊ CREATE USER 40, 343, 344, 349
  ▫ IDENTIFIED BY 344
  ▫ PASSWORD 344
◊ CREATE VIEW 349, 427, 436, 438
◊ DEALLOCATE PREPARE 332
◊ DECLARE 372, 420

Оператор *(прод.)*:
◊ DECLARE … CONDITION FOR 406, 409
◊ DECLARE … HANDLER FOR 405
◊ DECLARE CURSOR 411, 412
◊ DELETE 113, 157, 158, 295, 349, 351, 355, 419, 482, 553
◊ DELIMITER 368
◊ DESC 91
◊ DESCRIBE 91, 93, 128, 131, 443, 445, 447
◊ DROP DATABASE 76, 338
◊ DROP EVENT 461
◊ DROP FUNCTION 403
◊ DROP INDEX 134, 338, 349
◊ DROP PREPARE 332
◊ DROP PROCEDURE 402, 403
◊ DROP PROCEDURE, IF EXISTS 403
◊ DROP TABLE 113, 116, 338, 349
◊ DROP TRIGGER 423
◊ DROP USER 345, 349, 355
◊ DROP VIEW 437
  ▫ IF EXISTS 437
◊ ELSEIF 385
◊ EXECUTE 330
◊ FETCH 411, 412
◊ FLUSH 349
◊ FLUSH HOSTS 553
◊ FLUSH LOGS 67, 68
◊ FLUSH TABLES 55, 361
◊ GRANT 348
  ▫ IDENTIFIED BY 353
  ▫ LIKE 353
  ▫ MAX_CONNECTIONS_PER_HOUR 356
  ▫ MAX_QUERIES_PER_HOUR 356
  ▫ MAX_UPDATES_PER_HOUR 356
  ▫ MAX_USER_CONNECTIONS 356
  ▫ ON 352
  ▫ REQUIRE 357
  ▫ REQUIRE CIPHER 358
  ▫ REQUIRE ISSUER 357
  ▫ REQUIRE SSL 357
  ▫ REQUIRE SUBJECT 357
  ▫ REQUIRE X509 357
◊ GROUP BY, WITH ROLLUP 319
◊ IF 382, 384, 386, 420
◊ INSERT 64, 113, 150, 169, 295, 334, 349, 351, 419, 422, 435, 543, 544, 553
◊ INSERT … SELECT 543
◊ INSERT:
  ▫ DELAYED 545, 554
  ▫ многострочный 154
  ▫ однострочный 141
◊ ITERATE 393, 395, 420
◊ JOIN 215, 218
◊ KILL 350
◊ LEAVE 391, 392, 395, 420
◊ LEFT JOIN 216
◊ LEFT OUTER JOIN 217
◊ LOAD DATA 58, 60, 155, 349, 543, 551, 555, 560
  ▫ INFILE 62
  ▫ LOCAL 551
◊ LOAD MASTER DATA 338
◊ LOCK TABLE 340, 349
  ▫ READ 340, 553
  ▫ WRITE 340, 553
  ▫ LOW_PRIORITY 342
◊ LOCK TABLES 65, 338, 339
◊ LOOP 395, 420
◊ NOT 302
◊ NOT LIKE 185
◊ OPEN 411, 412
◊ OPTIMIZE TABLE 73
◊ OR 175, 302
◊ PREPARE 330
◊ PURGE MASTER LOGS 350
◊ REGEXP 285
◊ RENAME 338
◊ RENAME TABLE 118, 338
◊ RENAME USER 345, 349
◊ REPAIR TABLE 121, 555
◊ REPAIR TABLEEXTENDED 123
◊ REPAIR TABLEQUICK 123
◊ REPAIR TABLES 122
◊ REPAIR TABLEUSE_FRM 123
◊ REPEAT 394, 420
◊ REPLACE 168, 169
◊ RESTORE TABLE 66
◊ RESTORE TABLES 66
◊ RETURN 366
◊ REVOKE 348, 355
◊ REVOKE ALL PRIVILEGES 349
◊ RIGHT JOIN 216
◊ RIGHT OUTER JOIN 217
◊ RLIKE 285
◊ ROLLBACK 336, 338, 544
◊ ROLLBACK TO SAVEPOINT 338
◊ SAVEPOINT 338
◊ SELECT 113, 119, 171, 173, 201, 205, 206, 209, 210, 222, 277, 298, 313, 322, 350, 351, 399, 401, 411, 544, 553, 554
◊ SELECT ... INTO 349
◊ SELECT ... INTO OUTFILE 62

◊ SELECT … INSERT 155
◊ SELECT … INTO … FROM 375, 410
◊ SELECT:
  ▫ SQL_NO_CACHE 558
  ▫ WHERE 554
◊ SET 106, 164, 324, 336, 371, 420
  ▫ DEFAULT 328
  ▫ LOCAL 328
  ▫ SESSION 328
◊ SET CHARACTER SET 53
◊ SET GLOBAL 327, 350
◊ SET NAMES 52
◊ SET PASSWORD 346
◊ SET SESSION 327
◊ SET TRANSACTION 337
◊ SHOW 329, 440
◊ SHOW CHARACTER SET 48, 96, 440
◊ SHOW COLLATION 49, 441
◊ SHOW COLUMNS 92, 440, 447
◊ SHOW CREATE EVENT 460, 461
◊ SHOW CREATE FUNCTION 381, 402
◊ SHOW CREATE PROCEDURE 381, 402
◊ SHOW CREATE TABLE 95, 104, 107
◊ SHOW CREATE VIEW 350, 438
◊ SHOW DATABASE 559
◊ SHOW DATABASES 75, 350, 353, 439, 440, 442
◊ SHOW ENGINES 441
◊ SHOW ERROR 554
◊ SHOW EVENTS 441, 463
◊ SHOW FIELDS 93
◊ SHOW FULL COLUMNS 95, 98
◊ SHOW FULL PROCESSLIST 349
◊ SHOW FUNCTION STATUS 398, 399
◊ SHOW FUNCTION STATUS 441
◊ SHOW GLOBAL STATUS 442
◊ SHOW GLOBAL VARIABLES 442
◊ SHOW GRANTS 356
◊ SHOW INDEX 440
◊ SHOW MASTER STATUS 362
◊ SHOW PLUGINS 441
◊ SHOW PRIVILEGES 352
◊ SHOW PROCEDURE STATUS 397—399, 441
◊ SHOW PROCESSLIST 441
◊ SHOW SESSION STATUS 442
◊ SHOW SESSION VARIABLES 442
◊ SHOW SLAVE STATUS 363
◊ SHOW TABLE STATUS 104, 440, 444, 535
◊ SHOW TABLES 353
◊ SHOW TRIGGERS 424, 441
◊ SHOW VARIABLES 326, 539
◊ SHOW WARNINGS 438, 554
◊ START SLAVE 363
◊ START TRANSACTION 337, 338
◊ TRUNCATE TABLE 158, 338
◊ UNION 205, 206
◊ UNION ALL 206
◊ UNION DISTINCT 207
◊ UNLOCK TABLE 340, 349
◊ UNLOCK TABLES 55, 339, 362
◊ UPDATE 113, 158, 165, 166, 168, 351, 419, 435, 553, 562
◊ WHILE 389, 420
◊ XOR 176
Ошибка операции 122

П

Параметр 29
◊ --des-key-file 267
◊ --skip-grant-tables 346
Параметр mysql:
◊ --delimiter 368
◊ --delimiter=name 43
◊ -p 41
◊ -u 36
◊ --vertical 46
Параметр mysqld:
◊ --basedir 541
◊ --bdb-logdir 542
◊ --bdb-shared-data 542
◊ --bdb-tmpdir 542
◊ --binlog-do-db 70
◊ --binlog-format 543
◊ --binlog-ignore-db 71
◊ --console 69
◊ --datadir 544
◊ --default-character-set 51
◊ --default-collation 51
◊ --defaults-file 29
◊ --event-scheduler 456
◊ --flush 546
◊ --ft_min_word_len 301
◊ --init-file 550
◊ --log 68, 547
◊ --log-bin 70, 360, 542
◊ --log-bin-index 70
◊ --log-error 68
◊ --log-output 68
◊ --log-queries-not-using-indexes 73

Параметр mysqld *(прод.)*:
◊ --log-slow-admin-statements 73
◊ --log-slow-queries 68, 72, 552, 560
◊ --memlock 551
◊ --myisam-rocover 555
◊ --pid-file 557
◊ --safe 544
◊ --secure-auth 559
◊ --server-id 360, 559
◊ --skip-bdb 541, 542, 547
◊ --skip-new 544
◊ --skip-slave-start 362
◊ --sql-mode 233
◊ --standalone 24
Параметр mysqldump:
◊ -A 57
◊ --all-databases 57
◊ -B 56
◊ --databases 56
Параметр таблицы, MAX_ROWS 555
Переменная:
◊ SQL 321
◊ глобальная 327
◊ окружения 38
◊ пользовательская 322, 371
◊ сеансовая 327
◊ системная 325
Плагин 557
Планировщик заданий 453, 546
Полнотекстовый поиск 295, 546
Пользователь:
◊ root 40, 346
◊ SSL-доступ 357
  ▫ REQUIRE CIPHER 358
  ▫ REQUIRE ISSUER 357
  ▫ REQUIRE SSL 357
  ▫ REQUIRE SUBJECT 357
  ▫ REQUIRE X509 357
◊ имя 40, 112, 275, 490
◊ ограничение 356
  ▫ MAX_CONNECTIONS_PER_HOUR 356
  ▫ MAX_QUERIES_PER_HOUR 356
  ▫ MAX_UPDATES_PER_HOUR 356
  ▫ MAX_USER_CONNECTIONS 356
◊ пароль 40, 112, 490, 556
◊ привилегии 344, 348
◊ создание 40
◊ удаление 345, 355
Порт 19, 112, 557
Последовательность:
◊ % 183, 353
◊ * 172, 198, 212
◊ .* 161
◊ /* 369
◊ @ 322, 370
◊ @@ 106
◊ _ 184, 353
◊ \n 60, 62
◊ \r\n 60—63
◊ \t 61, 63
◊ бинарная 394
Постраничная навигация 277
Представление 427
◊ ALGORITHM 430
◊ CASCADED 435
◊ CHARACTER_SETS 440
◊ COLLATION_CHARACTER_SET_APPLICABILITY 441
◊ COLLATIONS 441
◊ COLUMN_PRIVILEGES 440
◊ COLUMNS 440, 447
◊ ENGINES 441
◊ EVENTS 441, 464
◊ FILES 441
◊ GLOBAL_STATUS 442
◊ GLOBAL_VARIABLES 442
◊ KEY_COLUMN_USAGE 441
◊ LOCAL 435
◊ MERGE 431
◊ PARTITIONS 441
◊ PLUGINS 441
◊ PROCESSLIST 441
◊ REFERENTIAL_CONSTRAINTS 442
◊ ROUTINES 441
◊ SCHEMA_PRIVILEGES 440
◊ SCHEMATA 442
◊ SCHEMATE 440
◊ SESSION_STATUS 442
◊ SESSION_VARIABLES 442
◊ STATISTICS 440
◊ TABLE_CONSTRAINTS 441
◊ TABLE_PRIVILEGES 440
◊ TABLES 440, 444
◊ TEMPTABLE 431
◊ TRIGGERS 441
◊ UNDEFINED 431
◊ USER_PRIVILEGES 440
◊ VIEWS 441
◊ WITH CHECK OPTION 435
◊ вертикальное 432
◊ горизонтальное 434

◊ редактирование 436
◊ смешанное 434
◊ создание 427
◊ удаление 437
Предупреждение 123
Привилегия 349
◊ ALL 349
◊ ALTER 349
◊ ALTER ROUTINE 349, 552
◊ CREATE 349
◊ CREATE ROUTINE 349, 552
◊ CREATE TEMPORARY TABLES 349
◊ CREATE USER 349
◊ CREATE VIEW 349, 437
◊ DELETE 349, 437
◊ DROP 349
◊ EVENT 349
◊ EXECUTE 349
◊ FILE 349
◊ GRANT OPTION 350
◊ INDEX 349
◊ INSERT 349
◊ LOCK TABLES 349
◊ PROCESS 349
◊ RELOAD 349
◊ REPLICATION CLIENT 349
◊ REPLICATION SLAVE 350
◊ SELECT 350, 437
◊ SHOW DATABASES 350, 559
◊ SHOW VIEW 350
◊ SHUTDOWN 350
◊ SUPER 350, 420, 552, 558
◊ TRIGGER 350
◊ UPDATE 350
◊ USAGE 350
◊ USEAGE 351
◊ WITH GRANT OPTION 354
◊ уровень 352
Протокол соединения 112
◊ версия 557

Р

Регулярное выражение 285
◊ $ 287
◊ * 292
◊ ? 292
◊ [[:<:]] 288
◊ [[:>:]] 288
◊ \\ 291
◊ \\ 289
◊ ^ 286, 290
◊ {} 294
◊ | 288
◊ + 292
◊ \f 291
◊ \n 291
◊ \r 291
◊ \t 291
◊ число формата ###.## 294
Режим SQL-сервера 560
Репликация 359, 549, 552, 554, 560
◊ главный сервер 552
◊ подчиненный сервер 552

С

Сегментирование 465, 549
◊ COALESCE PARTITION 482
◊ DATA DIRECTORY 480
◊ DELETE PARTITION 481
◊ HASH 467, 474
◊ INDEX DIRECTORY 480
◊ KEY 467, 475
◊ LIST 467, 472
◊ MAXVALUE 470
◊ PARTITION 468
◊ PARTITION BY 467
◊ PARTITIONS 474
◊ RANGE 467, 468
◊ REORGANIZE PARTITION 485
◊ SUBPARTITION BY 476
◊ SUBPARTITIONS 476
◊ VALUES IN 472
◊ VALUES LESS THAN 468
◊ вертикальное 465
◊ горизонтальное 465
◊ добавление разделов 483
◊ индексное 475
◊ линейное 467
◊ перепланировка разделов 485
◊ подразделы 476
◊ списочное 471
◊ удаление разделов 481
◊ хэш 474
Секретный ключ 266, 268
Сервис 7, 20, 24
Системная переменная 554
◊ auto_increment_increment 539
◊ auto_increment_offset 540

Системная переменная *(прод.)*:
◊ autocommit 336
◊ back_log 541
◊ basedir 541
◊ bdb_cache_parts 541
◊ bdb_cache_size 541
◊ bdb_home 541
◊ bdb_log_buffer_size 542
◊ bdb_logdir 542
◊ bdb_max_lock 542
◊ bdb_region_size 542
◊ bdb_shared_data 542
◊ bdb_tmpdir 542
◊ BIG_TABLES 106
◊ binlog_cache_size 542
◊ binlog_format 542
◊ bulk_insert_buffer_size 543
◊ character_set_client 543
◊ character_set_connection 543
◊ character_set_database 543
◊ character_set_filesystem 543
◊ character_set_results 543
◊ character_set_server 543
◊ character_set_system 543
◊ character_sets 543
◊ character_sets_dir 544
◊ collation_connection 544
◊ collation_database 544
◊ collation_server 544
◊ completion_type 544
◊ concurrent_insert 544
◊ connect_timeout 544
◊ datadir 544
◊ date_format 544
◊ datetime_format 544
◊ default_week_format 544
◊ delay_key_write 545
◊ delayed_insert_limit 545
◊ delayed_insert_timeout 545
◊ delayed_queue_size 545
◊ div_precision_increment 545
◊ event_scheduler 454, 546
◊ expire_logs_days 546
◊ flush 546
◊ flush_time 546
◊ foreign_key_checks 164
◊ ft_boolean_syntax 546
◊ ft_max_word_len 547
◊ ft_min_word_len 301, 547
◊ ft_query_expansion_limit 547
◊ ft_stopword_file 547
◊ general_log 547
◊ general_log_file 547
◊ group_concat_max_len 547
◊ have_archive 547
◊ have_bdb 547
◊ have_blackhole_engine 548
◊ have_compress 548
◊ have_crypt 548
◊ have_csv 548
◊ have_dynamic_loading 548
◊ have_example_engine 548
◊ have_federated_engine 548
◊ have_geometry 548
◊ have_innodb 548
◊ have_ndbcluster 549
◊ have_openssl 549
◊ have_partitioning 465, 549
◊ have_query_cache 549
◊ have_row_based_replication 549
◊ have_rtree_keys 549
◊ have_symlink 549
◊ init_connect 549
◊ init_file 550
◊ init_slave 550
◊ interactive_timeout 550
◊ join_buffer_size 550
◊ key_buffer_size 550
◊ key_cache_age_threshold 550
◊ key_cache_block_size 550
◊ key_cache_division_limit 550
◊ language 550
◊ large_file_support 551
◊ large_pages 551
◊ lc_time_names 551
◊ license 551
◊ local_infile 551
◊ locked_in_memory 551
◊ log 551
◊ log_bin 551
◊ log_bin_trust_function_creators 551
◊ log_error 552
◊ log_output 552
◊ log_queries_not_using_indexes 552
◊ log_slave_updates 552
◊ log_slow_queries 552
◊ log_warnings 552
◊ long_query_time 552
◊ low_priority_updates 553
◊ lower_case_file_system 553

◊ lower_case_table_names 553
◊ max_allowed_packet 553, 556
◊ max_binlog_cache_size 553
◊ max_binlog_size 70, 553
◊ max_connect_errors 553
◊ max_connections 553
◊ max_error_count 554
◊ max_heap_table_size 554
◊ max_insert_delayed_threads 554
◊ max_join_size 554
◊ max_length_for_sort_data 554
◊ max_prepared_stmt_count 554, 557
◊ max_relay_log_size 554
◊ max_seeks_for_key 554
◊ max_sort_length 555
◊ max_sp_recursion_depth 416
◊ max_tmp_tables 555
◊ max_user_connections 555
◊ max_write_lock_count 555
◊ myisam_data_pointer_size 555
◊ myisam_max_sort_file_size 555
◊ myisam_recover_options 555
◊ myisam_repair_threads 555
◊ myisam_sort_buffer_size 555
◊ myisam_stats_method 555
◊ myisam_use_mmap 556
◊ named_pipe 556
◊ net_buffer_length 556
◊ net_read_timeout 556
◊ net_retry_count 556
◊ net_write_timeout 556
◊ new 556
◊ old_passwords 556
◊ open_files_limit 556
◊ optimizer_prune_level 556
◊ optimizer_search_depth 557
◊ pid_file 557
◊ plugin_dir 557
◊ port 557
◊ preload_buffer_size 557
◊ prepared_stmt_count 557
◊ protocol_version 557
◊ query_alloc_block_size 557
◊ query_cache_limit 557
◊ query_cache_min_res_unit 557
◊ query_cache_size 557
◊ query_cache_type 558
◊ query_cache_wlock_invalidate 558
◊ query_prealloc_size 558
◊ range_alloc_block_size 558
◊ read_buffer_size 558
◊ read_only 558
◊ read_rnd_buffer_size 559
◊ relay_log_purge 558
◊ rpl_recovery_rank 559
◊ secure_auth 559
◊ server_id 559
◊ shared_memory 559
◊ shared_memory_base_name 559
◊ skip_external_locking 559
◊ skip_networking 559
◊ skip_show_database 559
◊ slave_compressed_protocol 559
◊ slave_load_tmpdir 560
◊ slave_net_timeout 560
◊ slave_skip_errors 560
◊ slow_launch_time 560
◊ slow_queries 552
◊ slow_query_log 560
◊ slow_query_log_file 560
◊ socket 560
◊ sort_buffer_size 560
◊ sql_mode 560
◊ SQL_QUOTE_SHOW_CREATE 107
◊ sql_slave_skip_counter 560
◊ ssl_ca 560
◊ ssl_capath 560
◊ ssl_cert 560
◊ ssl_cipher 561
◊ ssl_key 561
◊ storage_engine 561
◊ sync_binlog 561
◊ sync_frm 561
◊ system_time_zone 561
◊ table_cache 561
◊ table_definition_cache 561
◊ table_lock_wait_timeout 561
◊ table_open_cache 561
◊ table_type 561, 562
◊ thread_cache_size 562
◊ thread_concurrency 562
◊ thread_stack 562
◊ time_format 562
◊ time_zone 562
◊ tmp_table_size 562
◊ tmpdir 562
◊ transaction_alloc_block_size 562
◊ transaction_prealloc_size 562
◊ tx_isolation 562
◊ updatable_views_with_limit 562

Системная переменная *(прод.)*:
◊ version 563
◊ version_bdb 563
◊ version_comment 563
◊ version_compile_machine 563
◊ version_compile_os 563
◊ wait_timeout 563
Соединение с удаленным хостом 45
Сортировка 441
Ссылка, внешняя 230
Статус операции 122
Столбец:
◊ добавление 115
◊ изменение 116
◊ кодировка 95, 103
◊ полное имя 211
◊ привилегии 353
◊ псевдоним 191, 214
◊ сортировка 95
◊ удаление 116
◊ уникальные значения 200

Т

Таблица:
◊ ALTER TABLE 335
◊ Archive 155
◊ ARCHIVE 547
◊ BDB 333, 335, 541, 547, 562, 563
◊ BLACKHOLE 548
◊ CREATE TABLE 555
◊ CSV 548
◊ DUAL 197
◊ EXAMPLE 548
◊ FEDERATED 111—113, 548
◊ HEAP 108, 333, 554
◊ InnoDB 17, 95, 135, 333, 335, 338, 548, 562
◊ MEMORY 108, 135, 155, 554
◊ MERGE 109, 110
◊ MRG_MYISAM 110
◊ MyISAM 17, 64, 104, 105, 109, 112, 122, 135, 155, 295, 333, 335, 339, 545, 550, 555, 558, 562
◊ NDB Cluster 549
◊ блокировка 339
◊ восстановление 121
  ▫ из резервной копии 66
◊ временная 105, 114, 555
◊ изменение структуры 114
◊ контрольная сумма 103
◊ объединение 205
◊ привилегии 353
◊ псевдоним 214
◊ резервное копирование 64
◊ результирующая 171, 173
◊ самообъединение 214
◊ создание 506
◊ сравнение имен 553
◊ структура 82
Тип данных:
◊ BIGINT 83, 131, 152, 279
◊ BIT 84
◊ BLOB 86, 94, 105, 108, 262, 555
◊ BOOLEAN 84
◊ CHAR 86, 105, 262, 295
◊ DATE 145
◊ DEC 84
◊ DECIMAL 84
◊ ENUM 86
◊ INT 83, 131, 152
◊ MEDIUMINT 83
◊ NULL 89, 177
◊ NUMERIC 84
◊ SET 86, 265
◊ SMALLINT 83
◊ TEXT 86, 94, 105, 108, 262, 295, 555
◊ TIMESTAMP 94, 148
◊ TINYINT 83, 152
◊ VARCHAR 86, 105, 262, 295
◊ временной, форматирование 188
◊ строковый 85, 144
Транзакция 333, 544, 553
◊ режим автоматического завершения 335
Триггер 419
◊ AFTER 419
◊ BEFORE 419
◊ DELETE 420
◊ INSERT 420
◊ NEW 420
◊ OLD 420
◊ UPDATE 420
◊ временной 453

У

Удаленный хост 45, 111, 112
Универсальный уникальный идентификатор 279, 282

Утилита:
◊ myisampack 35
◊ mysql 34, 35, 275
◊ MySQL Administrator 54
◊ MySQL Command Line Client 23
◊ MySQL Control Center 54
◊ MySQL Query Browser 54
◊ MySQL Server Instance Configuration Wizard 16
◊ mysqladmin 34, 350
◊ mysqlbinlog 35
◊ mysqlcheck 35
◊ mysqldump 30, 34, 56, 64
◊ mysqlhotcopy 34, 55
◊ mysqlimport 34
◊ mysqlshow 34
◊ perror 35
◊ replace 35
◊ tar 57
Учетная запись 343
◊ % 347
◊ блокировка 559
◊ ограничения 356
◊ привилегии 344, 348
  ▫ удаление 355
◊ удаление 345
◊ форма 346

Ф

Файл:
◊ .frm 64, 67, 108, 109, 111, 123, 438
◊ .MRG 109
◊ .MYD 64, 67, 104, 111
◊ .MYI 64, 67, 104, 109, 123
◊ db.opt 75
◊ DES 267
◊ my.cnf 301, 326, 327, 549
◊ my.ini 301, 326, 327, 549
◊ trg 422
◊ журнальный 67, 546, 551, 552, 558
◊ текстовый 58, 59
Форматирование даты 188
Функция:
◊ ABS() 233, 378
◊ ACOS() 233
◊ ADDDATE() 240, 245
◊ ADDTIME() 240
◊ AES_DECRYPT() 267, 268
◊ AES_ENCRYPT() 266, 268
◊ ASCII() 254, 473
◊ ASIN() 233
◊ ATAN() 234
◊ ATAN2() 234
◊ AVG() 307, 308
◊ BENCHMARK() 275
◊ BIN() 254
◊ BIT_AND() 307
◊ BIT_LENGTH() 254
◊ BIT_OR() 307
◊ BIT_XOR() 307
◊ CASE() 271
◊ CEIL() 234
◊ CEILING() 234, 238
◊ CHAR() 254
◊ CHAR_LENGTH() 254
◊ CHARSET() 254
◊ COALESCE() 234
◊ COLLATION() 254
◊ COMPRESS() 254, 262, 548
◊ CONCAT() 157, 254, 260, 376, 394
◊ CONCAT_WS() 254
◊ CONNECTION_ID() 275
◊ CONV() 254
◊ CONVERT() 254, 258
◊ CONVERT_TZ() 240
◊ COS() 234
◊ COT() 234
◊ COUNT() 191, 198, 201, 202, 307, 312
◊ CRC32() 234
◊ CURDATE() 240
◊ CURRENT_TIME() 241, 242
◊ CURRENT_TIMESTAMP() 242
◊ CURRENT_USER() 275
◊ CURTIME() 241
◊ DATABASE() 275, 276
◊ DATE() 188, 240, 241
◊ DATE_ADD() 240, 245
◊ DATE_FORMAT() 188, 241, 250, 551
◊ DATE_SUB() 243
◊ DATEDIFF() 241
◊ DAY() 241
◊ DAYNAME() 241, 551
◊ DAYOFMONTH() 241
◊ DAYOFWEEK() 241
◊ DAYOFYEAR() 241
◊ DECODE() 267
◊ DEFAULT() 278
◊ DEGREES() 234
◊ DES_DECRYPT() 267

Файл *(прод.)*:
◊ DES_ENCRYPT() 267
◊ ELT() 254
◊ ENCODE() 267
◊ ENCRYPT() 267, 548
◊ EXP() 234
◊ EXPORT_SET() 255
◊ EXTRACT() 241
◊ FIELD() 255
◊ FIND_IN_SET() 255, 265
◊ FLOOR() 234, 238, 251
◊ FORMAT() 255
◊ FOUND_ROWS() 275, 277
◊ FROM_DAYS() 241
◊ FROM_UNISTIME() 378
◊ FROM_UNIX() 150
◊ FROM_UNIXTIME() 241, 252
◊ GET_FORMAT() 241
◊ GET_LOCK() 278, 280
◊ GREATEST() 234
◊ GROUP_CONCAT() 307, 315
◊ GROUP_CONTACT() 547
◊ HEX() 255
◊ HOUR() 242
◊ IF() 271
◊ IFNULL() 271, 273
◊ INET_ATON() 278, 279
◊ INET_NTOA() 278, 280
◊ INSERT() 255
◊ INSTR() 255
◊ INTERVAL() 234
◊ IS_FREE_LOCK() 278, 281
◊ IS_USED_LOCK() 278, 282
◊ LAST_DAY() 242
◊ LAST_INSERT_ID() 151, 275
◊ LCASE() 256
◊ LEAST() 234
◊ LEFT() 255
◊ LENGTH() 256, 264
◊ LN() 234
◊ LOAD_FILE() 256, 263
◊ LOCALTIME 242
◊ LOCALTIME() 242
◊ LOCALTIMESTAMP 242
◊ LOCALTIMESTAMP() 242
◊ LOCATE() 256
◊ LOG() 234
◊ LOG10() 234
◊ LOG2() 234
◊ LOWER() 256, 261
◊ LPAD() 256
◊ LTRIM() 256
◊ MAKE_SET() 256
◊ MAKEDATE() 242
◊ MAKETIME() 242
◊ MAX() 199, 308, 318
◊ MD5() 268, 269
◊ MICROSECOND() 242
◊ MID() 256
◊ MIN() 199, 308, 317
◊ MINUTE() 242
◊ miscellaneous 278
◊ MOD() 234
◊ MONTH() 188, 242
◊ MONTHNAME() 242, 551
◊ NAME_CONST() 278
◊ NOW() 146, 197, 242, 249
◊ NULLIF() 271
◊ OCT() 256
◊ OCTET_LENGTH() 256
◊ OLD_PASSWORD() 268, 271
◊ ORD() 256
◊ PASSWORD() 268, 344, 346, 476
◊ PERIOD_ADD() 242
◊ PERIOD_DIFF() 242
◊ PI() 234
◊ POSITION() 256
◊ POW() 234
◊ POWER() 234
◊ QUARTER() 243
◊ QUOTE() 257
◊ RADIANS() 235, 236
◊ RAND() 235, 239
◊ RELEASE_LOCK() 279, 280
◊ REPEAT() 257
◊ REPLACE() 257
◊ REVERSE() 257
◊ RIGHT() 257
◊ ROUND() 235, 237
◊ ROW_COUNT() 275
◊ RPAD() 257
◊ RTRIM() 257
◊ SCHEMA() 275
◊ SEC_TO_TIME() 243
◊ SECOND() 243
◊ SESSION_USER() 275
◊ SHA1() 268
◊ SIGN() 235
◊ SIN() 235, 236
◊ SLEEP() 279

◊ SOUNDEX() 257
◊ SPACE() 257
◊ SQRT() 235
◊ STD() 308
◊ STDDEV() 308
◊ STDDEV_POP() 308
◊ STDDEV_SAMP() 308
◊ STR_TO_DATE() 243
◊ SUBDATE() 243, 248
◊ SUBSTRING() 256, 259, 473
◊ SUBSTRING_INDEX() 257
◊ SUBTIME() 243
◊ SUM() 308, 318
◊ SYSTEM_USER() 275
◊ TAN() 235
◊ TIME() 241, 243
◊ TIME_FORMAT() 243
◊ TIME_TO_SEC() 244
◊ TIMEDIFF() 243
◊ TIMESTAMP() 243
◊ TIMESTAMPADD() 243
◊ TIMESTAMPDIFF() 243, 251, 377
◊ TO_DAYS() 244, 251
◊ TRIM() 258
◊ TRUNCATE() 235, 238
◊ UCASE() 258
◊ UNCOMPRESS() 258, 262, 548
◊ UNCOMPRESSED_LENCTH() 258
◊ UNHEX() 258
◊ UNIX_TIMESTAMP() 149, 244
◊ UPPER() 222, 258, 261
◊ USER() 275
◊ UTC_DATE() 244
◊ UTC_TIME() 244
◊ UTC_TIMESTAMP() 244
◊ UUID() 279, 282
◊ VAR_POP() 308
◊ VAR_SAMP() 308
◊ VARIANCE() 308
◊ VERSION() 197, 275, 276
◊ WEEK() 244, 544
◊ WEEKDAY() 244
◊ WEEKOFYEAR() 244
◊ YEAR() 188, 244
◊ YEARWEEK() 245
◊ агрегатная 307
◊ информационная 274
◊ календарная 240
◊ конструкции GROUP BY 307
◊ математическая 233
◊ строковая 253
◊ суммирующая 307
◊ управления потоком 271
◊ шифрования 266

Х

Хост, локальный 347
Хранимая процедура 413
◊ COMMENT 381, 404
◊ CONTAINS SQL 380
◊ DETERMINISTIC 379
◊ LANGUAGE SQL 379
◊ MODIFIES SQL DATA 380
◊ NO SQL 380
◊ NOT DETERMINISTIC 379
◊ READ SQL DATA 380
◊ SQL SECURITY 380, 404, 431
◊ групповые характеристики 379
◊ изменение 404
◊ имя 398, 400
◊ параметр 365
◊ создание 365
◊ тело 366
◊ тип 400
◊ удаление 402

Ч

Часовой пояс 561, 562

Ш

Шифрование:
◊ необратимое 267, 269
◊ обратимое 268