Максим Кузнецов
Игорь Симдянов

bhv

MySQL 5

- Полный список функций MySQL 5
- Язык запросов SQL
- Хранимые процедуры
- Локализация
- Представления
- Тонкости администрирования
- Взаимодействие с C++, PHP и Perl

Наиболее полное руководство



В ПОДЛИННИКЕ®

Максим Кузнецов
Игорь Симдянов

MySQL 5

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.06
ББК 32.973.26-018.1
К89

Кузнецов, М. В.

К89 MySQL 5 / М. В. Кузнецов, И. В. Симдянов. — СПб.: БХВ-Петербург, 2006. — 1024 с.: ил.

ISBN 5-94157-928-4

Описывается новая версия популярной СУБД MySQL, в том числе вложенные запросы, хранимые процедуры, представления, триггеры, курсоры, информационная схема, пространственные расширения, репликация и другие элементы. Большое внимание уделяется администрированию и настройке MySQL. Приводится большое число примеров самых разнообразных SQL-запросов и программ на языках C/C++, Perl и PHP.

Для программистов и разработчиков баз данных

УДК 681.3.06
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 15.08.06.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 82,56.
Тираж 3000 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 ОАО "Техническая книга"
190005, Санкт-Петербург, Измайловский пр., 29.

ISBN 5-94157-928-4

© Кузнецов М. В., Симдянов И. В., 2006
© Оформление, издательство "БХВ-Петербург", 2006

Оглавление

Введение **1**

Для кого и о чем эта книга 1
Благодарности 2

ЧАСТЬ I. ЗНАКОМСТВО С SQL **3**

Глава 1. История развития баз данных. Понятие реляционной базы данных **5**

1.1. История развития СУБД. Реляционные базы данных 6
1.1.1. Иерархические базы данных 6
1.1.2. Сетевые базы данных 8
1.2. Особенности реляционных баз данных 9
1.2.1. Первичные ключи 11
1.2.2. Нормализация базы данных 13
1.3. СУБД и сети 14
1.3.1. Централизованная архитектура 15
1.3.2. Архитектура "клиент-сервер" 16
1.3.3. Трехуровневая архитектура Интернета 17
1.3.4. Кластерная модель 19
1.4. Как работают базы данных и что такое SQL? 20
1.5. Версии MySQL 21
1.5.1. Что нового в MySQL 4.1? 23
1.5.2. Что нового в MySQL 5.0? 23

Глава 2. Установка MySQL 5 ..... **24**

2.1. Получение дистрибутива 24
2.2. Установка на платформу Windows 25
2.3. Установка на платформу Linux 39

Глава 3. Работа с утилитами MySQL **43**

3.1. Утилита *mysql* 44
3.2. Утилита *mysqldump* 63

Глава 4. Создание баз данных и таблиц. Типы данных 69

4.1. Создание базы данных 69
4.2. Создание таблицы 73
4.3. Типы данных 76
4.3.1. Числовые данные 77
4.3.2. Строковые данные 80
4.3.3. Календарные данные 83
4.3.4. Тип данных *NULL* 86
4.3.5. Выбор типа данных 88
4.4. Учебная база данных 89

Глава 5. Индексы 93

5.1. Первичный ключ 94
5.2. Обычный и уникальный индексы 98

Глава 6. Добавление данных 103

6.1. Однострочный оператор *INSERT* 103
6.2. Многострочный оператор *INSERT* 110
6.3. Отложенная вставка записей 111
6.4. Пакетная загрузка данных 111
6.5. Утилита *mysqlimport* 115

Глава 7. Выборка данных 118

7.1. Изменение количества и порядка следования столбцов 118
7.2. Условия 121
7.3. Сортировка 124
7.4. Ограничение выборки 128
7.5. Использование функций 129
7.6. Группировка записей 132
7.7. Объединение таблиц 138
7.8. Сохранение результатов во внешний файл 143

Глава 8. Многотабличные запросы 147

8.1. Перекрестное объединение таблиц 147
8.2. Объединение таблиц при помощи *JOIN* 159

Глава 9. Удаление данных 167

9.1. Оператор *DELETE* 167
9.2. Оператор *TRUNCATE TABLE* 168
9.3. Удаление из нескольких таблиц 169

Глава 10. Обновление данных 173

10.1. Оператор *UPDATE* 173
10.2. Многотабличный оператор *UPDATE* 178
10.3. Оператор *REPLACE* 181

ЧАСТЬ II. СЛОЖНЫЕ ВОПРОСЫ MYSQL 183

Глава 11. Типы и структура таблиц 185

11.1. MyISAM 185
11.2. MERGE 186
11.3. MEMORY (HEAP) 188
11.4. EXAMPLE 189
11.5. BDB (BerkeleyDB) 190
11.6. InnoDB 191
11.7. NDB Cluster 192
11.8. ARCHIVE 192
11.9. CSV 193
11.10. FEDERATED 194
11.11. BLACKHOLE 196

Глава 12. Создание таблиц и удаление таблиц 198

12.1. Оператор *CREATE TABLE* 198
12.1.1. Структура таблицы 199
12.1.2. Параметры таблицы 206
12.2. Оператор *DROP TABLE* 211

Глава 13. Редактирование структуры таблиц 213

13.1. Добавление и удаление столбцов 213
13.2. Изменение уже существующих столбцов 216
13.3. Добавление и удаление индексов 218
13.4. Преобразование параметров таблицы 220

Глава 14. Приведение типов 225

14.1. Ключевое слово *BINARY* 225
14.2. Функция *CAST()* 226
14.3. Функция *CONVERT()* 228
14.4. Поддержка кодировок 229
14.5. Преобразование кодировок 237
14.6. Применение ключевого слова *COLLATE* 238

Глава 15. Операторы и математические функции 240

15.1. Операторы 240
15.1.1. Арифметические операторы 240
15.1.2. Операторы сравнения 245
15.1.3. Логические операторы 257
15.1.4. Битовые операторы 259
15.1.5. Приоритет операторов 265
15.2. Математические функции 266
15.2.1. Функция *ABS()* 266
15.2.2. Функция *ACOS()* 267

15.2.3. Функция *ASIN()* 267
15.2.4. Функция *ATAN()* 267
15.2.5. Функция *ATAN2()* 268
15.2.6. Функция *CEILING()* 268
15.2.7. Функция *COS()* 269
15.2.8. Функция *COT()* 270
15.2.9. Функция *CRC32()* 270
15.2.10. Функция *DEGREES()* 270
15.2.11. Функция *EXP()* 271
15.2.12. Функция *FLOOR()* 271
15.2.13. Функция *LOG()* 272
15.2.14. Функция *LOG2()* 272
15.2.15. Функция *LOG10()* 273
15.2.16. Функция *MOD()* 273
15.2.17. Функция *PI()* 273
15.2.18. Функция *POW()* 273
15.2.19. Функция *RADIANS()* 274
15.2.20. Функция *RAND()* 274
15.2.21. Функция *ROUND()* 275
15.2.22. Функция *SIGN()* 276
15.2.23. Функция *SIN()* 277
15.2.24. Функция *SQPT()* 277
15.2.25. Функция *TAN()* 277
15.2.26. Функция *TRUNCATE()* 278

Глава 16. Функции даты и времени 279

16.1. Функция *ADDDATE()* 279
16.2. Функция *ADDTIME()* 285
16.3. Функция *CONVERT_TZ()* 286
16.4. Функция *CURDATE()* 286
16.5. Функция *CURTIME()* 287
16.6. Функция *DATE()* 287
16.7. Функция *DATEDIFF()* 288
16.8. Функция *DATE_FORMAT()* 288
16.9. Функция *DAY()* 291
16.10. Функция *DAYNAME()* 291
16.11. Функция *DAYOFMONTH()* 292
16.12. Функция *DAYOFWEEK()* 293
16.13. Функция *DAYOFYEAR()* 293
16.14. Функция *EXTRACT()* 293
16.15. Функция *FROM_DAYS()* 295
16.16. Функция *FROM_UNIXTIME()* 295
16.17. Функция *GET_FORMAT()* 297
16.18. Функция *HOUR()* 298
16.19. Функция *LAST_DAY()* 299

16.20. Функция *MAKEDATE()* 300
16.21. Функция *MAKETIME()* 301
16.22. Функция *MICROSECOND()* 301
16.23. Функция *MINUTE()* 302
16.24. Функция *MONTH()* 302
16.25. Функция *MONTHNAME()* 302
16.26. Функция *NOW()* 303
16.27. Функция *PERIOD_ADD()* 304
16.28. Функция *PERIOD_DIFF()* 304
16.29. Функция *QUARTER()* 305
16.30. Функция *SECOND()* 305
16.31. Функция *SEC_TO_TIME()* 305
16.32. Функция *STR_TO_DATE()* 306
16.33. Функция *SUBDATE()* 307
16.34. Функция *SUBTIME()* 307
16.35. Функция *TIME()* 308
16.36. Функция *TIMEDIFF()* 309
16.37. Функция *TIMESTAMP()* 309
16.38. Функция *TIMESTAMPADD()* 310
16.39. Функция *TIMESTAMPDIFF()* 311
16.40. Функция *TIME_FORMAT()* 312
16.41. Функция *TIME_TO_SEC()* 312
16.42. Функция *TO_DAYS()* 313
16.43. Функция *UNIX_TIMESTAMP()* 314
16.44. Функция *UTC_DATE()* 314
16.45. Функция *UTC_TIME()* 315
16.46. Функция *UTC_TIMESTAMP()* 315
16.47. Функция *WEEK()* 316
16.48. Функция *WEEKDAY()* 317
16.49. Функция *WEEKOFYEAR()* 317
16.50. Функция *YEAR()* 318
16.51. Функция *YEARWEEK()* 318

Глава 17. Строковые функции 320

17.1. Функция *ASCII()* 320
17.2. Функция *BIN()* 321
17.3. Функция *BIT_LENGTH()* 321
17.4. Функция *CHAR()* 321
17.5. Функция *CHAR_LENGTH()* 323
17.6. Функция *CHARSET()* 324
17.7. Функция *COLLATION()* 325
17.8. Функция *COMPRESS()* 325
17.9. Функция *CONCAT()* 327
17.10. Функция *CONCAT_WS()* 328
17.11. Функция *CONV()* 329

17.12. Функция *ELT()* 329
17.13. Функция *EXPORT_SET()* 330
17.14. Функция *FIELD()* 331
17.15. Функция *FIND_IN_SET()* 331
17.16. Функция *FORMAT()* 332
17.17. Функция *HEX()* 333
17.18. Функция *INSERT()* 333
17.19. Функция *INSTR()* 334
17.20. Функция *LEFT()* 335
17.21. Функция *LENGTH()* 335
17.22. Функция *LOAD_FILE()* 335
17.23. Функция *LOCATE()* 336
17.24. Функция *LOWER()* 337
17.25. Функция *LPAD()* 337
17.26. Функция *LTRIM()* 338
17.27. Функция *MAKE_SET()* 338
17.28. Функция *MID()* 338
17.29. Функция *OCT()* 340
17.30. Функция *ORD()* 340
17.31. Функция *POSITION()* 341
17.32. Функция *QUOTE()* 341
17.33. Функция *REPEAT()* 341
17.34. Функция *REPLACE()* 342
17.35. Функция *REVERSE()* 344
17.36. Функция *RIGHT()* 344
17.37. Функция *RPAD()* 345
17.38. Функция *RTRIM()* 345
17.39. Функция *SOUNDEX()* 346
17.40. Функция *SPACE()* 346
17.41. Функция *SUBSTRING()* 346
17.42. Функция *SUBSTRING_INDEX()* 348
17.43. Функция *TRIM()* 348
17.44. Функция *UNCOMPRESS()* 349
17.45. Функция *UNCOMPRESSED_LENGTH()* 350
17.46. Функция *UNHEX()* 351
17.47. Функция *UPPER()* 351

Глава 18. Безопасность и MySQL 352

18.1. Функции *AES_ENCRYPT()* и *AES_DECRYPT()* 352
18.2. Функции *ENCODE()* и *DECODE()* 355
18.3. Функции *DES_ENCRYPT()* и *DES_DECRYPT()* 355
18.4. Функция *ENCRYPT()* 356
18.5. Функция *MD5()* 357
18.6. Функция *PASSWORD()* 358
18.7. Функция *SHA1()* 359

Глава 19. Поиск и регулярные выражения 361

19.1. Оператор *LIKE* 361
19.2. Оператор *NOT LIKE* 366
19.3. Оператор *SOUND LIKE* 367
19.4. Оператор *RLIKE (REGEXP)* 367
19.5. Оператор *NOT RLIKE* 379
19.6. Функция *STRCMP()* 380

Глава 20. Полнотекстовый поиск 381

20.1. Индекс *FULLTEXT* 381
20.2. Конструкция *MATCH (...) AGAINST (...)* 383
20.3. Логический режим 388
20.4. Режим расширения запроса 395

Глава 21. Функции, применяемые вместе с конструкцией *GROUP BY* 397

21.1. Функция *AVG()* 397
21.2. Функция *BIT_AND()* 400
21.3. Функция *BIT_OR()* 401
21.4. Функция *BIT_XOR()* 402
21.5. Функция *COUNT()* 403
21.6. Функция *GROUP_CONCAT()* 406
21.7. Функция *MIN()* 408
21.8. Функция *MAX()* 409
21.9. Функция *STD()* 410
21.10. Функция *STDDEV_SAMP()* 411
21.11. Функция *SUM()* 412
21.12. Функция *VAR_POP()* 412
21.13. Функция *VAR_SAMP()* 413
21.14. Конструкция *WITH ROLLUP()* 413

Глава 22. Разные функции 417

22.1. Функции управления потоком выполнения 417
22.1.1. Функция *CASE()* 417
22.1.2. Функция *IF()* 418
22.1.3. Функция *IFNULL()* 420
22.1.4. Функция *NULLIF()* 421
22.2. Информационные функции 421
22.2.1. Функция *BENCHMARK()* 422
22.2.2. Функция *CONNECTION_ID()* 422
22.2.3. Функция *CURRENT_USER()* 423
22.2.4. Функция *DATABASE()* 423
22.2.5. Функция *FOUND_ROWS()* 424
22.2.6. Функция *LAST_INSERT_ID()* 425
22.2.7. Функция *ROW_COUNT()* 427

22.2.8. Функция *USER()* 429
22.2.9. Функция *VERSION()* 429
22.3. Разные функции 430
22.3.1. Функция *DEFAULT()* 430
22.3.2. Функция *GET_LOCK()* 432
22.3.3. Функция *INET_ATON()* 433
22.3.4. Функция *INET_NTOA()* 433
22.3.5. Функция *IS_FREE_LOCK()* 434
22.3.6. Функция *IS_USED_LOCK()* 435
22.3.7. Функция *NAME_CONST()* 435
22.3.8. Функция *RELEASE_LOCK()* 436
22.3.9. Функция *SLEEP()* 436
22.3.10. Функция *UUID()* 436

Глава 23. Переменные и временные таблицы 438

23.1. Переменные SQL 438
23.2. Временные таблицы 442

Глава 24. Вложенные запросы 446

24.1. Вложенный запрос как скалярный операнд 449
24.2. Вложенные запросы, возвращающие несколько строк 452
24.2.1. Ключевое слово *IN* 453
24.2.2. Ключевое слово *ANY (SOME)* 454
24.2.3. Ключевое слово *ALL* 456
24.3. Проверка на существование 458
24.4. Коррелированные запросы 461
24.5. Вложенные запросы, возвращающие несколько столбцов 461
24.6. Подзапросы в конструкции *FROM* 463
24.7. Вложенные запросы в операторе *CREATE TABLE* 466
24.8. Вложенные запросы в операторе *INSERT* 469

Глава 25. Внешние ключи и ссылочная целостность 473

Глава 26. Транзакции и блокировки 481

26.1. Транзакции 481
26.2. Блокировка таблиц 487

Глава 27. Управление учетными записями пользователей 490

27.1. Учетные записи СУБД MySQL 490
27.2. Оператор *CREATE USER* 493
27.3. Оператор *DROP USER* 495
27.4. Оператор *RENAME USER* 497
27.5. Оператор *GRANT* 498
27.6. Оператор *REVOKE* 508

ЧАСТЬ III. СРЕДСТВА АДМИНИСТРИРОВАНИЯ СУБД MYSQL 509

Глава 28. Администрирование СУБД MySQL.....511

28.1. Параметры запуска сервера MySQL 511
28.2. Системные переменные сервера 519
28.3. Режим SQL-сервера.....540
28.4. Журнальные файлы 544
28.4.1. Журнальные таблицы 544
28.4.2. Журнал ошибок 545
28.4.3. Общий журнал запросов 546
28.4.4. Бинарный журнал регистраций 546
28.4.5. Утилита *mysqlbinlog* 549
28.4.6. Журнал медленных запросов.....551
28.5. Оператор *CACHE INDEX*.....552
28.6. Оператор *FLUSH*.....553
28.7. Оператор *KILL* 554
28.8. Оператор *LOAD INDEX INTO CACHE*.....555
28.9. Оператор *RESET*.....555
28.10. Утилита *mysqladmin* 556

Глава 29. Оператор *SET*.....560

Глава 30. Оператор *SHOW*.....573

30.1. Оператор *SHOW CHARACTER SET* 574
30.2. Оператор *SHOW COLLATION* 575
30.3. Оператор *SHOW COLUMNS*.....576
30.4. Оператор *SHOW CREATE DATABASE*.....579
30.5. Оператор *SHOW CREATE TABLE* 579
30.6. Оператор *SHOW DATABASES* 580
30.7. Оператор *SHOW ENGINES* 581
30.8. Оператор *SHOW ENGINE* 582
30.9. Оператор *SHOW ERRORS*.....583
30.10. Оператор *SHOW GRANTS*.....585
30.11. Оператор *SHOW INDEX*.....586
30.12. Оператор *SHOW PLUGIN* 589
30.13. Оператор *SHOW PRIVILEGES*.....589
30.14. Оператор *SHOW PROCESSLIST*.....593
30.15. Оператор *SHOW STATUS*.....596
30.16. Оператор *SHOW TABLE STATUS* 601
30.17. Оператор *SHOW TABLES*.....604
30.18. Оператор *SHOW VARIABLES* 606
30.19. Оператор *SHOW WARNINGS*.....607
30.20. Утилита *mysqlshow* 610

Глава 31. Предотвращение катастроф и восстановление 613

31.1. Оператор *CHECK TABLE* 613
31.2. Оператор *ANALYZE TABLE* 616
31.3. Оператор *CHECKSUM TABLE* 617
31.4. Оператор *OPTIMIZE TABLE* 619
31.5. Оператор *REPAIR TABLE* 620
31.6. Оператор *BACKUP TABLE* 622
31.7. Оператор *RESTORE TABLE* 623
31.8. Резервное копирование 625
31.9. Утилита *mysqlcheck* 627

Глава 32. Репликация в MySQL 630

32.1. Введение в репликацию 630
32.2. Детали реализации процесса репликации 631
32.2.1. Состояние потока репликации главного сервера 634
32.2.2. Состояние потока ввода/вывода подчиненного сервера 634
32.2.3. Состояние потока обработки ретрансляционных журналов подчиненного сервера 636
32.2.4. Журнал ретрансляции и файлы состояния 636
32.3. Настройка репликации 638
32.4. Совместимость репликации между версиями MySQL 641
32.5. Параметры запуска репликации 641
32.6. Операторы управления главным сервером 646
32.6.1. Оператор *PURGE MASTER LOGS* 647
32.6.2. Оператор *RESET MASTER* 647
32.6.3. Оператор *SET SQL_BIN_LOG* 647
32.6.4. Оператор *SHOW BINLOG EVENTS* 648
32.6.5. Оператор *SHOW MASTER LOGS* 648
32.6.6. Оператор *SHOW MASTER STATUS* 648
32.6.7. Оператор *SHOW SLAVE HOSTS* 649
32.7. Операторы управления подчиненными серверами 649
32.7.1. Оператор *CHANGE MASTER TO* 649
32.7.2. Оператор *LOAD DATA FROM MASTER* 651
32.7.3. Оператор *LOAD TABLE FROM MASTER* 652
32.7.4. Функция *MASTER_POS_WAIT()* 652
32.7.5. Оператор *RESET SLAVE* 653
32.7.6. Оператор *SET GLOBAL SQL_SLAVE_SKIP_COUNTER* 653
32.7.7. Оператор *SHOW SLAVE STATUS* 653
32.7.8. Оператор *START SLAVE* 656
32.7.9. Оператор *STOP SLAVE* 657

ЧАСТЬ IV. НОВОВВЕДЕНИЯ MYSQL 5.0 659

Глава 33. Хранимые процедуры 661

33.1. Хранимые процедуры и привилегии 662
33.2. Создание хранимой процедуры 662
33.2.1. Тело процедуры 664
33.2.2. Параметры процедуры 668
33.2.3. Работа с таблицами базы данных 671
33.2.4. Хранимые функции 679
33.3. Группа характеристик хранимых процедур 681
33.4. Операторы управления потоком данных 684
33.4.1. Оператор *IF...THEN...ELSE* 684
33.4.2. Оператор *CASE* 688
33.4.3. Оператор *WHILE* 691
33.4.4. Оператор *REPEAT* 696
33.4.5. Оператор *LOOP* 697
33.4.6. Оператор *GOTO* 698
33.5. Метаданные 700
33.5.1. Оператор *SHOW PROCEDURE STATUS* 700
33.5.2. Оператор *SHOW CREATE* 702
33.5.3. Извлечение информации из таблицы *mysql.proc* 703
33.6. Удаление хранимых процедур 706
33.7. Редактирование хранимых процедур 707
33.8. Обработчики ошибок 708
33.9. Курсоры 714

Глава 34. Триггеры 718

34.1. Оператор *CREATE TRIGGER* 718
34.2. Оператор *DROP TRIGGER* 722

Глава 35. Представления 723

35.1. Создание представлений 724
35.2. Удаление представлений 738
35.3. Редактирование представлений 739
35.4. Оператор *SHOW CREATE VIEW* 739

Глава 36. Информационная схема 741

36.1. Представление *CHARACTER_SETS* 742
36.2. Представление *COLLATIONS* 745
36.3. Представление *COLLATION_CHARACTER_SET_APPLICABILITY* 747
36.4. Представление *COLUMN_PRIVILEGES* 748
36.5. Представление *COLUMNS* 751
36.6. Представление *KEY_COLUMN_USAGE* 755
36.7. Представление *ROUTINES* 757

36.8. Представление *SCHEMA_PRIVILEGES* 761
36.9. Представление *SCHEMATA* 763
36.10. Представление *STATISTICS* 765
36.11. Представление *TABLE_CONSTRAINTS* 769
36.12. Представление *TABLE_PRIVILEGES* 770
36.13. Представление *TABLES* 772
36.14. Представление *USER_PRIVILEGES* 775
36.15. Представление *VIEWS* 776

ЧАСТЬ V. ВЗАИМОДЕЙСТВИЕ MYSQL С ЯЗЫКАМИ ПРОГРАММИРОВАНИЯ 779

Глава 37. Взаимодействие MySQL и C/C++ 781

37.1. Взаимодействие с MySQL в Linux 781
37.1.1. Типы данных 786
37.1.2. Функции интерфейса C 791
37.2. Взаимодействие с MySQL в Windows 815

Глава 38. Взаимодействие MySQL и Perl 838

Глава 39. Взаимодействие MySQL и PHP 844

39.1. Функция *mysql_connect()* 844
39.2. Функция *mysql_close()* 846
39.3. Функция *mysql_select_db()* 847
39.4. Функция *mysql_query()* 850
39.5. Функция *mysql_result()* 851
39.6. Функция *mysql_fetch_row()* 852
39.7. Функция *mysql_fetch_assoc()* 853
39.8. Функция *mysql_fetch_array()* 855
39.9. Функция *mysql_fetch_object()* 858
39.10. Функция *mysql_num_rows()* 859
39.11. Система регистрации 862
39.12. Система авторизации 871
39.13. Базовая HTTP-авторизация 873
39.14. Пользователи online 876
39.15. Постраничная навигация 879
39.16. Алфавитная навигация 883
39.17. Сортировка 885
39.18. Двойной выпадающий список 888
39.19. Удаление сразу нескольких позиций 894
39.20. Хранение MP3-файлов в базе данных 897
39.21. Хранение изображений в базе данных 901

Заключение 907

ПРИЛОЖЕНИЯ **909**

Приложение 1. Пространственные расширения MySQL **911**

П1.1. Геометрическая модель OpenGIS 912
П1.1.1. Класс *Geometry* 913
П1.1.2. Класс *Point* 914
П1.1.3. Класс *Curve* 915
П1.1.4. Класс *LineString* 915
П1.1.5. Класс *Surface* 916
П1.1.6. Класс *Polygon* 916
П1.1.7. Класс *GeometryCollection* 917
П1.1.8. Класс *MultiPoint* 917
П1.1.9. Класс *MultiCurve* 917
П1.1.10. Класс *MultiString* 918
П1.1.11. Класс *MultiSurface* 918
П1.1.12. Класс *MultiPolygon* 918
П1.2. Форматы пространственных данных 918
П1.2.1. Объект *Point* 919
П1.2.2. Объект *LineString* 921
П1.2.3. Объект *Polygon* 921
П1.2.4. Объект *MultiPoint* 922
П1.2.5. Объект *MultiLineString* 923
П1.2.6. Объект *MultiPolygon* 924
П1.2.7. Объект *GeometryCollection* 925
П1.3. Работа с геометрическими элементами в MySQL 926
П1.3.1. Функция *PointFromText()* 927
П1.3.2. Функция *PointFromWKB()* 929
П1.3.3. Функция *Point()* 930
П1.3.4. Функция *LineFromText()* 930
П1.3.5. Функция *LineFromWKB()* 931
П1.3.6. Функция *LineString()* 931
П1.3.7. Функция *PolyFromText()* 933
П1.3.8. Функция *PolyFromWKB()* 933
П1.3.9. Функция *Polygon()* 934
П1.3.10. Функция *GeomFromText()* 935
П1.3.11. Функция *GeomFromWKB()* 936
П1.3.12. Функция *MPointFromText()* 936
П1.3.13. Функция *MPointFromWKB()* 937
П1.3.14. Функция *MultiPoint()* 937
П1.3.15. Функция *MLineFromText()* 938
П1.3.16. Функция *MLineFromWKB()* 939
П1.3.17. Функция *MultiLineString()* 939
П1.3.18. Функция *MPolyFromText()* 940
П1.3.19. Функция *MPolyFromWKB()* 941
П1.3.20. Функция *MultiPolygon()* 941

П1.3.21. Функция *GeomCollFromText()* 942
П1.3.22. Функция *GeomCollFromWKB()* 943
П1.3.23. Функция *GeometryCollection()* 943
П1.4. Общие функции геометрических объектов 944
П1.4.1. Функция *Dimension()* 945
П1.4.2. Функция *Envelope()* 945
П1.4.3. Функция *GeometryType()* 948
П1.4.4. Функция *SRID()* 949
П1.5. Функции для работы с объектом *Point* 949
П1.5.1. Функция *X()* 949
П1.5.2. Функция *Y()* 950
П1.6. Функции для работы с объектом *LineString* 951
П1.6.1. Функция *EndPoint()* 951
П1.6.2. Функция *StratPoint()* 951
П1.6.3. Функция *PointN()* 952
П1.6.4. Функция *NumPoints()* 953
П1.6.5. Функция *GLength()* 953
П1.6.6. Функция *IsClosed()* 954
П1.7. Функции для работы с объектом *MultiLineString* 955
П1.7.1. Функция *GLenght()* 955
П1.7.2. Функция *IsClosed()* 956
П1.8. Функции для работы с объектом *Polygon* 957
П1.8.1. Функция *Area()* 957
П1.8.2. Функция *ExteriorRing()* 959
П1.8.3. Функция *InteriorRingN()* 960
П1.8.4. Функция *NumInteriorRings()* 961
П1.9. Функции для работы с объектом *MultiPolygon* 961
П1.9.1. Функция *Area()* 961
П1.10. Функции для работы с объектом *GeometryCollection* 962
П1.10.1. Функция *GeometryN()* 962
П1.10.2. Функция *NumGeometries()* 963
П1.11. Функции для проверки отношения минимальных ограничивающих прямоугольников 964
П1.11.1. Функция *MBRContains()* 964
П1.11.2. Функция *MBRDisjoint()* 964
П1.11.3. Функция *MBREqual()* 966
П1.11.4. Функция *MBRIntersects()* 967
П1.11.5. Функция *MBROverlaps()* 968
П1.11.6. Функция *MBRTouches()* 970
П1.11.7. Функция *MBRWithin()* 970
П1.12. Пространственные индексы 971

Предметный указатель 975

Введение

Для кого и о чем эта книга

Книга посвящена описанию СУБД MySQL и особенностям ее диалекта SQL. Популярность реляционных баз данных неуклонно возрастает последние десятилетия. СУБД MySQL представляет собой СУБД с открытым кодом, разработкой которой занимается корпорация MySQL AB. СУБД MySQL успешно работает под управлением свыше 20 различных операционных систем (Windows, Linux, OS/2, Mac OS X, FreeBSD, Solaris, SunOS, OpenBSD, NetBSD, AIX, DEC UNIX, Tru64 UNIX и т. д.), а также предоставляет интерфейс для взаимодействия с множеством языков программирования: C, C++, Eiffel, Java, Perl, PHP, Python, Ruby и Tcl.

Принимая во внимание тот факт, что это одна из немногих СУБД с открытым кодом, и лицензионное соглашение, по которому она распространяется, не требует денежных отчислений (исключением является случай, когда MySQL входит в состав коммерческого продукта), не удивительно, что она получила широкое распространение в Российской Федерации. В частности MySQL является стандартом де-факто у российских хост-провайдеров.

На сегодняшний день MySQL, наряду с СУБД Oracle, является одной из самых быстрых баз данных. В версии MySQL 5.0 была достигнута функциональность, доступная ранее только в коммерческих базах данных. Так введены долгожданные вложенные запросы, хранимые процедуры, курсоры, обработчики ошибок, триггеры, представления и множество новых внутренних функций.

Книга будет полезна как начинающим разработчикам приложений с применением баз данных, которые хотят разобраться во всех тонкостях работы с MySQL, так и профессионалам, использовавшим в своей работе более ранние версии MySQL (3 и 4) и желающим познакомиться с нововведениями MySQL 5.0. Изложение в книге ведется таким образом, чтобы у читателя, не знакомого с языком запросов SQL, материал книги не вызвал затруднений. Сложность материала возрастает последовательно от первой главы до последней. На протяжении всей книги нововведения версий 4.1, 5.0 и 5.1 указываются явно в тексте, поэтому книга может быть полезна разработчикам, использующим более старые версии СУБД MySQL, но планирующим со временем перейти на новые версии.

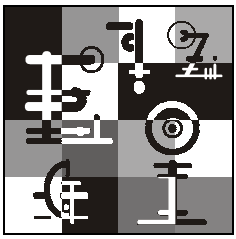
Часть I книги посвящена основам SQL и взаимодействию с сервером MySQL. В этой части ориентация идет в основном на начинающих разработчиков. *Часть II* посвящена более сложным вопросам, таким как внутренние функции, полнотекстовый поиск, регулярные выражения в MySQL, временные таблицы и переменные, вложенные и многотабличные запросы, транзакции и блокировки, учетные записи и администрирование. *Часть III* предоставляет сведения об административных средствах СУБД MySQL, утилитах, параметрах настройки, восстановлении данных и т. д. *Часть IV* полностью посвящена нововведениям 5-й версии СУБД MySQL — рассматриваются хранимые процедуры и функции, обработчики ошибок, курсоры, триггеры, представления и информационная схема. Наконец, в заключительной части *(часть V)* рассматривается взаимодействие СУБД MySQL с прикладными программами, разработанными с помощью языков программирования C/C++, Perl и PHP.

На протяжении всей книги используется учебная база данных электронного магазина, которую можно найти в каталоге base прилагаемого к книге компакт-диска, а также загрузить по адресу **http://www.softtime.ru/sql/base.zip**. Помимо этого, компакт-диск содержит дистрибутивы СУБД MySQL версий 5.0 и 5.1 для операционных систем Windows и Linux. Кроме того, на прилагаемом компакт-диске вы сможете найти графические клиенты для обращения к серверу MySQL.

По всем вопросам, которые могут возникнуть по мере чтения материала книги, вы можете обращаться на форум, расположенный на Web-сайте IT-студии SoftTime, сотрудниками которой являются авторы книги (**http://www.softtime.ru/forum/index.php?id_forum=3**). Авторы присутствуют на форуме каждый день и ответят на возникшие у вас вопросы.

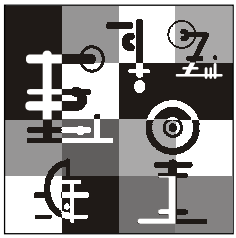
Благодарности

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.



ЧАСТЬ I

ЗНАКОМСТВО С SQL



Глава 1

История развития баз данных. Понятие реляционной базы данных

Компьютерные технологии постоянно эволюционируют. Как следствие, программы становятся все более и более изощренными, а чем сложнее программа, тем больше кода необходимо для ее создания. Исследования показали, что плотность ошибок на 1000 строк кода не зависит от используемой технологии, а зависит лишь от квалификации программиста. Поэтому чем короче программа, тем меньше ошибок она содержит. В связи с этим основные усилия программистов направлены на создание таких языков программирования и технологий, которые позволили бы уменьшить объем программ, а следовательно, и ошибок.

За время существования компьютерной отрасли программисты сменили машинные коды на ассемблер, ассемблер — на языки высокого уровня. Каждый новый виток в технологии программирования обеспечивал одну простую вещь — программист мог создавать более сложные программы, т. к. рутинные операции брали на себя компиляторы и библиотеки.

Однако в компьютерной отрасли сложилась уникальная ситуация: на протяжении 40 лет выполняется закон Мура, каждые 18—24 месяца происходит удвоение производительности микрочипов при неизменном падении цены на них. Это приводит к тому, что компьютеры способны обрабатывать все больший объем данных и выполнять все более сложные программы.

Практически сразу после появления языков высокого уровня их средств стало не хватать для создания ясных и легко сопровождаемых программ, т. к. последние достигли объемов десятков тысяч строк (в одном файле). Ответом на это явилось структурное программирование, позволившее увеличить объем программы до 100 000 строк. Добиться этого удалось за счет особого набора правил и введения функций, позволивших разбить код на более мелкие блоки, оперировать которым много проще, чем огромной монолитной программой. Очень скоро и данного предела оказалось не достаточно, и был предложен объектно-ориентированный подход, который позволил создавать еще более сложные и объемные программы.

Программы — это процессы, которые манипулируют данными. Как же происходила эволюция данных? В далеких 60-х годах прошлого столетия программисты использовали машинные коды и файлы. В настоящий момент ни прикладное, ни даже системное

программное обеспечение в машинных кодах не пишется, однако файлы используются повсеместно. Значит ли это, что эволюция данных не происходила? Ответу на этот вопрос посвящена данная глава.

1.1. История развития СУБД. Реляционные базы данных

Задача длительного хранения и обработки информации появилась практически сразу с созданием первых компьютеров. Для решения этой задачи в конце 60-х годов XX века были разработаны специализированные программы, получившие название *систем управления базами данных* (СУБД). СУБД проделали длительный путь эволюции от системы управления файлами через иерархические и сетевые базы данных к реляционным моделям. В конце 80-х годов XX века доминирующей стала *система управления реляционными базами данных* (СУРБД). С этого времени такие СУБД стали стандартом де-факто. Каждая СУБД имела собственный язык запросов, и для того чтобы унифицировать работу с ними, был разработан *язык структурированных запросов* (Structured Query Language, SQL), который представляет собой язык управления реляционными базами данных. Цель данного раздела — рассмотреть реляционную модель данных, сравнив ее с более ранними СУБД: иерархическими и сетевыми.

> ***Замечание***
>
> Взаимодействие с базой данных происходит при помощи *системы управления базой данных* (СУБД), которая расшифровывает запросы и производит операции с информацией в базе данных. Поэтому более правильно было бы говорить о запросе к СУБД и о взаимодействии приложения с СУБД. Но так как это несколько усложняет восприятие, далее везде мы будем говорить "база данных", подразумевая при этом СУБД.

Существуют следующие разновидности баз данных:

- система управления файлами;
- иерархические;
- сетевые;
- реляционные;
- объектно-ориентированные;
- гибридные.

1.1.1. Иерархические базы данных

Первыми базами данных были иерархические. Иерархическая база данных основана на древовидной структуре хранения информации и в этом смысле напоминает файловую систему компьютера. Наиболее известным и распространенным примером иерархической базы данных является Information Management System (IMS) фирмы IBM, первая версия которой появилась в 1968 году. С точки зрения организации хранения информации иерархическая база данных состоит из упорядоченного набора

деревьев одного типа, т. е. каждая запись в базе данных реализована в виде отношений "предок-потомок". На рис. 1.1 показана схема организации такой иерархической базы данных.

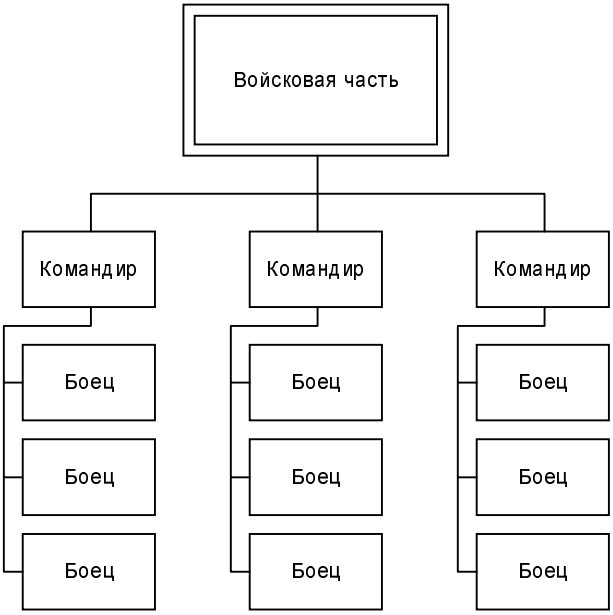


Рис. 1.1. Схема организации иерархической базы данных на примере войсковой части

В этой схеме объекты "Командир" являются потомками для объекта "Войсковая часть", а объекты "Боец" являются потомками для объекта "Командир". Вся база представляет собой иерархическое дерево, используя которое можно ответить на вопросы: сколько бойцов в части, сколько бойцов необходимо перебросить в часть, если она будет укомплектована еще 12 командирами и т. п.

Основной недостаток иерархической структуры базы данных — невозможность реализовать отношения "многие ко многим". Например, если мы создаем каталог книг, то одна книга может относиться сразу к нескольким разделам, в этом случае отношения будут выглядеть так, как это представлено на рис. 1.2.

Иерархические базы данных наиболее пригодны для моделирования структур, являющихся иерархическими по своей природе. Однако они не позволяют организовывать более сложные отношения, например, отнести отдельные книги сразу к нескольким подкаталогам (рис. 1.2), т. к. иерархия подразумевает только одного родителя. Если при моделировании воинской части это условие легко выполнить, поскольку воинские подразделения подразумевают единоначалие, то в остальных сферах деятельности разработчики столкнулись с тем, что иерархические базы данных плохо справляются с моделированием.

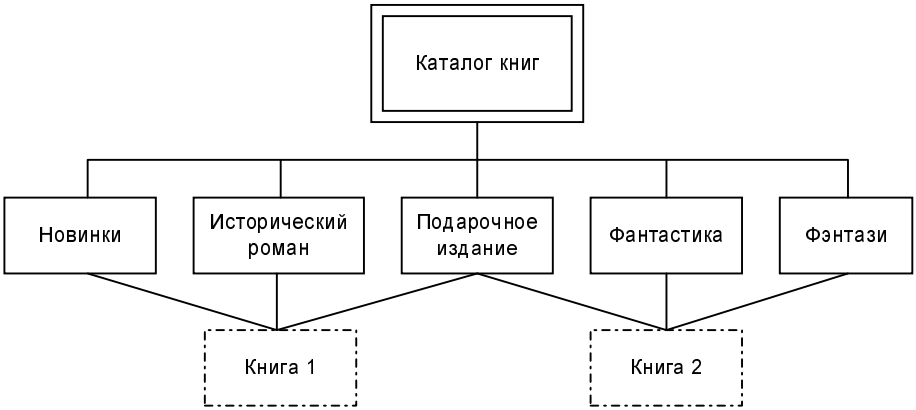


Рис. 1.2. Отношение "многие ко многим"

Примечание

В настоящее время не разрабатываются СУБД, поддерживающие только иерархическую модель организации хранения данных. В лучшем случае проводится совмещение иерархической и сетевой модели (см. далее), которые допускают связывание древовидных структур между собой и установление связей внутри них. Такие СУБД называются *сетевыми даталогическими СУБД*.

1.1.2. Сетевые базы данных

Иерархические базы данных в силу большого количества недостатков просуществовали недолго и были заменены сетевыми базами данных, являющимися, по сути, расширением иерархических. То есть если в иерархических базах данных структура "запись-потомок" должна иметь только одного предка, то в сетевой базе данных потомок может иметь любое количество предков (см. рис. 1.2).

Примечание

Термин "сетевая" в данном случае не подразумевает никакого намека на локальные сети. Этим словом обозначается модель связей в базе данных, когда каждая запись может находиться в отношениях "многие ко многим" с другими записями, что делает графическую модель базы похожей на рыбацкую сеть (рис. 1.3).

Схема сетевой базы данных на примере организации сети железнодорожных путей показана на рис. 1.3. Как видно из этого рисунка, каждая крупная железнодорожная станция может иметь связи (пути сообщения) с несколькими другими станциями (связь "многие ко многим"). Примерно таким же образом выглядит графическая схема любой сетевой базы данных.

Основным недостатком сетевых баз данных является сложность разработки больших приложений.

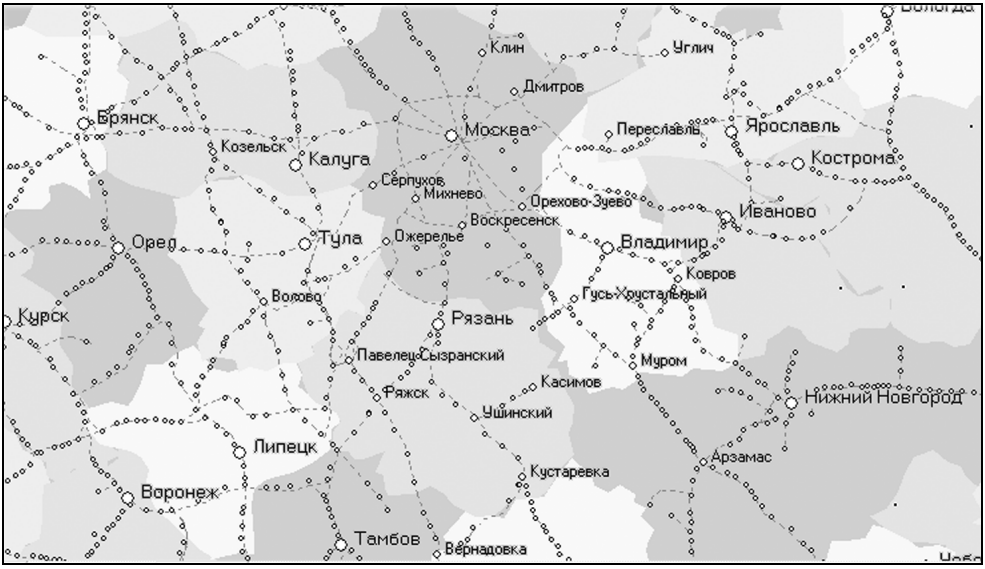


Рис. 1.3. Схема организации сетевой базы данных

1.2. Особенности реляционных баз данных

Настоящий прорыв в развитии теории баз данных произошел тогда, когда возросшая мощность компьютеров позволила реализовать реляционную модель данных. Теория реляционных баз данных была разработана доктором Коддом из компании IBM в 1970 году. Одной из задач реляционной модели была попытка упростить структуру базы данных. В ней отсутствовали явные указатели на предков и потомков, а все данные были представлены в виде простых таблиц, разбитых на строки и столбцы, на пересечении которых расположены данные.

Можно кратко сформулировать особенности реляционной базы данных.

- Данные хранятся в таблицах, состоящих из столбцов и строк.
- На пересечении каждого столбца и строки находится только одно значение.
- У каждого столбца есть свое имя, которое служит его названием, и все значения в одном столбце имеют один тип. Например, в столбце `id_catalogs` все значения имеют целочисленный тип, а в строке `name` — текстовый.
- Столбцы располагаются в определенном порядке, который задается при создании таблицы, в отличие от строк, которые располагаются в произвольном порядке. В таблице может не быть ни одной строчки, но обязательно должен быть хотя бы один столбец.
- Запросы к базе данных возвращают результат в виде таблиц, которые тоже могут выступать как объект запросов.

На рис. 1.4 приведен пример таблицы `catalogs` базы данных электронного магазина изделий компьютерных комплектующих, которые подразделяются на разделы. Каждая строка этой таблицы представляет собой один вид товарных позиций, для описания которой используется поле `id_catalog` — уникальный номер раздела, `name` — название раздела. Столбцы определяют структуру таблицы, а строки — число записей в таблице. Как правило, одна база данных содержит несколько таблиц, которые могут быть как связаны друг с другом, так и независимы друг от друга.

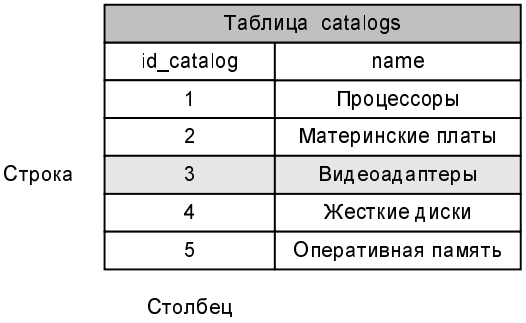


Рис. 1.4. Таблица реляционной базы данных

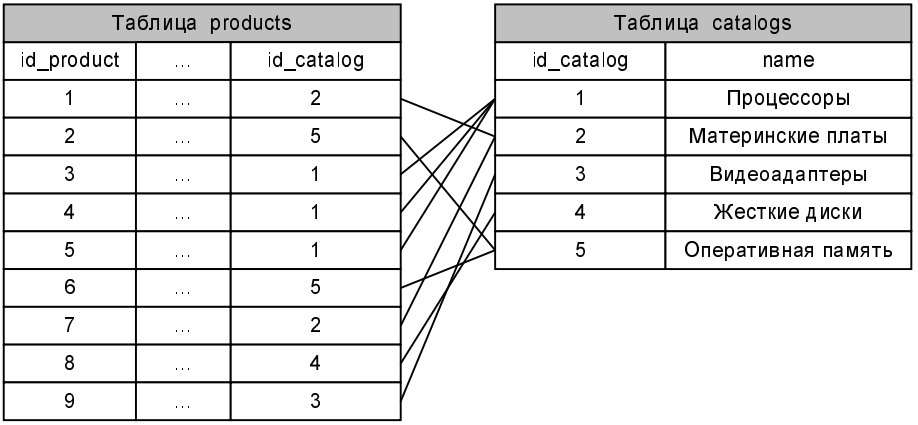


Рис. 1.5. Связанные друг с другом таблицы

На рис. 1.5 приведена структура базы данных, состоящей из двух таблиц, `catalogs` и `products`. Таблица `catalogs` определяет число и названия разделов, а таблица `products` содержит описание товарных позиций. Одной товарной позиции соответствует одна строка таблицы. Последнее поле таблицы `products` содержит значения из поля `id_catalog` таблицы `catalogs`. По этому значению можно однозначно опреде-

лить, из какого раздела товарная позиция. Таким образом, таблицы получаются связанными друг с другом, и хотя эта связь условна, может отсутствовать и в большинстве случаев проявляется только в результате специальных запросов, именно благодаря наличию связей такая форма организации информации получила название *реляционной* (связанной отношениями).

> ***Примечание***
>
> В математике таблица, все строки которой отличаются друг от друга, называется *отношением*, — по-английски relation. Именно этому английскому термину реляционные базы данных и обязаны своим названием.

Конечно, при таком подходе теряется наглядность, которая присутствовала в иерархической и сетевой базах данных, зато значительно возрастает гибкость в организации данных.

> ***Определение***
>
> *Реляционной базой данных* называется такая база данных, в которой все данные организованы в виде таблиц, а все операции над этими данными сводятся к операциям над таблицами.

Сила реляционных баз данных заключается не только в уникальной организации информации в виде таблиц. Запросы к таблицам базы данных сами возвращают таблицы, которые называют результирующими таблицами. Даже если возвращается одно значение, это все равно считается таблицей, состоящей из одного столбца и одной строки. То, что SQL-запрос возвращает таблицу, очень важно. Это означает, что результаты запроса можно записать обратно в базу данных в виде таблицы, а результаты двух или более запросов, которые (т. е. результаты) имеют одинаковую структуру, можно объединить в одну таблицу. И, наконец, это говорит о том, что результаты запроса сами могут стать предметом дальнейших запросов.

1.2.1. Первичные ключи

Строки в реляционной базе данных неупорядочены, т. е. в таблице нет "первой", "последней", "тридцать шестой" и "сорок третьей" строк, а есть просто неупорядоченный набор строк. Возникает вопрос: каким же образом выбирать в таблице конкретную строку? Для этого в правильно спроектированной базе данных для каждой таблицы создается один или несколько столбцов, значения которых во всех строках различны. Такой столбец называется *первичным ключом* таблицы. Первичный ключ обычно сокращенно обозначают как PK (primary key). Никакие из двух записей таблицы не могут иметь одинаковых значений первичного ключа, т. е. благодаря наличию первичных ключей каждая строка таблицы обладает своим уникальным идентификатором. Так на рис. 1.5 в качестве первичного ключа таблицы `products` выступает поле `id_product`, а в качестве первичного ключа таблицы `catalogs` — поле `id_catalog`.

По способу задания первичных ключей различают *логические* (естественные) ключи и *суррогатные* (искусственные).

Для логического задания первичного ключа необходимо выбрать в базе данных то, что естественным образом определяет запись. Примером такого ключа является номер паспорта в базе данных о паспортных данных жителей. К примеру, в таблице базы данных, содержащей паспортные данные, уникальный индекс можно создать для поля "номер паспорта", поскольку каждый такой номер является единственным в своем роде. Однако дата рождения уже не уникальна, поэтому поле "Дата рождения" не может выступать в качестве первичного ключа.

Если подходящих примеров для естественного задания первичного ключа не находится, пользуются суррогатным ключом. Суррогатный ключ представляет собой дополнительное поле в базе данных, предназначенное для обеспечения записей первичным ключом (именно такой подход принят на рис. 1.5).

> ***Совет***
>
> Даже если в базе данных содержится естественный первичный ключ, лучше использовать суррогатные ключи, поскольку их применение позволяет абстрагировать первичный ключ от реальных данных. В этом случае облегчается работа с таблицами, поскольку суррогатные ключи не связаны ни с какими фактическими данными этой таблицы.

Как уже упоминалось, в реляционных базах данных практически всегда разные таблицы логически связаны друг с другом. Одним из предназначений первичных ключей и является однозначная организация такой связи.

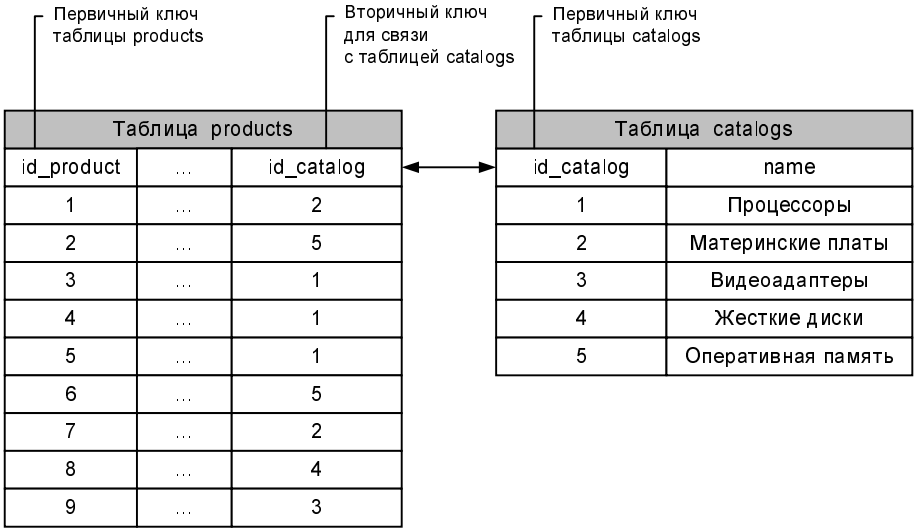


Рис. 1.6. Связь между таблицами `products` и `catalogs`

Например, рассмотрим упоминавшуюся базу данных электронного магазина (рис. 1.6). Каждая из таблиц имеет свой первичный ключ, значение которого уникально в пределах таблицы. Еще раз подчеркнем, что таблица не обязана содержать первичные

ключи, но это очень желательно, если данные связаны друг с другом. Столбец id_catalog таблицы products принимает значения из столбца id_catalog таблицы catalogs. Благодаря такой связи мы можем выстроить иерархию товарных позиций и определить, к какому разделу они относятся. Поле id_catalog таблицы products называют внешним или вторичным ключом. Внешний ключ сокращенно обозначают как FK (foreign key).

1.2.2. Нормализация базы данных

Нормализацией схемы базы данных называется процедура, производимая над базой данных с целью удаления в ней избыточности.

Для того чтобы лучше уяснить приведенное определение нормализации, рассмотрим следующий пример. В электронном магазине, таблицы которого рассматриваются, необходимо учитывать не только товарные позиции, но и покупки. Это может быть полезно для автоматического пополнения запасов на складе, ведения бухгалтерии, отслеживания предпочтений покупателей или для предоставления им скидок. Для учета сделок можно использовать таблицу orders, структура которой представлена на рис. 1.7.

Таблица orders				
id_order	user	ordertime	number	id_product
1	Симдянов И. В.	2005-10-04	1	8
2	Корнеев А. А.	2005-02-10	2	10
3	Иванов В. А.	2005-02-18	4	20
4	Симдянов И. В.	2005-03-10	1	20
5	Симдянов И. В.	2005-03-17	1	20

Рис. 1.7. Пример избыточности в таблице базы данных

Представленная на рис. 1.7 таблица содержит первичный ключ id_order, фамилию и инициалы покупателя user, время покупки ordertime, число приобретенных товаров number и вторичный ключ id_product, по которому в таблице products можно определить приобретенную товарную позицию. Эта таблица избыточна — для каждого заказа вводится имя покупателя, что может приводить к ошибкам и замедлять операции, связанные с выбором заказов каждого из покупателей. Для нормализации таблицы ее обычно разбивают на две, как это показано на рис. 1.8.

Теперь в таблице orders вместо фамилии покупателя используется вторичный ключ id_user для связи с таблицей users, в которую вынесена вся информация о каждом покупателе. Как видно из рис. 1.8, разбиение таблицы orders на две связанные таблицы позволило добавить контактную информацию для каждого из покупателей. В противном случае потребовалось бы вводить дополнительные поля в таблицу orders и дублировать информацию при каждой новой покупке. При этом изменение

контактной информации потребовало бы обновления всех записей таблицы `orders`, соответствующих данному покупателю.

Таблица orders				
id_order	id_user	ordertime	number	id_product
1	1	2005-10-04	1	8
2	2	2005-02-10	2	10
3	3	2005-02-18	4	20
4	1	2005-03-10	1	20
5	1	2005-03-17	1	20

Таблица users			
id_user	name	phone	email
1	Симдянов И. В.	9056666100	
2	Корнеев А. А.	89-78-36	korneev@domen.ru
3	Иванов В. А.	58-98-78	ivanov@mail.ru

Рис. 1.8. Нормализованная база данных

Очевидно, что нормализация несет в себе немало преимуществ: в нормализованной базе данных уменьшается вероятность возникновения ошибок, она занимает меньше места на жестком диске и т. д.

Замечание

Хотя в теории баз данных и говорится о том, что схема базы данных должна быть полностью нормализована, в реальности при работе с полностью нормализованными базами данных необходимо применять весьма сложные SQL-запросы, что приводит к обратному эффекту — замедлению работы базы данных. Поэтому иногда для упрощения запросов даже прибегают к обратной процедуре — *денормализации*.

1.3. СУБД и сети

Развитие компьютерных сетей, а тем более рост популярности Интернета, сыграли серьезную роль в развитии СУБД. Если раньше и приложение, и СУБД находились на одном центральном главном компьютере (мэйнфрейме), то теперь они отделены друг от друга: клиентская часть выполняется на рабочих машинах, а СУБД и собственно сама база данных находятся на сервере. Клиент и сервер взаимодействуют друг с другом посредством специального языка запросов SQL. Рассмотрим основные этапы эволюции СУБД, связанной с развитием компьютерных сетей.

1.3.1. Централизованная архитектура

При централизованной архитектуре и приложение, и СУБД, и сама база данных размещаются на одном центральном мэйнфрейме (от англ. *mainframe* — базовая универсальная вычислительная машина), пользователи (иногда до сотни на один мэйнфрейм) подключались к нему посредством терминалов, как это показано на рис. 1.9. Сам терминал представлял собой клавиатуру, монитор и сетевую карту, посредством которой происходил обмен данных терминала с мэйнфреймом. Роль приложения в данном случае состоит в принятии вводимых данных с пользовательского терминала (клавиатура на рисунке) по сети и передаче их на обработку СУБД с последующей передачей полученного от СУБД ответа на монитор терминала.

Такой подход был выгоден, когда компьютеры стоили миллионы долларов, стоимость монитора по сравнению с самим компьютером являлась бесконечно малой величиной, а предоставление программисту или пользователю машины в единоличное использование хотя бы на час было непозволительной роскошью.

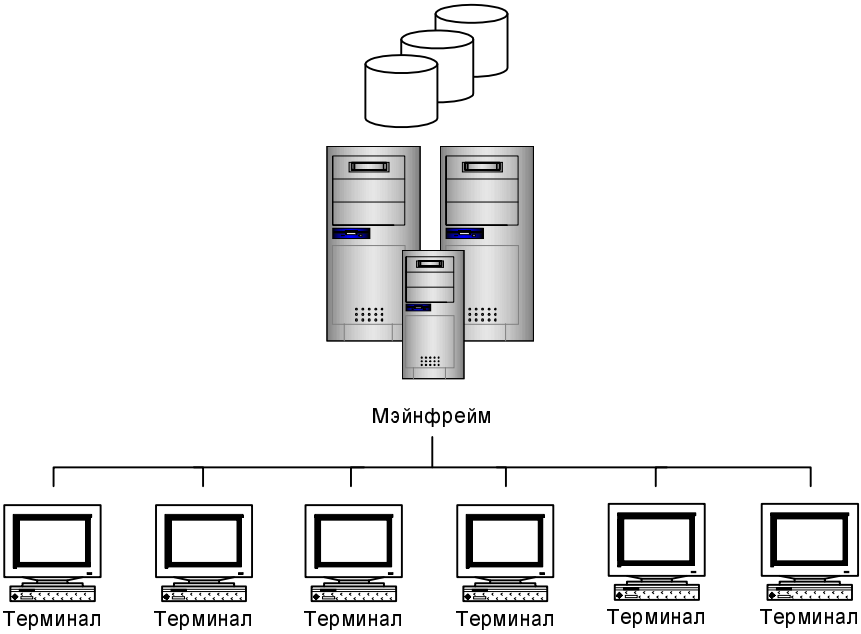


Рис. 1.9. Организация работы СУБД в централизованной архитектуре

Подчеркнем еще раз: основной признак централизованной архитектуры в том, что и приложение, и СУБД работают на одном компьютере. На этом же компьютере находятся и сами данные (база данных).

1.3.2. Архитектура "клиент-сервер"

В клиент-серверной архитектуре персональные компьютеры (клиенты) объединены в локальную сеть, в этой же сети находится и сервер баз данных, на котором содержатся общие для всех клиентов базы данные и СУБД. Клиентские машины не являются полноценными компьютерами в том смысле, в котором мы привыкли понимать, т. е. обладающие собственным процессором, жестким диском, операционной системой (терминал ничем этим не обладал — все вычисления производились на мэйнфрейме и передавались затем по сети). Таким образом, клиенты получили возможность сами выполнять многочисленные приложения. Отныне постраничная навигация, представление результатов, их распечатывание и прочие рутинные возможности легли на клиентские машины. Однако сама СУБД и базы данных полностью располагались на сервере, который по-прежнему превосходил клиентские машины по вычислительным возможностям, но мог значительно уступать по производительности мэйнфремам. Последнее связано с тем, что теперь не было необходимости производить вычисления для сотни терминалов, и вычислительные возможности сервера можно было полностью сосредоточить на обслуживании СУБД.

Переход к клиент-серверным технологиям связан с компьютерной революцией, которая привела к значительному удешевлению персональных компьютеров, сравняв их по стоимости с терминалами. Схема организации архитектуры "клиент-сервер" изображена на рис. 1.10.

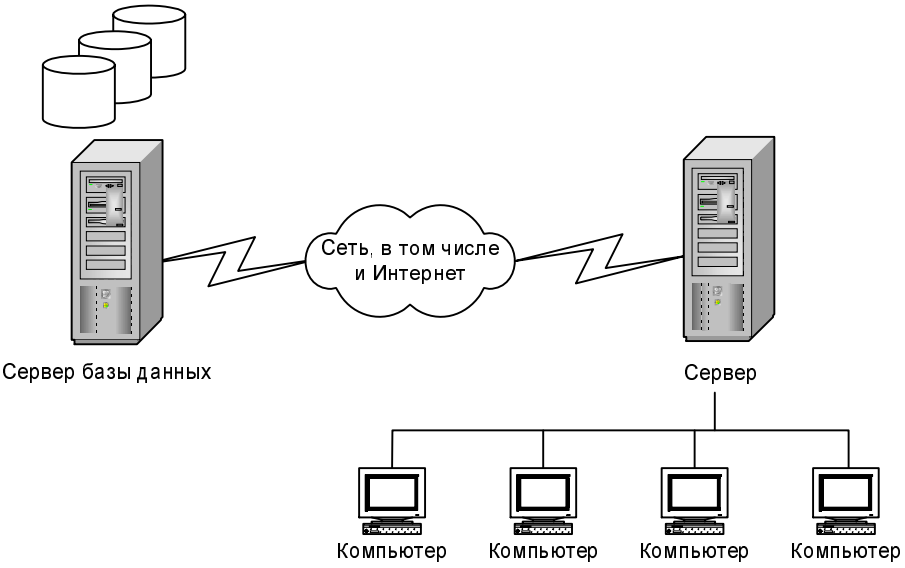


Рис. 1.10. Место СУБД в архитектуре "клиент-сервер"

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.

В связи с тем, что очень часто возникает путаница, что называть клиентом, а что сервером, остановимся на этом вопросе подробнее. Попросту говоря, клиенты — это те, кто обращается (посылают запросы), а сервер — это тот, кто отвечает на эти запросы. Сервер постоянно прослушивает, не поступили ли запросы от клиентов, и если поступили — отвечает на эти запросы. Для прослушивания предназначены один или несколько портов, к которым и обращаются клиентские приложения. Жизненным примером такой организации может служить система "продавец — очередь из покупателей". Продавец за прилавком в данном случае является сервером, обслуживающим покупателей-клиентов. Точно так же как и компьютер-сервер, продавец-сервер, для того чтобы обслужить покупателя-клиента, должен получить от него запрос, к примеру: "Взвесьте мне килограмм яблок". Нужно также понимать, что сервер может одновременно и сам быть клиентом в том случае, если он тоже обращается к другому серверу с каким-либо запросом. Тогда наш первоначальный сервер по отношению к этому "другому серверу" уже будет клиентом, оставаясь сервером для тех клиентов, которых он обслуживает. Продолжая аналогию с продавцами и покупателями, продавец-сервер тоже может стать клиентом, например, когда сам пойдет в магазин и встанет в очередь за каким-то товаром, т. е. станет покупателем-клиентом.

Таким образом, в архитектуре "клиент-сервер" функции работы с пользователем, такие как обработка ввода и отображение данных, выполняются на персональном компьютере (клиенте), а функции работы с данными (выполнение запросов, дисковый ввод/вывод) осуществляются сервером баз данных.

1.3.3. Трехуровневая архитектура Интернета

С развитием Интернета архитектура сетевого управления базами данных получила дальнейшее развитие. Ранние сервисы Интернета обеспечивали доступ к базам данных напрямую. Через некоторое время появилась среда Web, которая предоставила пользователям Интернета дружественный интерфейс. За формирование этого интерфейса несет ответственность Web-сервер, т. е. на пути между пользователем-клиентом и базой данных появился еще один посредник, который стал выполнять функции клиентской программы. Такой подход позволяет многочисленным пользователям Интернета иметь единственную программу — Web-браузер — для работы с различными базами данных, будь это интернет-магазин или форум, не прибегая к услугам специфических программ-клиентов. Так как браузер зачастую входит в состав операционной системы и стандартизирован, разработчикам можно не беспокоиться о его установке на клиентских машинах (число которых может превышать десятки тысяч) и об обновлениях. Достаточно изменить программный код на Web-сервере для того, чтобы изменения коснулись сразу всех клиентов.

Замечание

Клиенты-браузеры, в терминологии баз данных, часто называют "тонкими клиентами". В противовес "толстым клиентам", которые ориентированы на конкретную базу данных, тонкие клиенты способны лишь отображать данные в строго определенном формате и отправлять серверу лишь простейшие запросы. ICQ, WebMoney представляют примеры "толстых клиентов", а Internet Explorer, Opera, WAP-браузер сотового телефона являются "тонкими клиентами".

Схема работы трехуровневой архитектуры следующая. Допустим, клиенты какой-либо компании, находясь за тысячи километров от нее, желают ознакомиться со списком товаров, доступных на данный момент. В этом случае они используют браузер для посещения сайта этой компании. Страницу со списком товаров при этом формирует специальное программное обеспечение, выполняющееся на Web-сервере компании. Для получения нужной информации этот скрипт обращается (посылает SQL-запросы) к СУБД, находящейся на сервере баз данных. Иллюстративно такая схема взаимодействия изображена на рис. 1.11.

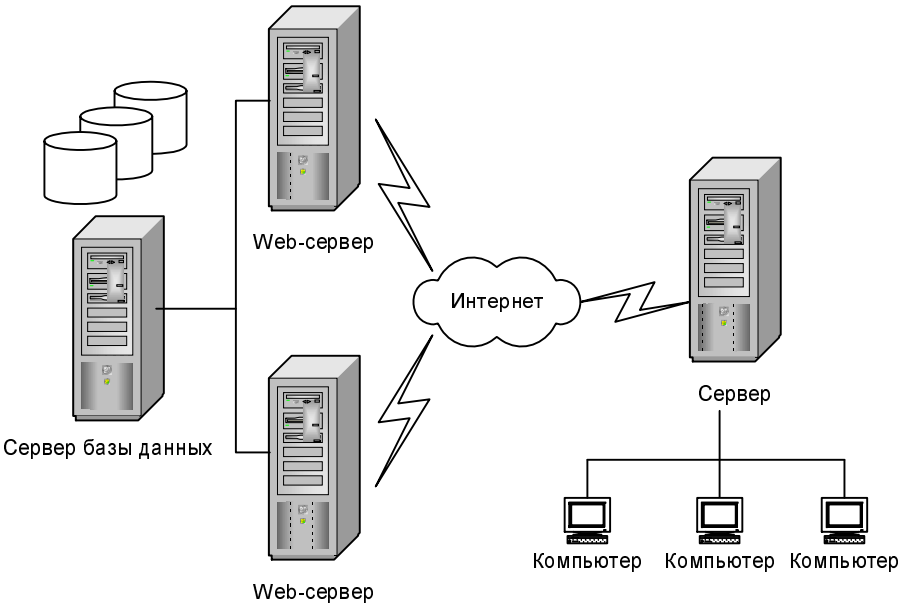


Рис. 1.11. Роль СУБД в трехуровневой архитектуре Интернета

Трехуровневая модель позволяет отделить клиентское программное обеспечение от серверной части, а на серверной стороне отделить Web-сервер от сервера базы данных. Последнее является достаточно актуальной проблемой. Дело в том, что нагрузка на Web-сайты постоянно возрастает по мере развития Интернета и подключения новых пользователей. Одним из способов борьбы с этим является увеличение производительности серверов. Можно увеличивать мощность сервера, например, заменив

двухпроцессорный сервер шестью- или десятипроцессорным, а можно поставить рядом два двухпроцессорных сервера, как это изображено на рис. 1.11. Последний вариант значительно дешевле — чем больше процессоров необходимо согласовать, тем дороже будет решение. Такой вариант более надежен — если сломается один сервер, второй продолжит работу. Наконец, использование нескольких серверов более масштабируемо: если мы заменяем двухпроцессорный сервер шестьюпроцессорным, на руках у нас останется старый сервер, который будет не задействован.

Замечание

Стоимость среднего двухпроцессорного сервера составляет порядка 3000 долларов.

Однако если Web-сервер и сервер базы данных находятся на одном хосте, их распределение будет невозможным, т. к. клиенты должны получать доступ к согласованной базе данных и видеть изменения, которые в нее вносятся другими клиентами. Ввод второго сервера базы данных в этом случае приведет к двум никак не согласованным базам данных.

Таким образом, в трехуровневой архитектуре Интернета интерфейсом пользователя является Web-браузер. Браузер взаимодействует с Web-сервером, посылая ему запросы на отображение той или иной Web-страницы. Уровень Web-сервера — прикладной уровень. Web-приложение, выполняющееся на Web-сервере, формирует SQL-запрос к СУБД, которая в свою очередь возвращает необходимые данные из базы данных. СУБД и база данных при этом размещены на сервере баз данных и представляют собой третий, информационный уровень трехуровневой архитектуры Интернета.

1.3.4. Кластерная модель

В трехуровневой модели затрагивалась проблема масштабирования — несколько серверов, работающих над одной и той же задачей, функционируют надежнее и обходятся дешевле, чем один сервер высокой производительности. Проблема отсутствия масштабируемости серверов баз данных стоит очень остро, т. к. зачастую выбор приходится останавливать на шести-, десятипроцессорных серверах, цены на которые остаются достаточно высокими. Кроме того, выход из строя такого сервера может парализовать работу компании.

Поэтому в последнее время все чаще звучат призывы к переходу к кластерной модели вычислений. Кластеры часто называют дешевыми супер-ЭВМ. Суть заключается в том, что ряд маломощных машин (как правило, персональных компьютеров), объединяют в локальную сеть. Специальное программное обеспечение распределяет вычисления между отдельными хостами сети. Выход из строя одного их хостов никак не отражается на работе всей сети, а сам кластер легко расширяется за счет ввода дополнительных машин.

Пока такой вид вычислений не получил широкого распространения, однако СУБД MySQL уже сейчас позволяет организовать кластер на базе нескольких серверов.

1.4. Как работают базы данных и что такое SQL?

Рассмотрим упрощенную схему работы пользователя с базой данных, показанную на рис. 1.12. Согласно этой схеме, в системе имеется база данных, в которой хранится некоторая информация, допустим, о работниках. По сути, база данных — это те же файлы, в которых хранится информация. Сами по себе базы данных не представляли бы никакого интереса, если бы не было систем управления базами данных, сокращенно — СУБД. СУБД — это программный комплекс, который управляет базой данных, т. е. берет на себя все низкоуровневые операции по работе с файлами, благодаря чему программист при работе с базой данных может оперировать лишь логическими конструкциями при помощи языка программирования, не прибегая к низкоуровневым операциям.

Примечание

База данных — это просто файловое хранилище информации, и не более. А программные продукты типа MySQL, Oracle, Dbase, Informix, PostgreSQL и другие — это системы управления базами данных (СУБД). Базы данных везде одинаковы — это файлы с записанной в них информацией. Все ранее приведенные программные продукты отличаются друг от друга именно способом организации работы с файловой системой. Однако для краткости эти СУБД часто называют просто базами данных. Следует учитывать, что когда мы будем говорить "база данных", речь пойдет именно о СУБД.

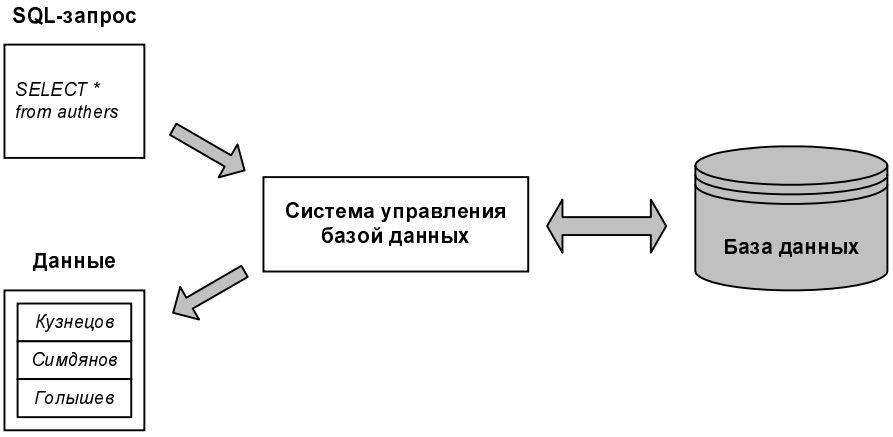


Рис. 1.12. Простейшая схема организации работы с базой данных

Язык программирования, с помощью которого пользователь общается с СУБД (или, как говорят, "осуществляет запросы к базе данных"), называется SQL (Structured Query Language, язык структурированных запросов).

Таким образом, для того чтобы получить какую-либо информацию из базы данных, необходимо направить базе данных запрос, созданный с использованием SQL, результатом выполнения которого будет результирующая таблица (рис. 1.12).

Несмотря на то, что SQL называется "языком запросов", в настоящее время этот язык представляет собой нечто большее, чем просто инструмент для создания запросов. С помощью SQL осуществляется реализация всех возможностей, которые представляются пользователям разработчиками СУБД, а именно:

- выборка данных (извлечение из базы данных содержащейся в ней информации);
- организация данных (определение структуры базы данных и установление отношений между ее элементами);
- обработка данных (добавление/изменение/удаление);
- управление доступом (ограничение возможностей ряда пользователей на доступ к некоторым категориям данных, защита данных от несанкционированного доступа);
- обеспечение целостности данных (защита базы данных от разрушения);
- управление состоянием СУБД.

SQL не является специализированным языком программирования, т. е. в отличие от языков высокого уровня (C++, Pascal и т. д.) с его помощью невозможно создать автономную программу. Все запросы выполняются либо в специализированных программах, либо из прикладных программ при помощи специальных библиотек.

Несмотря на то, что язык запросов SQL строго стандартизирован, существует множество его диалектов, по сути, каждая база данных реализует собственный диалект со своими особенностями и ключевыми словами, недоступными в других базах данных. Такая ситуация связана с тем, что стандарты SQL появились достаточно поздно, в то время как компании-поставщики баз данных существуют давно и обслуживают большое число клиентов, для которых требуется обеспечить обратную совместимость со старыми версиями программного обеспечения. Кроме того, рынок реляционных баз данных оперирует сотнями миллиардов долларов в год, все компании находятся в жесткой конкуренции и постоянно совершенствуют свои продукты. Поэтому, когда дело доходит до принятия стандартов, базы данных уже имеют реализацию той или иной особенности, и комиссии по стандартам приходится в условиях жесткого давления выбирать в качестве стандарта решение одной из конкурирующих фирм. Так, хранимые процедуры, впервые появившиеся в MySQL 5 и чуть ранее в стандарте SQL, до недавнего времени были расширением SQL в языке запросов Transact-SQL базы данных MS SQL.

1.5. Версии MySQL

СУБД MySQL является развитием СУБД mSQL, которую разработчики шведской компании MySQL AB взяли за основу при создании собственной базы данных. Далее перечислены достоинства СУБД MySQL.

- Скорость выполнения запросов. Наряду с Oracle, MySQL считается одной из самых быстрых СУБД в мире.

- СУБД MySQL разработана с использованием языков C/C++ и оттестирована более чем на 23 платформах, среди которых Windows, Linux, FreeBSD, HP-UX, Mac OS X, OS/2, Solaris и др.
- Открытый код, который доступен для просмотра и модернизации всем желающим. Лицензия GPL (General Public License, общедоступная лицензия) позволяет постоянно улучшать программный продукт и быстро находить и устранять уязвимые места. Особенностью лицензии GPL является тот факт, что любой код, скомпилированный с GPL-кодом, попадает под GPL-лицензию, т. е. может свободно распространяться, и условием его распространения является предоставление исходных кодов. MySQL AB придерживается двойного лицензирования, позволяя пользователям бесплатно использовать СУБД MySQL под GPL-лицензией, а также предоставляет MySQL под коммерческой лицензией тем, кому необходимо использовать его в коммерческих целях, например, в составе программы.
- Высокое качество СУБД MySQL. Ни для кого не является секретом постоянно возрастающая сложность программных продуктов, что влечет за собой появление большого числа ошибок в конечных выпусках программного обеспечения. СУБД MySQL приятно удивит пользователей устойчивой работой.
- СУБД MySQL поддерживает API (Application Programming Interface, программный интерфейс приложения) для C, C++, Eiffel, Java, Perl, PHP, Python, Ruby и Tcl. MySQL можно успешно применять как для построения Web-страниц с использованием Perl, PHP и Java, так и для работы прикладной программы, созданной с использованием Builder C++ или платформы .NET.
- Наличие встроенного сервера. СУБД MySQL может быть использована как с внешним сервером, поддерживающим соединение с локальной машиной и с удаленным хостом, так и в качестве встроенного сервера. Достаточно скомпилировать программу с библиотекой встроенного сервера, и приложение будет содержать в себе полноценную СУБД MySQL с возможностью создания баз данных, таблиц и осуществления запросов к ним.
- Широкий выбор типов таблиц, в том числе и сторонних разработчиков, что позволяет реализовать оптимальную для решаемой задачи производительность и функциональность. Кого-то привлечет наиболее быстрый тип таблиц MyISAM, а кому-то требуется поддержка объемов информации до 1 Тбайт с выполнением транзакций на уровне строк (таблицы типа InnoDB).
- Локализация в MySQL выполнена корректно. У пользователя не возникнет проблем из-за того, что в конце проекта обнаружится невозможность поддержки базой данных сортировки русских строк.
- MySQL, начиная с версии 5.0, практически полностью удовлетворяет стандарту SQL и, следовательно, совместима с другими базами данных.

Все эти особенности сделали MySQL стандартной базой данных, используемой на серверах хост-провайдеров и в качестве встроенной базы данных в прикладных программах.

На официальном сайте MySQL **http://www.mysql.com** поддерживается несколько версий СУБД MySQL, обычно три версии. На момент написания данной книги это были следующие версии:

- MySQL 4.1 — устаревшая версия СУБД;
- MySQL 5.0 — стабильная (текущая) версия СУБД, рекомендуемая для загрузки;
- MySQL 5.1 — версия, находящаяся в разработке.

> ***Замечание***
>
> На компакт-диске, поставляемом вместе с данной книгой, можно найти дистрибутивы всех трех версий СУБД MySQL для операционных систем Windows и Linux.

1.5.1. Что нового в MySQL 4.1?

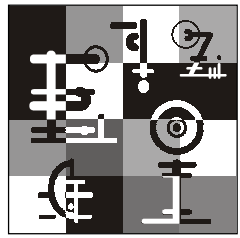
Версия MySQL 4.1 стала стабильной в начале 2005 года и по сравнению с предыдущими версиями характеризуется приведенными далее нововведениями:

- полная поддержка вложенных запросов (данная тема рассматривается в *главе 24*);
- увеличена производительность базы данных;
- введен целый набор дополнительных функций;
- изменен формат типа `TIMESTAMP`, а также введены дополнительные ключевые слова, которые позволяют управлять автоматическим обновлением времени при использовании операторов `INSERT` и `UPDATE`;
- введена новая система поддержки кодировок, теперь можно назначить кодировку по умолчанию как таблице, так и отдельному столбцу в таблице;
- введены новые типы таблиц, такие как EXAMPLE, NDB Cluster, ARCHIVE, CSV, которые подробно рассматриваются в *главе 11*.

1.5.2. Что нового в MySQL 5.0?

Версия MySQL 5.0 является поворотной в линейке СУБД MySQL и содержит значимые нововведения:

- представления *(см. главу 35)*;
- хранимые процедуры и функции *(см. главу 33)*;
- курсоры *(см. главу 33)*;
- обработчики ошибок *(см. главу 33)*;
- триггеры *(см. главу 34)*;
- информационная схема *(см. главу 36)*;
- новый тип таблиц FEDERATED *(см. главу 11)*;
- множество новых функций *(см. часть II)*.



Глава 2

Установка MySQL 5

В данной главе описывается установка сервера MySQL на платформы ОС Windows и Linux.

2.1. Получение дистрибутива

Загрузить дистрибутив СУБД MySQL 5 можно с официального сайта компании MySQL AB по адресу **http://www.mysql.com/**. В разделе **downloads** (**http://dev.mysql.com/downloads/**) дистрибутив доступен в нескольких версиях: стабильной, устаревшей и будущей версии, находящейся в стадии альфа-, бета-, гамма- или дельта-тестирования.

> ***Замечание***
>
> На компакт-диске, прилагаемом к данной книге, можно найти дистрибутивы MySQL версий 5.0 и 5.1 в бинарном виде (готовые к установке) — в каталоге binaries, а также в виде исходных кодов — в каталоге source для ОС Windows и Linux.

Для загрузки дистрибутива следует перейти по ссылке MySQL 5.0 (**http://dev.mysql.com/downloads/mysql/5.0.html**). На открывшейся странице будет представлен список дистрибутивов, скомпилированных под разные операционные системы. Для Linux имеются дистрибутивы в архивах (.tar.gz) и в rpm-пакетах (.rpm), откомпилированные под 32- и 64-битную архитектуры. Для Windows представлены три вида дистрибутивов:

- 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 Windows (x86), поскольку несмотря на то, что его объем превышает все остальные дистрибутивы из Windows-серии, он наиболее удобен в работе. Дальнейшее изложение материала будет основано именно на этом дистрибутиве.

Замечание

Полное справочное руководство на русском языке можно найти на официальном сайте MySQL по адресу **http://dev.mysql.com/doc/mysql/ru/index.html**. К сожалению, оно является устаревшим и описывает MySQL вплоть до версии 4.0. Полное справочное руководство на английском языке можно загрузить в различных форматах со страницы **http://dev.mysql.com/doc/** из раздела **MySQL Reference Manual**.

2.2. Установка на платформу Windows

При работе в Windows NT/2000/XP/Server 2003 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. 2.1.



Рис. 2.1. Стартовое окно автоматического установщика

Для продолжения установки следует нажать кнопку **Next >**, после чего откроется окно, показанное на рис. 2.2, в котором предлагается выбрать различные способы установки:

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

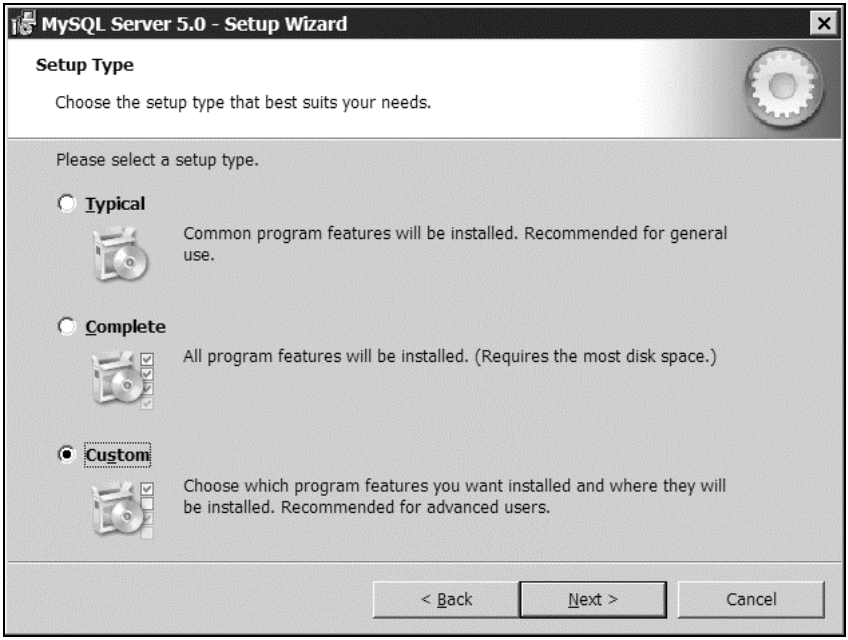


Рис. 2.2. Выбор режима установки MySQL

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. 2.3) 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.

После того как все компоненты выбраны, в этом же диалоговом окне (см. рис. 2.3) 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. 2.4).

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.

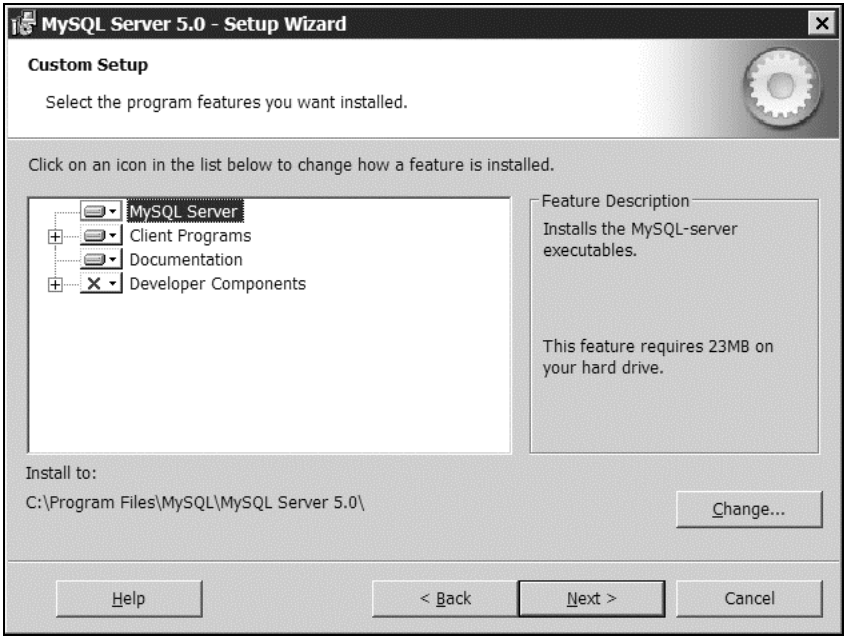


Рис. 2.3. Выбор компонентов для установки

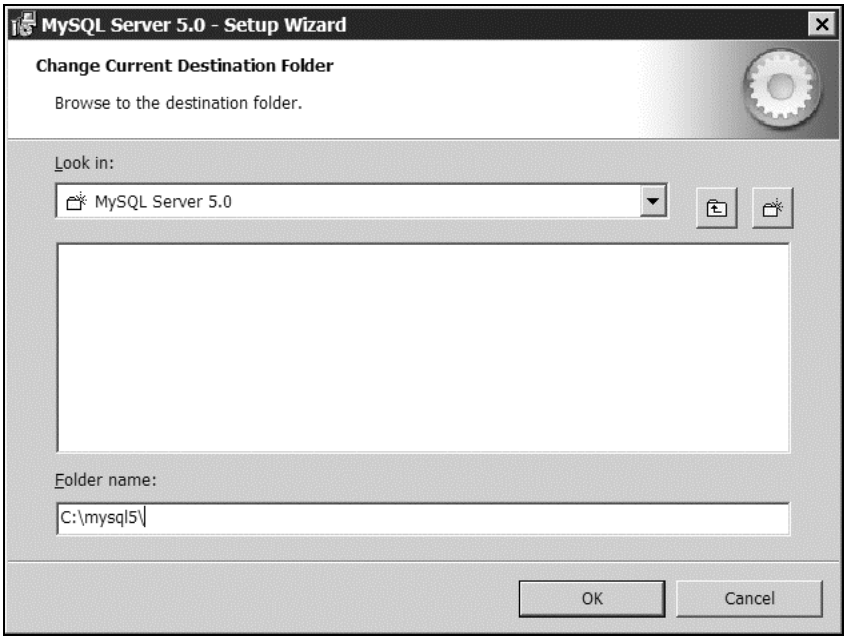


Рис. 2.4. Выбор каталога, в который будет установлен пакет MySQL

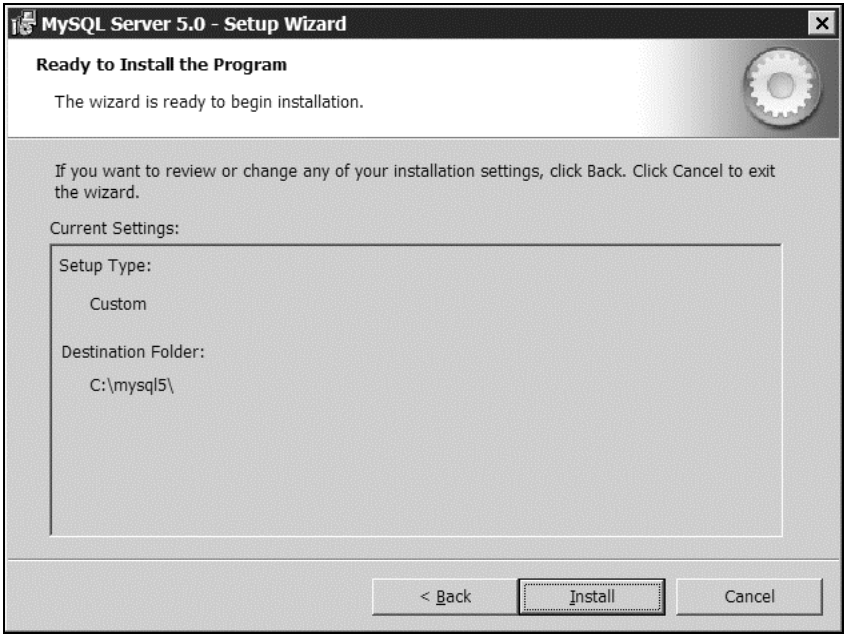


Рис. 2.5. Завершающее окно перед процессом инсталляции

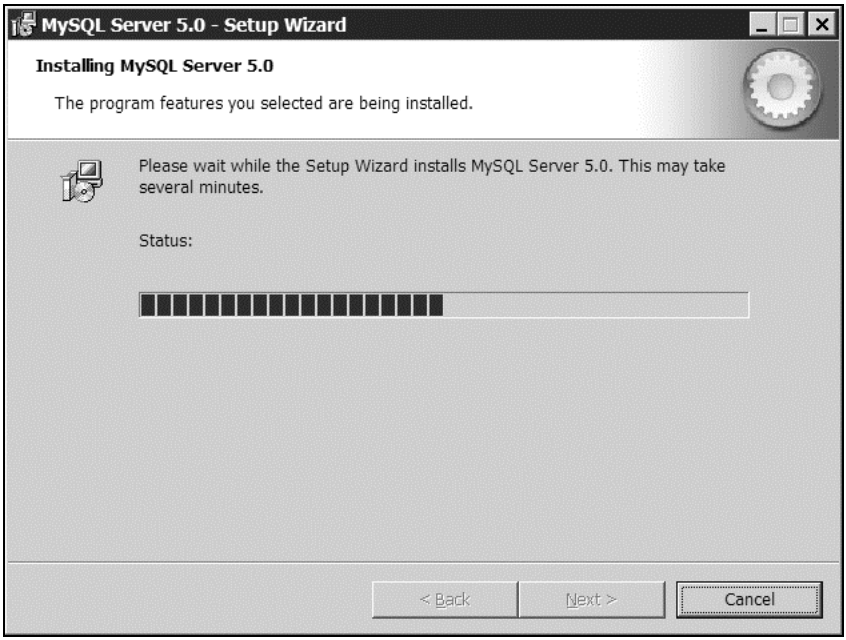


Рис. 2.6. Копирование файлов



Рис. 2.7. Регистрация на сайте **http://www.mysql.com**

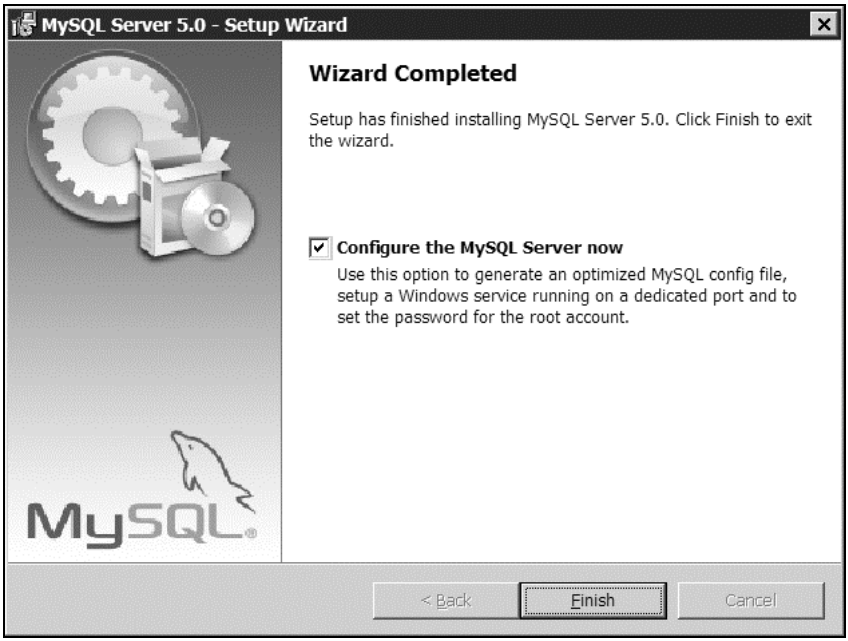


Рис. 2.8. Завершение основного этапа установки MySQL

Поэтому если вы производите обновление MySQL, помимо остановки сервера, следует удалить каталог системной базы данных `mysql C:\mysql\data\mysql`. При первой инсталляции проблем, связанных с несовместимостью, возникать не должно.

После завершения предынсталляционной настройки выводится окно, представленное на рис. 2.5. Нажатие кнопки **Install** начинает процесс копирования фалов, представленный на рис. 2.6.

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. 2.7). Следующее окно (рис. 2.8) сообщает о завершении основного этапа установки MySQL.

Если в данном окне не отключать возможность немедленной настройки сервера, то после нажатия кнопки **Finish** запускается постинсталляционная настройка MySQL 5.

> ***Замечание***
>
> 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. 2.9. После нажатия кнопки **Next >** открывается окно, представленное на рис. 2.10, в котором предлагается выбрать режим настройки:

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

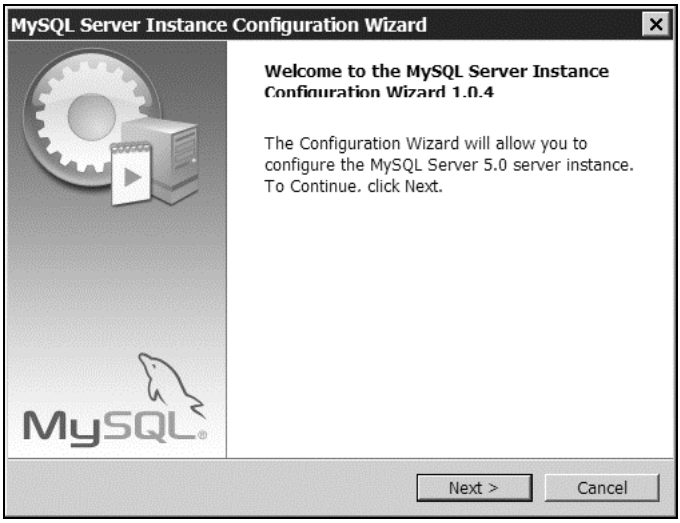


Рис. 2.9. Настройка MySQL при помощи мастера MySQL Server Instance Configuration Wizard

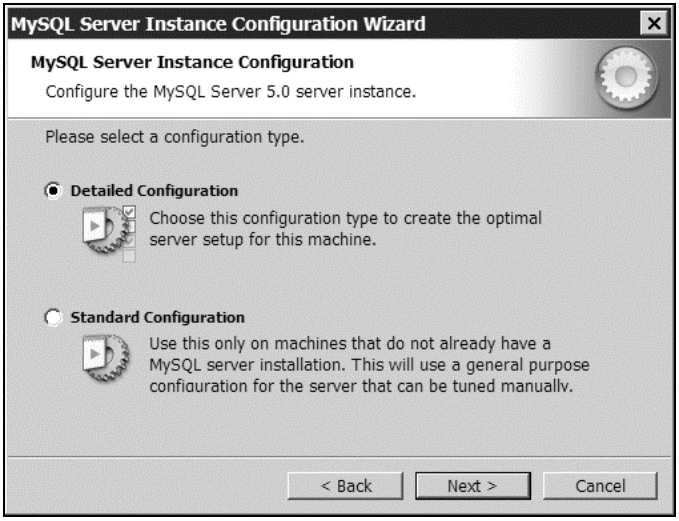


Рис. 2.10. Выбор режима настройки

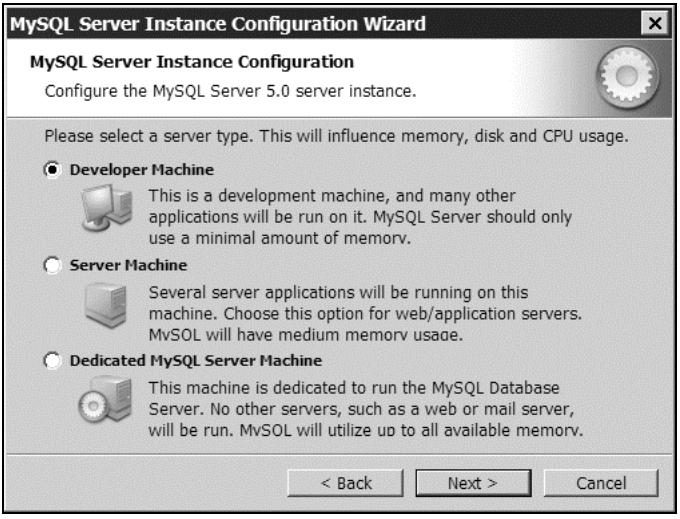


Рис. 2.11. Настройка производительности сервера MySQL

Для более гибкой настройки системы следует выбрать первый пункт (**Detailed Configuration**). После нажатия кнопки **Next >** открывается окно настройки производительности MySQL (рис. 2.11) 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 серверов, 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.

Окно, представленное на рис. 2.12, 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 всегда можно отредактировать вручную, поэтому тип таблиц по умолчанию также несложно откорректировать позже.

В окне, показанном на рис. 2.12, предлагается выбор, по сути, между двумя типами таблиц: MyISAM и InnoDB, которые более подробно обсуждаются в *главе 11*. Главное различие заключается в том, что под каждую таблицу MyISAM создается отдельный файл, который хранится в соответствующем базе данных каталоге, а все таблицы InnoDB хранятся в едином табличном пространстве, под которое выделяется один файл. Кроме этого, скорость работы с MyISAM в несколько раз превышает скорость работы с таблицами типа InnoDB. Однако таблицы InnoDB поддерживают транзакции на уровне строк и более надежны. При первом знакомстве рекомендуется выбрать более простые в обслуживании и настройке таблицы типа MyISAM.

Выбор диска для хранения этого файла и пути, согласно которому MySQL Server будет установлен, осуществляется в следующем окне, представленном на рис. 2.13.

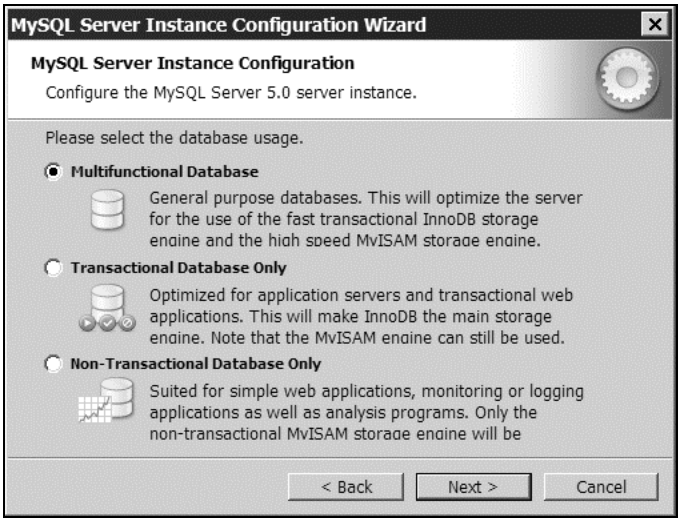


Рис. 2.12. Выбор типа таблиц по умолчанию

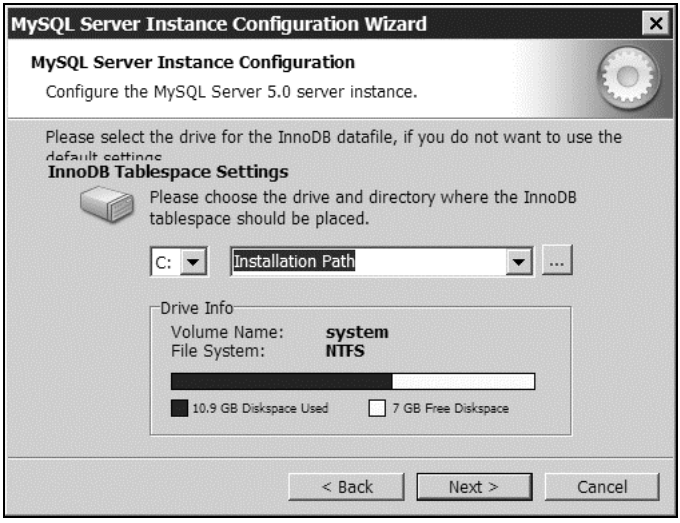


Рис. 2.13. Выбор пути для хранения файла под таблицы InnoDB

Следующее окно (рис. 2.14) предлагает выбрать 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.

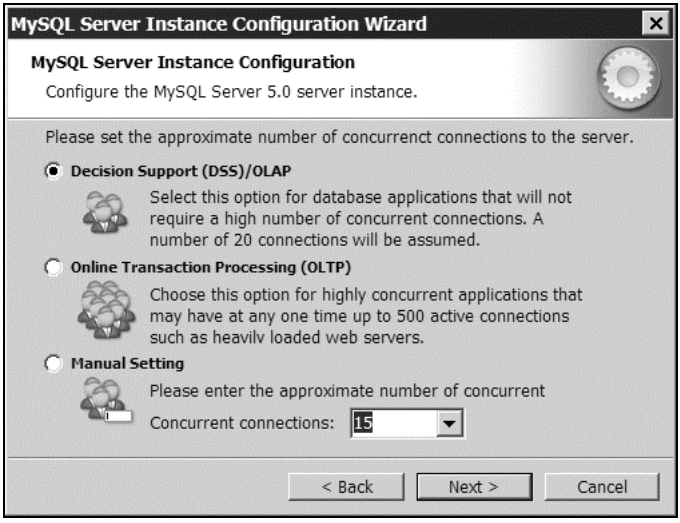


Рис. 2.14. Выбор максимального числа активных соединений с сервером

В следующем окне (рис. 2.15) 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.

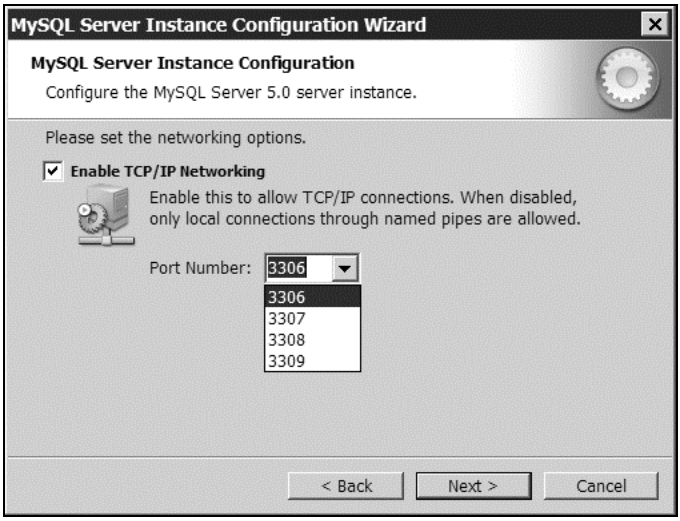


Рис. 2.15. Выбор порта, по которому сервер MySQL будет слушать клиентские запросы

Замечание

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. 2.16). Необходимо выбрать третий пункт (ручной выбор кодировки) и в раскрывающемся списке выбрать пункт **cp1251**, соответствующий русской Windows-кодировке.

При работе в среде Windows NT/2000/XP/Server 2003 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. 2.17, предназначено для настройки этого сервиса. Установка флажка **Install As Windows Service** позволяет установить сервис с именем, которое следует выбрать в раскрывающемся 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 потребуется 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.

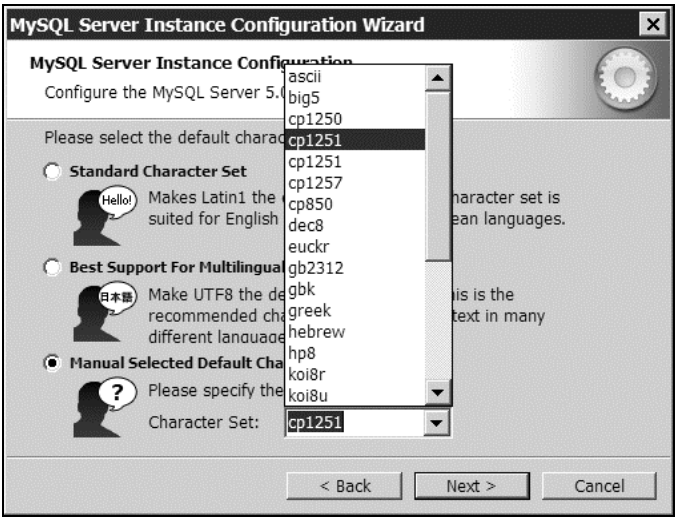


Рис. 2.16. Выбор кодировки на MySQL-сервере по умолчанию

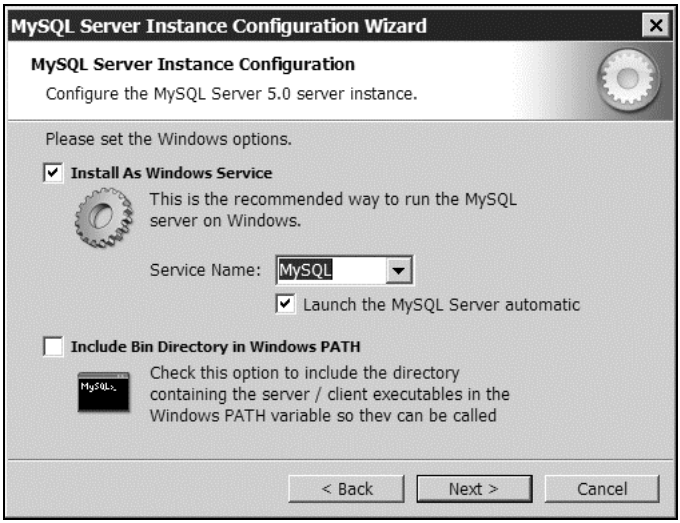


Рис. 2.17. Настройка сервиса для автоматического запуска MySQL-сервера

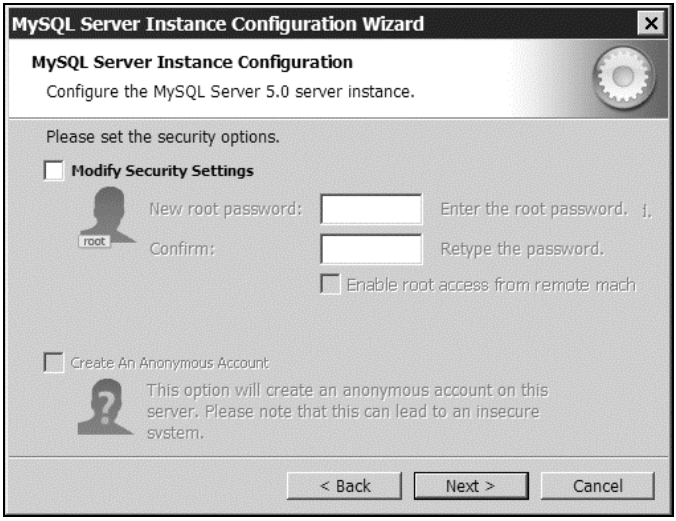


Рис. 2.18. Настройка учетных записей

В следующем окне (рис. 2.18) 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 *главах 3* и *27*. 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 в окне утилиты настройки MySQL-сервера **MySQL Server Instance Configuration Wizard** представлена на рис. 2.19.

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 установкой 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.

После того как установка и конфигурирование MySQL завершены, необходимо убедиться в работоспособности сервера MySQL. Для этого следует запустить режим командной строки, 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 2.20. На предложение ввести пароль нажмите клавишу <Enter>. После того как появилось приглашение `mysql>`, введите команду

```
SELECT VERSION();
```

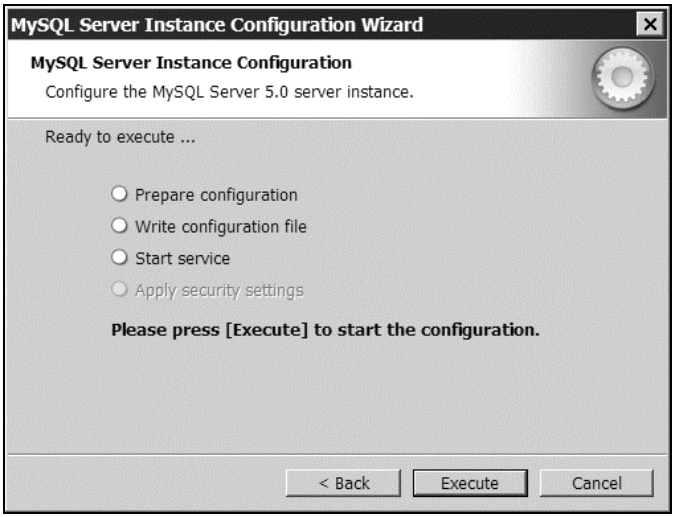


Рис. 2.19. Последняя страница в окне утилиты MySQL Server Instance Configuration Wizard

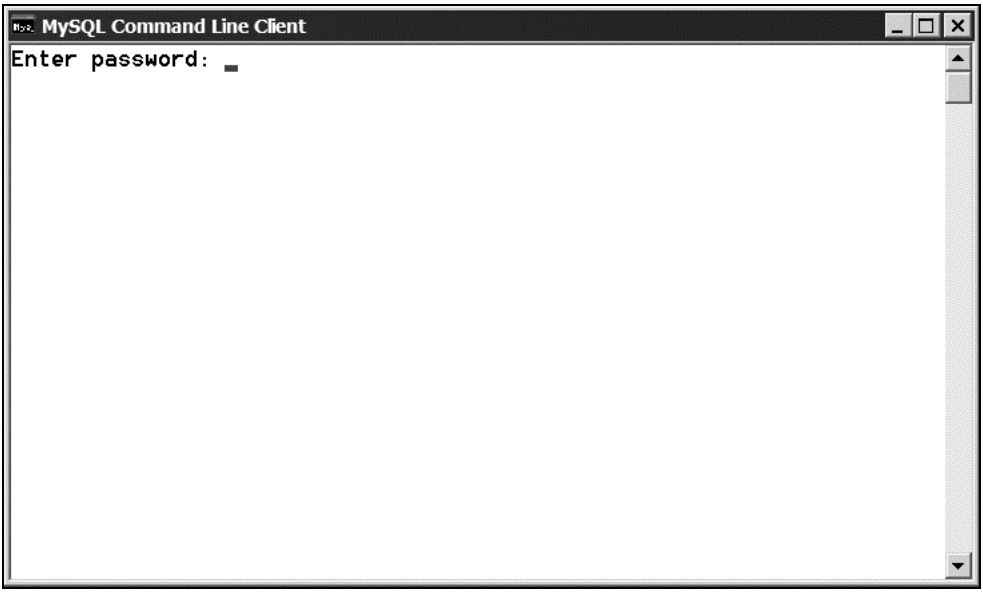


Рис. 2.20. Окно командной строки утилиты MySQL Command Line Client

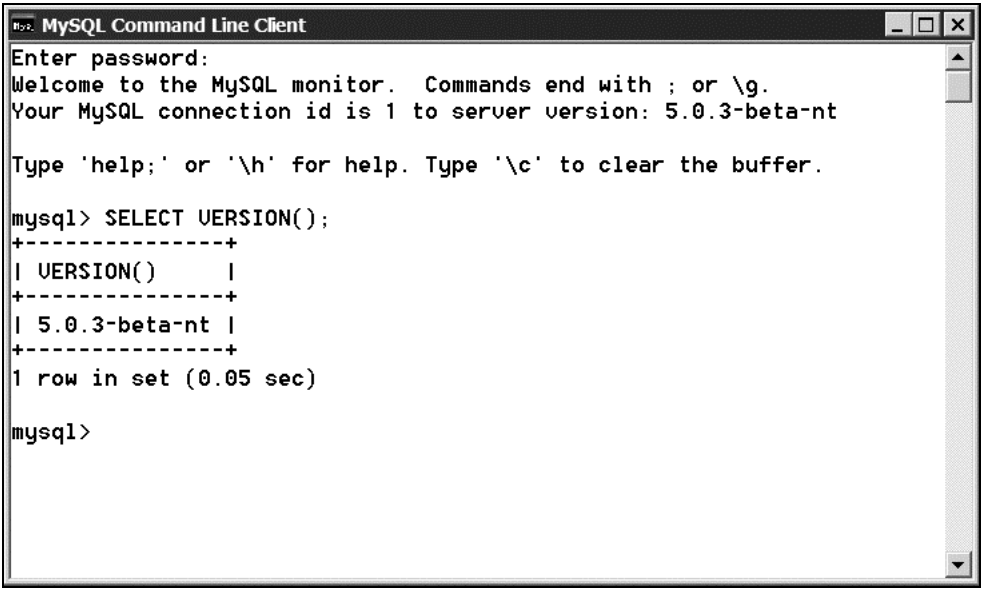


Рис. 2.21. Результат выполнения команды `SELECT VERSION()`

В результате сервер должен показать текущую версию, в нашем случае это `5.0.3-beta-nt` (рис. 2.21).

> ***Замечание***
>
> Для выхода введите команду `EXIT`.

Утилита MySQL Commend Line Client может не запуститься, если в системе отсутствует анонимный пользователь (пользователь, в качестве имени которого выступает пустая строка). В этом случае следует запустить утилиту mysql.exe через командную строку, как это описывается в *главе 3*, либо создать в корне диска C: конфигурационный файл my.ini следующего содержания

```
[mysql]
user=root
```

Это сообщит утилите MySQL Commend Line Client, что для запуска следует использовать учетную запись `root`, а не анонимного пользователя.

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. 2.22.

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** говорит о необходимости запуска сервиса пользователем через консоль управления сервисами. Можно изменить режим запуска сервиса, выбрав в контекстном меню сервиса пункт **Свойства**.

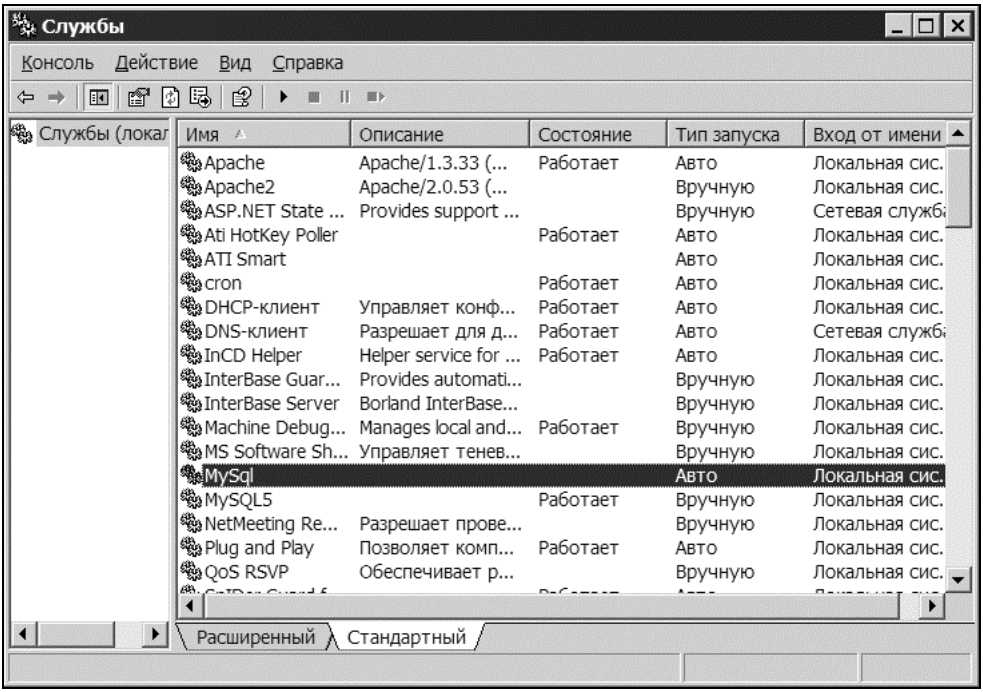


Рис. 2.22. Консоль управления сервисами

2.3. Установка на платформу 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 учетной записи `root`: `/sbin/service mysql stop`.

Чтобы просмотреть все файлы в пакете RPM, выполните команду

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

Для того чтобы выяснить требования пакета, необходимо выполнить команду

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

Для осуществления стандартной установки выполните команды, предварительно войдя в систему как `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 игнорирование 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 2.1.

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

Листинг 2.1. 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:

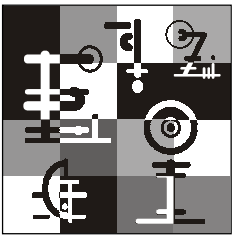
```
/etc/rc.d/init.d/mysql start
```

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
```



Глава 3

Работа с утилитами 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 сервера.

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

> ***Замечание***
>
> Символ d на конце имени программы происходит от термина "demon" (демон) — резидентной программы, которая постоянно находится в памяти. Резидентные программы в UNIX традиционно называют демонами (духами).

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. Утилита описывается в *разд. 3.1*;
- `mysqladmin` — утилита для выполнения административных функций, таких как создание или удаление баз данных, получения информации с сервера о номере версии, процессах, состоянии сервера и т. п. Утилита описывается в *разд. 28.10*;

- `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 дампом (dump) базы данных. Утилита описывается в *разд. 3.2*;
- `mysqlhotcopy` — утилита для создания резервной копии таблиц без остановки сервера MySQL, т. е. создания "горячей" копии базы данных. Утилита описывается в *разд. 31.8*;
- `mysqlimport` — выполняет перенос информации из текстового файла в таблицы базы данных. Утилита описывается в *разд. 6.4*;
- `mysqlshow` — отображает информацию о существующих базах данных, таблицах, полях и индексах. Утилита описывается в *разд. 30.20*.

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. Утилита описывается в *разд. 31.9*;
- `mysqlbinlog` — данная утилита используется для чтения содержимого журнала двоичной регистрации при восстановлении данных в нештатных ситуациях. Утилита описывается в *разд. 28.4.5*;
- `perror` — утилита, которая выводит расшифровку кодов системных ошибок и ошибок MySQL;
- `replace` — утилита замены строк в файлах или стандартном потоке.

На протяжении ближайших нескольких глав нам потребуются только две из названных выше утилит: `mysql` и `mysqldump`, которые будут рассмотрены наиболее подробно. Остальные утилиты будут рассмотрены по мере изучения материала.

> ***Замечание***
>
> В стандартном дистрибутиве MySQL поставляются только консольные утилиты. Пользователям Windows и X Window часто трудно приспособиться к командной строке. Решением этой проблемы может стать выбор клиента с графическим интерфейсом. Так, на Web-странице **http://dev.mysql.com/downloads/** для свободной загрузки доступны графические клиенты MySQL Administrator, MySQL Query Browser и MySQL Control Center. Данные программы можно найти на компакт-диске, прилагаемом к данной книге (каталог clients).

3.1. 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**. При такой

форме запуска утилита `mysql` не позволяет вводить никакие 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. 3.1.

> ***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. 3.1. Окно командной строки

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 его свойствах 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 данной главе, позволяет изменять режим работы утилит.

Для соединения с сервером базы данных в параметрах утилиты `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. 3.2, после вывода версии сервера и информации о справочных ключах выводится приглашение `mysql>`, за которым можно набирать команды.

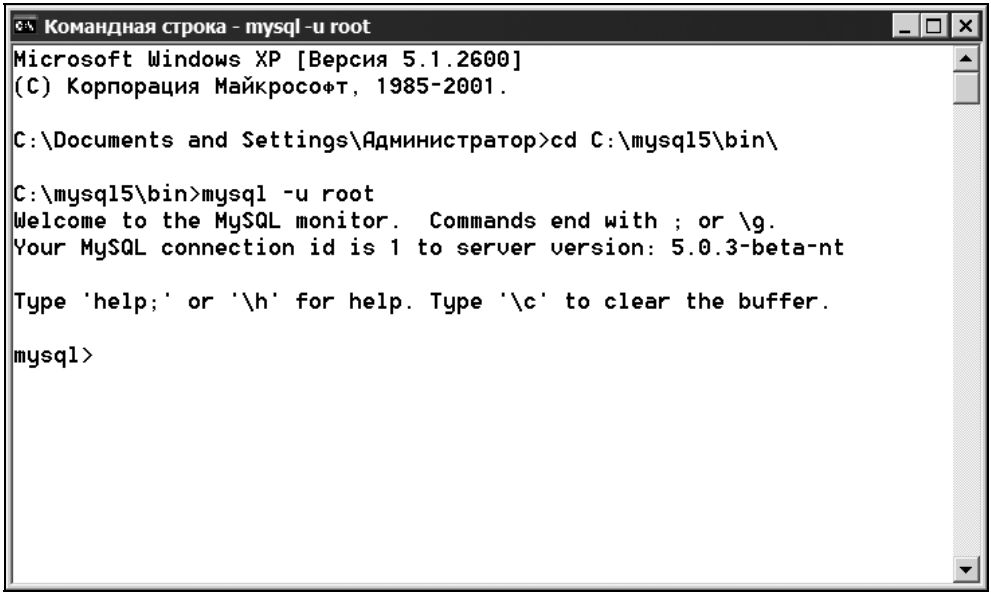


Рис. 3.2. Приглашение `mysql>` консольного клиента `mysql`

Для демонстрации полноценной авторизации создадим нового пользователя `user` с паролем `hello` при помощи инструкции `CREATE USER`, представленной в листинге 3.1. Эту инструкцию следует набрать сразу после приглашения `mysql>`, как это показано на рис. 3.3.

> ***Замечание***
>
> Подробнее проблема безопасности, создания и удаления пользователей обсуждается в *главе 27*.

Листинг 3.1. Создание нового пользователя `user`

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

После этого выйдем из оболочки `mysql` и попробуем зайти из-под нового пользователя. Выход из оболочки `mysql` производится при помощи команды `EXIT` (`exit`) или `QUIT` (`qiut`) — рис. 3.3.

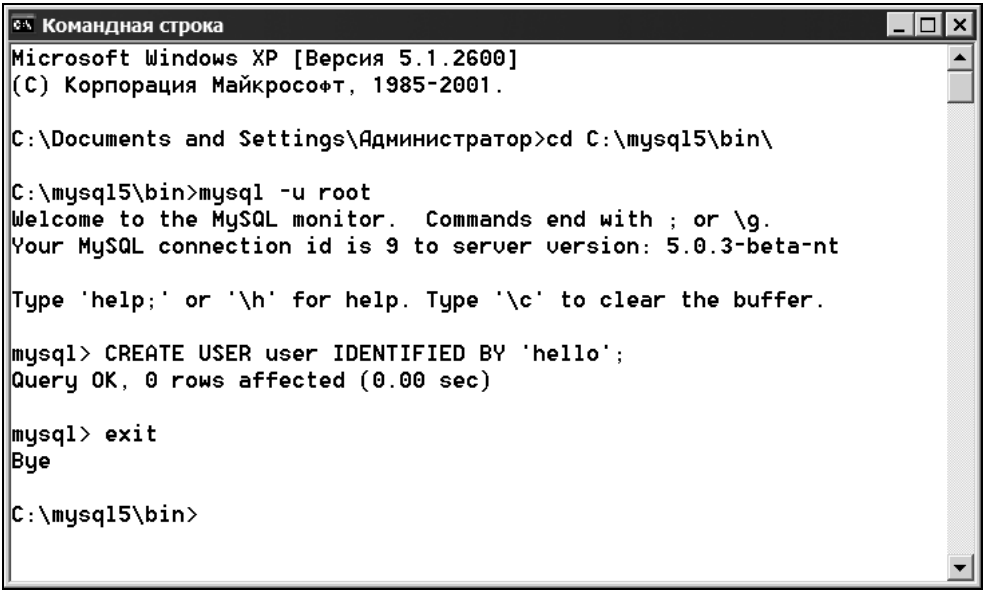


Рис. 3.3. Создание пользователя `user` с паролем `hello`

Теперь команда `mysql -u user` не проходит — сервер отказывается предоставить доступ, без подтверждения полномочия паролем (рис. 3.4). 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. 3.5).

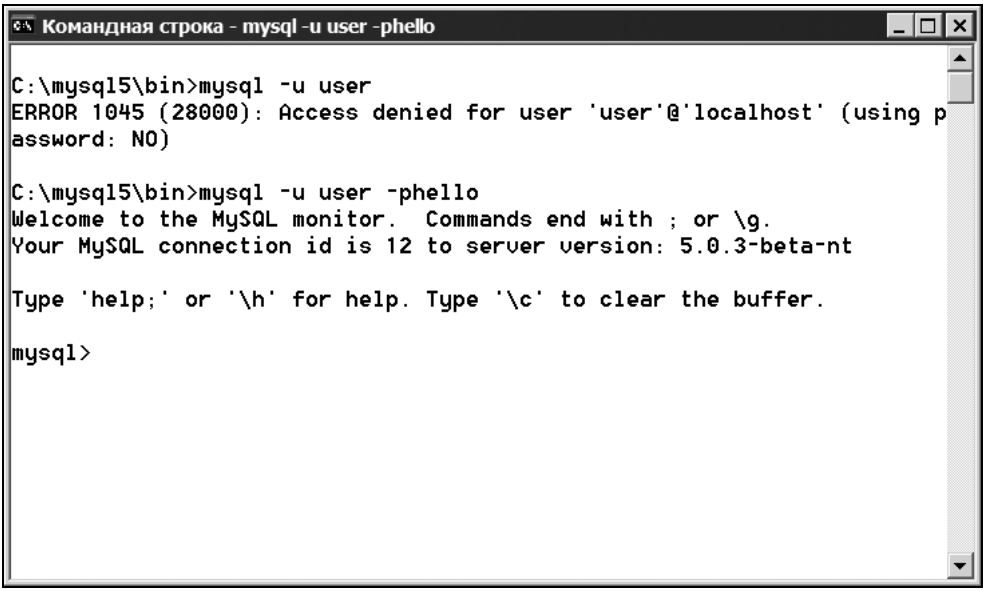


Рис. 3.4. Авторизация с использованием пароля

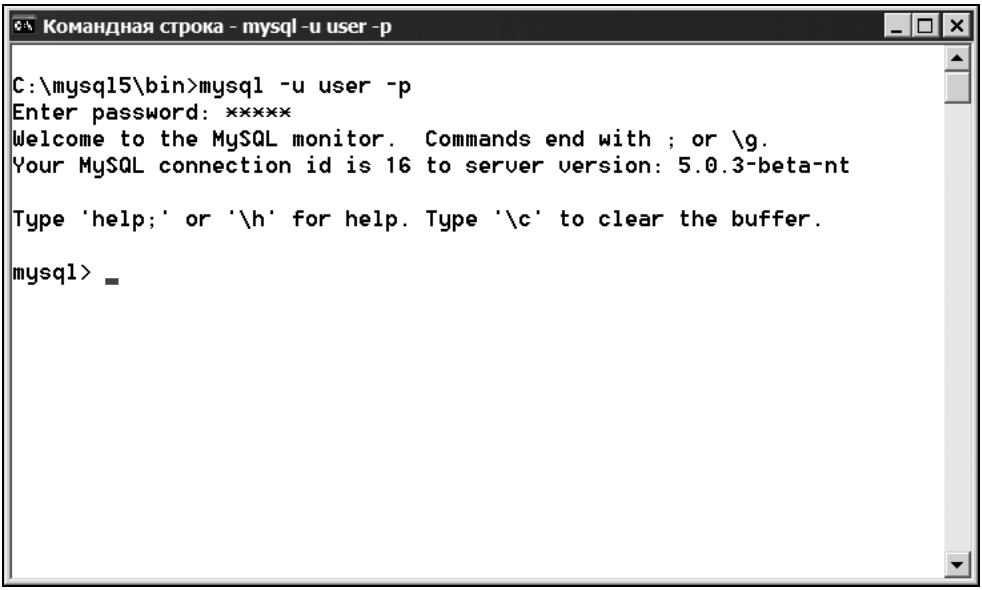


Рис. 3.5. Альтернативный способ авторизации

Команды и SQL-инструкции, за редким исключением (`exit`, `quit`, `use`), должны заканчиваться точкой с запятой. Например, на рис. 3.6 приведена команда, запрашивающая у сервера MySQL его версию и текущую дату.

> ***Версия***
>
> 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 при помощи параметра `--delimiter=`*`name`* *(см. главу 28)*.

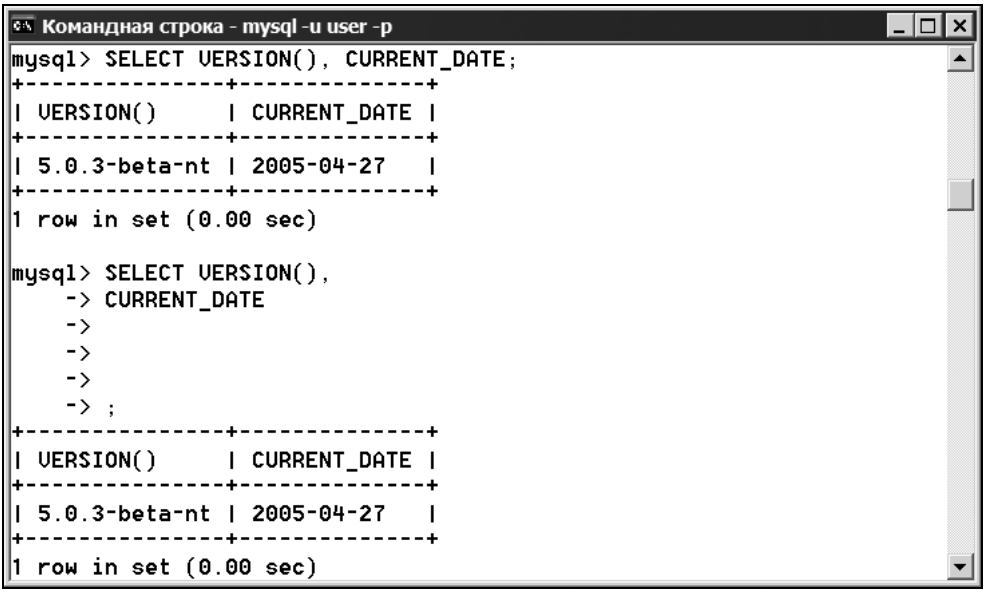


Рис. 3.6. Выполнение команд

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.

> ***Замечание***
>
> Для сокращения листингов в следующих главах книги число возвращаемых строк и время выполнения запроса будет опускаться.

Для ввода ключевых слов можно использовать любой регистр символов. Так приведенные в листинге 3.2 запросы абсолютно идентичны.

Листинг 3.2. Имена инструкций и ключевые слова не зависят от регистра символов

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> 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 `->` (рис. 3.6). 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. 3.7).

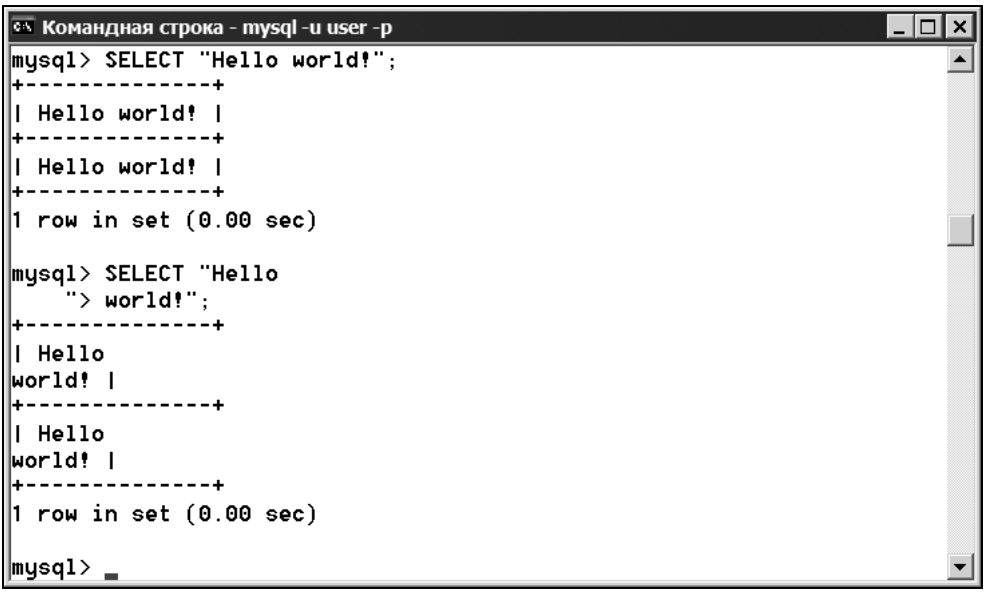


Рис. 3.7. Многострочный ввод текста, заключенного в двойные кавычки

Как видно из рис. 3.7, при переходе не 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. 3.1.

Таблица 3.1. *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

Параметры в утилитах MySQL могут иметь две формы: полную, начинающуюся с двух дефисов (например, `--user`), и краткую, которая начинается с одного дефиса (например, `-u`). Можно применять оба варианта, но для ряда параметров имеется только полная форма. В табл. 3.2 приведены стандартные параметры, применять которые можно со всеми утилитами MySQL.

Замечание

Параметры зависят от регистра, поэтому `-v` и `-V` — это не одно и то же.

Таблица 3.2. *Параметры, общие для всех утилит MySQL*

Параметр	Описание
`--character-sets-dir=`*`dir_name`*	Параметр указывает имя каталога *`dir_name`*, в котором хранятся файлы кодировок
`--default-character-set=`*`charset`*	В параметре задается кодировка *`charset`* по умолчанию
`--compress`, `-C`	Параметр активирует сжатие данных при обмене по сети между клиентом и сервером, если они оба поддерживают сжатие
`--debug[=`*`option`*`]`, `-#[ `*`option`*`]`	Данный параметр требует записи в журнал отладочной информации. Если значение *`option`* не задается, информация выводится в формате `d:t:o, file_name`. Для работы параметра необходимо, чтобы СУБД MySQL была скомпилирована с отладочным выводом
`--help`, `-I`, `-?`	Данный параметр требует от программы вывести справочную информацию и завершить работу
`--host=`*`host_name`*, `-h `*`host_name`*	В параметре указывается сетевое имя или IP-адрес компьютера *`host_name`*, на котором установлен MySQL-сервер
`--password[=`*`pas_val`*`]`, `-p[`*`pas_val`*`]`	В данном параметре указывается пароль *`pas_val`* для доступа к MySQL-серверу
`--port=`*`port_name`*, `-P `*`port_name`*	Параметр указывает номер порта *`port_name`*, по которому MySQL-сервер ожидает соединения с клиентом (по умолчанию 3306; если данный параметр не указан, производится попытка установить соединение именно по данному порту)
`--set-variable var=`*`value`*, `-O var=`*`value`*	Данный параметр позволяет задать значения переменных, которые более подробно рассматриваются в *главах 28* и *29*
`--silent`, `-s`	Данный параметр определяет так называемый "тихий" режим, при котором программа старается выводить минимальный объем информации
`--socket=`*`path_name`*, `-S `*`path_name`*	Через данный параметр можно указать путь к сокету, используемый для подключения к серверу. Сокет часто служит в качестве альтернативы соединению по TCP/IP, т. к. скорость и безопасность при его использовании возрастает
`--user=`*`user_name`*, `-u `*`user_name`*	Параметр указывает пользователя *`user_name`*, от имени которого осуществляется соединение с MySQL-сервером
`--version`, `-V`	Указание данного параметра требует от утилиты вывести строку с информацией о текущей версии и завершить работу

Помимо общих для всех утилит параметров, утилита `mysql` поддерживает специфические параметры, представленные в табл. 3.3.

> ***Замечание***
>
> Смысл многих параметров будет понятен лишь при более подробном рассмотрении операторов SQL в последующих главах. Поэтому при первом прочтении табл. 3.3 можно пропустить.

Таблица 3.3. *Специфические параметры утилиты* `mysql`

Параметр	Описание
`--batch`, `-B`	Данный параметр запускает утилиту `mysql` в пакетном режиме. В этом режиме каждая запись выводится в отдельной строке, а значения столбцов разделены символом табуляции
`--database=`*`name`*, `-D`	Через этот параметр передается имя базы данных *`name`*, с которой будет осуществляться работа
`--debug-info`, `-T`	Передача этого параметра требует вывода отладочной информации по завершению программы
`--delimiter=`*`name`*	Задает разделитель запросов *`name`*, по умолчанию используется точка с запятой (поддерживается, начиная с MySQL 5.0.0)
`--execute=`*`query`*, `-e` *`query`*	При передаче данного параметра утилита `mysql` выполняет запрос *`query`* и завершает работу
`--force`, `-f`	При запуске с данным параметром `mysql` продолжает работу, даже если возникает ошибка в SQL-запросе
`--html`, `-H`	Результирующая таблица выводится в виде HTML-таблицы
`--ignore-space`, `-i`	При запуске с данным параметром утилита `mysql` игнорирует пробелы между именем функции и круглыми скобками в SQL-запросе, в противном случае наличие пробелов будет приводить к ошибке
`--local-infile[={0|1}]`	Данный параметр запрещает (0) или разрешает (1) использовать режим `LOCAL` для оператора `LOAD DATA INFILE`.
`--named-commands`, `-G`	Параметр разрешает длинную форму внутренних команд `mysql`. Внутренние команды описываются в табл. 3.4
`--no-auto-rehash`, `-A`	В процессе запуска `mysql` упорядочивает названия баз данных, таблиц и столбцов, для того чтобы можно было быстро завершить названия при вводе, нажав клавишу <Tab>. Данный параметр позволяет отключить эту процедуру для более быстрого запуска `mysql`
`--no-beep`, `-b`	Обычно при возникновении ошибки слышен звуковой сигнал. Параметр `--no-beep` позволяет его отключить

Таблица 3.3 *(продолжение)*

Параметр	Описание
`--no-named-commands`, `-g`	Параметр отключает длинную форму внутренних команд и требует использования только краткой формы, начинающейся с символа обратного слеша `\`. Внутренние команды описываются в табл. 3.4
`--no-pager`	Отключает использование постраничного вывода результатов запроса при помощи утилиты `less` или `more`. Данный параметр игнорируется в операционной системе Windows
`--no-tee`	Запрещает добавлять копию всего вывода в файл. Данный параметр используется в противовес параметру `--tee`
`--one-database`, `-O`	Данный параметр используется для восстановления базы данных при помощи регистрационного журнала восстановления
`--pager[=`*`program`*`]`	Параметр позволяет задать программу *`program`*, которая будет использоваться для постраничного вывода результатов запроса. Как правило, в качестве таких утилит выступает `less` или `more`. Данный параметр игнорируется в операционной системе Windows
`--prompt=str`	Параметр позволяет задать новую строку подсказки *`str`*, которая будет выводиться вместо `mysql>`. В этом параметре допускаются специальные последовательности, которые описываются в табл. 3.5
`--protocol=` `{TCP|SOCKET|PIPE|MEMORY}`	Параметр позволяет задать протокол подключения к серверу
`--quick`, `-q`	Обычно `mysql` перед выдачей результатов запроса предварительно дожидается окончания поступления их с сервера. Данный параметр позволяет выдавать результат сразу, не дожидаясь их полной загрузки и экономя тем самым память
`--raw`, `-r`	Выводит значения столбцов без преобразования управляющих символов. Обычно используется совместно с `--batch`
`--reconnect`	Если связь с сервером разорвана, осуществляется попытка ее автоматического восстановления
`--safe-updates`, `--i-am-a-dummy`, `-U`	Разрешает только такие операторы `UPDATE` и `DELETE`, в которых используется либо конструкция `WHERE`, либо конструкция `LIMIT`. Это позволяет предотвратить запросы, которые ошибочно удаляют или изменяют всю таблицу целиком
`--secure-auth`	Данный параметр запрещает передавать на сервер пароли в старом формате (который использовался до версии 4.1.1)
`--secure-auth`	Отображать предупреждения, если они имеются

Таблица 3.3 *(окончание)*

Параметр	Описание
`--skip-column-names`, `-N`	Параметр запрещает выводить имена столбцов в результирующем запросе
`--skip-line-numbers`, `-L`	Данный параметр подавляет запись номеров строк при возникновении ошибок
`--table`, `-t`	Данный параметр требует выводить результат в табличном режиме. Такое форматирование включено по умолчанию в интерактивном режиме. Параметр позволяет потребовать использовать аналогичный формат в пакетном режиме
`--tee=`*`file`*	Добавлять копию всего вывода в файл *`file`*
`--unbuffered`, `-n`	При указании данного параметра после каждого запроса `mysql` очищает буфер, используемый для обмена с сервером
`--vertical`, `-E`	При передаче этого параметра результирующие таблицы выводятся в вертикальном стиле
`--wait`, `-W`	Если подключиться к серверу не удается, `mysql` немного ждет и осуществляет попытку соединения повторно
`--xml`, `-X`	Результирующая таблица выводится в формате XML

Интересна возможность установки соединения с удаленным хостом, так в домашней локальной сети одного из авторов, на компьютере с IP-адресом 192.168.200.1 установлен сервер MySQL версии 5.0.18. На рис. 3.8 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 приписать другому порту, например, 3307, в этом случае при установке соединения в параметре `-P` необходимо указывать номер порта (`-P 3307`).

> ***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 27*, посвященной учетным записям MySQL.

Параметр `-e` позволяет без запуска `mysql` в интерактивном режиме получать ответ сервера (рис. 3.9). Этот режим особенно полезен при запуске `mysql` из скриптов.

Иногда результирующие таблицы содержат слишком большое число столбцов (рис. 3.10), в результате структура результирующей таблицы нарушается, и воспринимать информацию практически невозможно.

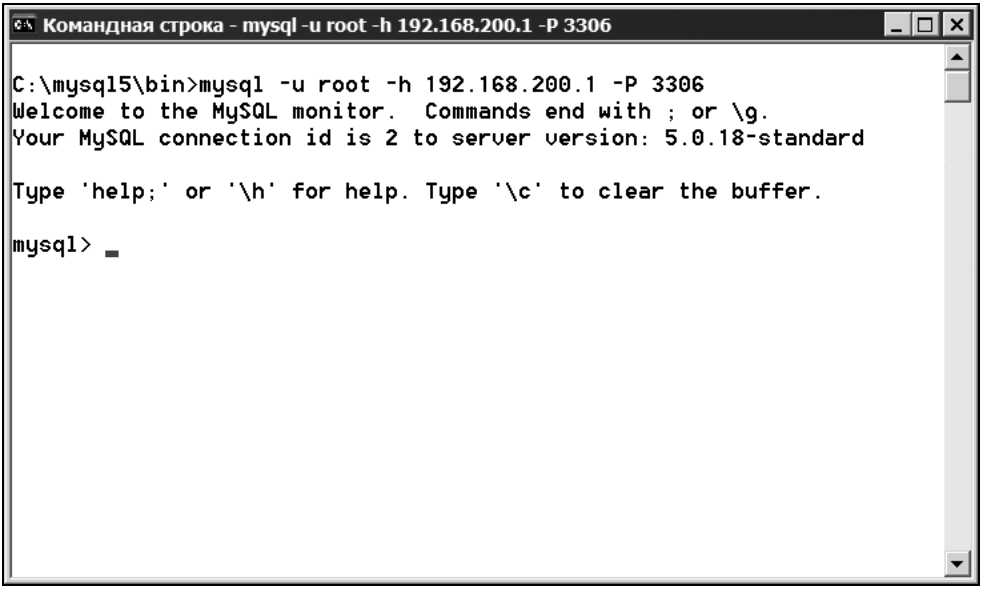


Рис. 3.8. Установка соединения с удаленным сервером 192.468.200.1

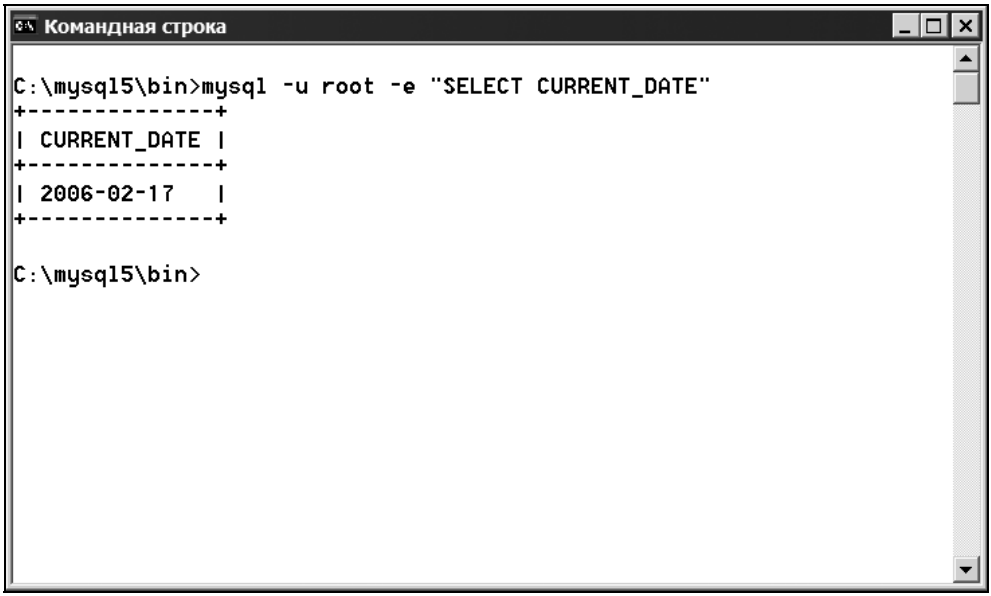


Рис. 3.9. Использование `mysql` без перехода в интерактивный режим

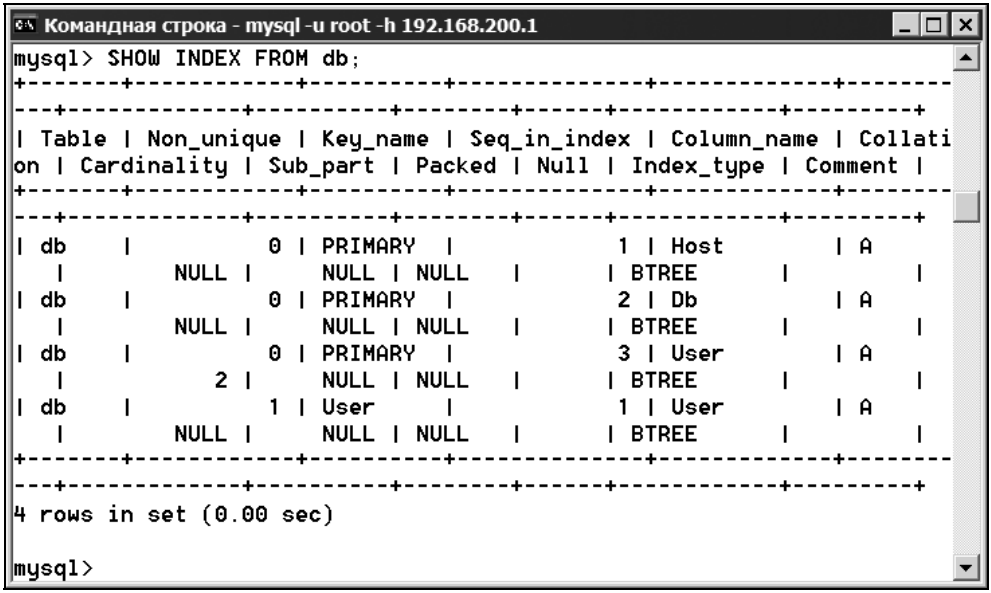


Рис. 3.10. Искажение структуры результирующей таблицы из-за большого числа столбцов

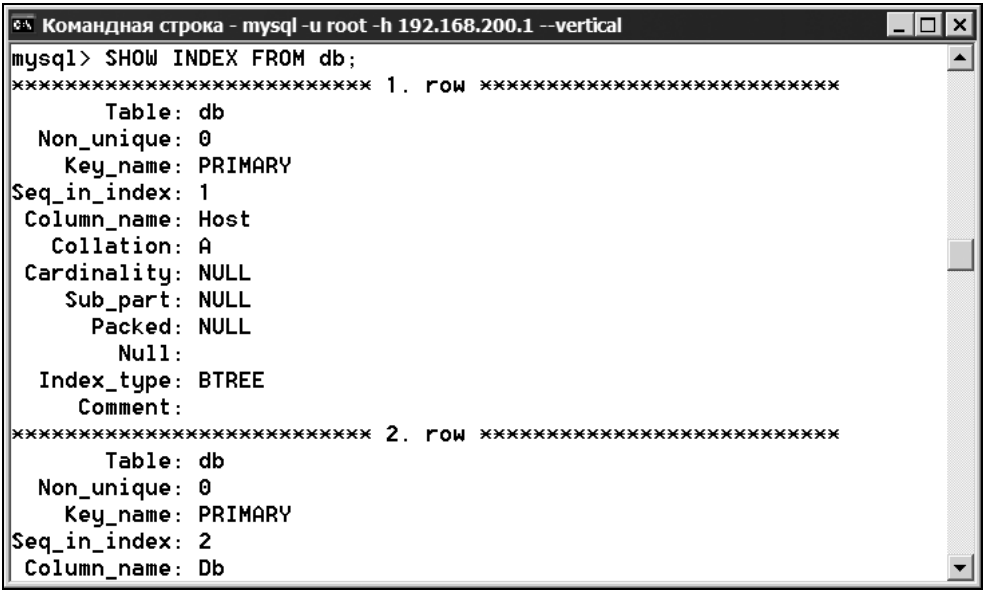


Рис. 3.11. Вертикальный режим вывода результирующей таблицы

Выходом из ситуации является использование так называемого вертикального режима вывода, включить который можно, задав для утилиты `mysql` параметр `--vertical` (см. табл. 3.3). Результат приведенного на рис. 3.10 запроса может выглядеть так, как это представлено на рис. 3.11.

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 применяется команда `\G`, которую следует расположить после запроса (рис. 3.12).

Замечание

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

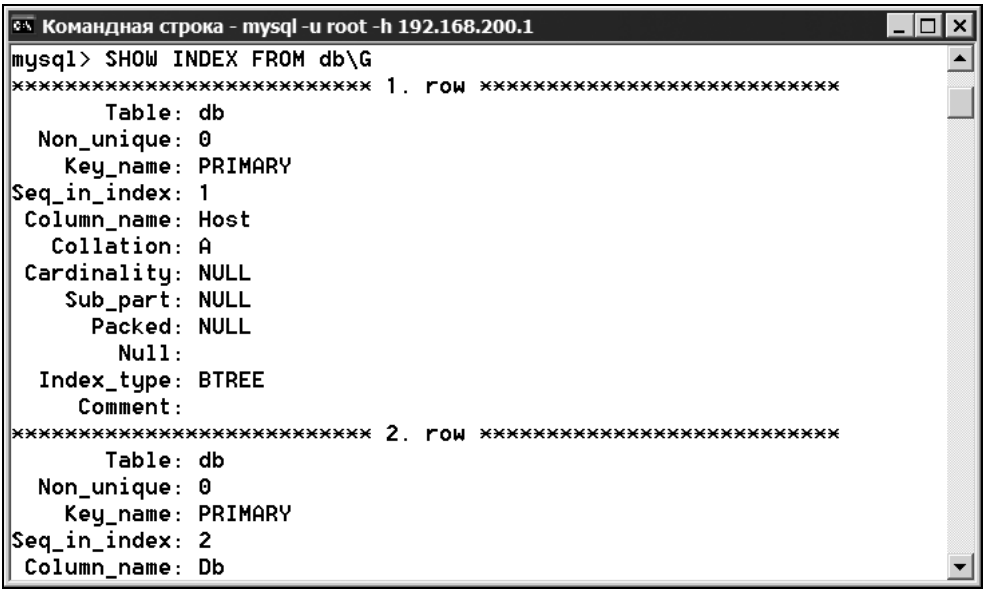


Рис. 3.12. Использование команды `\G` для вызова вертикального режима отображения результатов

Каждая команда имеет краткую и полную форму, полный их список представлен в табл. 3.4. Параметр `--named-commands` позволяет включить использование полной формы внутренних команд, в то время как параметр `--no-named-commands` позволяет отключить полные команды, оставив только короткий вариант.

Замечание

Регистр полных команд не имеет значения, их можно набирать как в верхнем, так и нижнем регистре. Здесь и далее мы будем использовать для обозначения команд верхний регистр. В краткой форме регистр имеет принципиальное значение и его необходимо учитывать.

Таблица 3.4. *Внутренние команды утилиты* `mysql`

Полная форма команды	Краткая форма команды	Описание
`?`	`\h`	Синоним для команды `HELP`
`CLEAR`	`\c`	Отменяет текущую команду, набрав несколько строк, можно отменить ввод команды, включив в строку последовательность `\c`
`CONNECT`	`\r`	Команда используется в формате `CONNECT [db_name [host_name]]` для подключения к базе данных *db_name*, расположенной на хосте *host_name*. Если имя базы данных и хоста не указаны, вместо них используются значения из текущего сеанса `mysql`
`DELIMITER`	`\d`	Данная команда введена в MySQL, начиная с версии 5.0.0, и позволяет изменить разделитель запросов, в качестве которого по умолчанию выступает точка с запятой
`EDIT`	`\e`	Команда позволяет отредактировать текущий запрос во внешнем редакторе, имя которого извлекается из переменной окружения `EDITOR` или `VISUAL`. Если переменные не заданы, используется стандартный UNIX-редактор vi. Данная команда не поддерживается в операционной системе Windows
`EGO`	`\G`	Команда отправляет текущий запрос на сервер и выводит результат в вертикальном формате
`EXIT`	`\q`	Эта команда является полным аналогом `QUIT`
`GO`	`\g`, `;`	Отправляет текущий запрос на сервер и выводит результат. Данная команда является заменителем точки с запятой, которая используется для сообщения, что ввод запроса завершен
`HELP`	`\h`	Отображает справочную информацию о доступных командах в текущей версии `mysql`
`NOPAGER`	`\n`	Отключает постраничный вывод результатов. Данная команда не поддерживается в операционной системе Windows
`NOTEE`	`\t`	Данная команда позволяет отключить запись в файл, которая была включена при помощи параметра `--tee`

Таблица 3.4 *(окончание)*

Полная форма команды	Краткая форма команды	Описание
PAGER	\P	Команда позволяет задать программу для постраничного вывода результата. Не поддерживается в операционной системе Windows
PRINT	\p	Выводит текущую команду (саму команду, а не результат ее выполнения)
PROMT	\R	Команда позволяет переопределить подсказку mysql, заменив mysql> на что-то еще. В этом параметре допускаются специальные последовательности, которые описываются в табл. 3.5
QUIT	\q	Выход из mysql
REHASH	\#	В процессе запуска mysql упорядочивает названия баз данных, таблиц и столбцов, для того чтобы можно было быстро завершить названия при вводе, нажав клавишу <Tab>. Данная команда позволяет перестроить хэш с этими данными
SOURCE	\.	Команда принимает в качестве параметра путь к файлу с SQL-инструкциями, которые она выполняет в пакетном режиме. При использовании в операционной системе Windows при формировании пути к файлу необходимо указывать либо прямой слеш /, либо экранировать обратный \\
STATUS	\s	Извлекает и выводит информацию о состоянии сервера
SYSTEM	\!	Команда SYSTEM *command* позволяет выполнить команду операционной системы *command*. Данная команда не поддерживается в операционной системе Windows
TEE	\T	Команда TEE *file* позволяет копировать весь вывод утилиты mysql в файл *file*
USE	\u	Команда USE *db_name* позволяет выбрать в качестве текущей базы данных *db_name*

Длинный ряд параметров не удобно всякий раз набирать при запуске консольного клиента mysql, поэтому они могут быть прописаны в конфигурационном файле my.ini, который следует поместить в каталог C:\Windows. Даже после того, как параметры размещены в конфигурационном файле my.ini, они могут быть изменены при запуске утилит посредством параметров, которые имеют приоритет над опциями конфигурационного файла.

Как следует из табл. 3.3 и 3.4, приглашение mysql> всегда можно изменить. Для этого можно воспользоваться либо параметром --prompt, либо командой PROMPT. Для последней можно воспользоваться как обычно строкой, например shell>, так и строкой, содержащей специальные значения, которые представлены в табл. 3.5.

Таблица 3.5. *Специальные последовательности команды* `PROMPT`

Последовательность	Описание
`\v`	Версия сервера
`\d`	Текущая база данных
`\h`	Хост сервера
`\p`	Текущий хост TCP/IP
`\u`	Имя пользователя MySQL
`\U`	Полное имя учетной записи, включая хост, имеет формат **user@host**
`\\`	Символ `\\`
`\n`	Символ перевода строки
`\t`	Символ табуляции
`\ `	Пробел (следует за обратным слешем)
`\_`	Пробел
`\R`	Текущее время в 24-часовом формате
`\r`	Текущее время в 12-часовом формате
`\m`	Минуты текущего времени
`\y`	Текущий год, двузначное число
`\Y`	Текущий год, четырехзначное число
`\D`	Полная текущая дата
`\s`	Секунды текущего времени
`\w`	Текущий день недели в трехсимвольном формате (Mon, Tue, ...)
`\P`	Время до полудня/Время после полудня (am/pm)
`\o`	Текущий месяц в числовом формате
`\O`	Текущий месяц в трехсимвольном формате (Jan, Feb, ...)
`\c`	Счетчик, увеличивающийся с каждым введенным оператором

Используя параметр `--prompt`, можно установить новое приглашение в интерактивном редакторе утилиты `mysql`. Порядок использования параметра показан на рис. 3.13.

На рис. 3.14 представлен порядок использования команды `PROMPT`.

Файл my.ini разделен на разделы, которые относятся к разным частям, например, раздел `[mysqld]` относится к серверу MySQL, а раздел `[client]` к консольному клиенту `mysql`. Так в каждом из разделов может быть указана кодировка по умолчанию:

```
default-character-set=cp1251
```

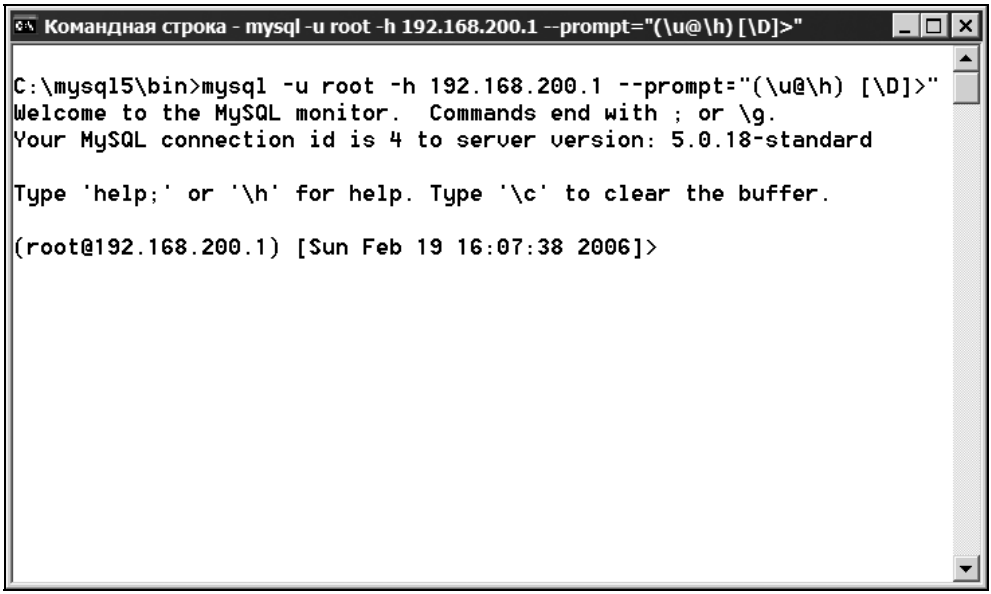


Рис. 3.13. Использование параметра `--prompt`

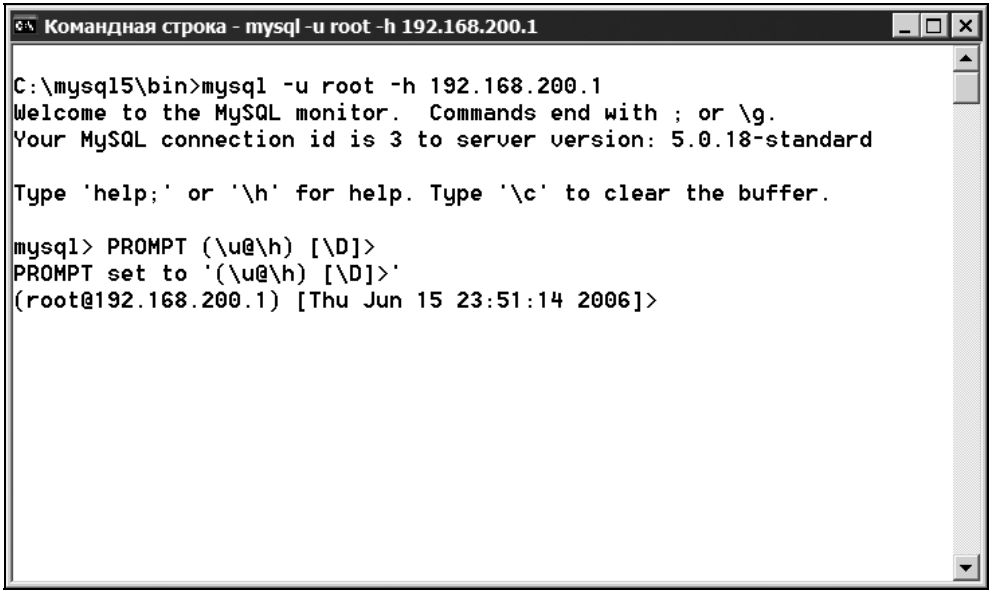


Рис. 3.14. Использование команды `PROMPT`

В разделе `[mysqld]` может быть указан тип таблиц по умолчанию, например, InnoDB

```
default-storage-engine=INNODB
```

или MyISAM

```
default-storage-engine=MYISAM
```

или любой другой. Изменения вступают в силу после рестарта сервера, который можно осуществить в консоли управления сервисами (см. рис. 2.21).

> ***Замечание***
>
> Можно указать версию сервера, например, `[mysqld-5.0]` и `[mysqld-5.1]` задают опции для разных версий серверов MySQL.

> ***Замечание***
>
> Подробнее ознакомиться с типами таблиц MySQL можно в *главе 11*.

В командной строке Windows по умолчанию используется кодировка DOS (cp866), поэтому для того чтобы в консоли русский текст был читаемый, необходимо перед запуском клиента `mysql` изменить кодировку консоли при помощи команды:

```
chcp 1251
```

> ***Замечание***
>
> Использование команды `chcp` без параметра позволяет узнать текущую кодировку командной строки.

> ***Замечание***
>
> Если в командной строке используются "точечные шрифты", после смены кодировки на cp1251 в системном меню следует изменить шрифт на Lucida Console, т. к. в данном семействе шрифтов имеются кириллические Windows-шрифты. В "точечных шрифтах" имеется шрифт только для кодировки cp866 (DOS).

3.2. Утилита *mysqldump*

Данная утилита позволяет получить дамп содержимого базы данных или совокупности баз для создания резервной копии или пересылки данных на другой SQL-сервер баз данных (не обязательно MySQL-сервер). Дамп будет содержать набор команд SQL для создания и/или заполнения таблиц.

Для работы с данной утилитой следует сделать небольшое отступление. В MySQL базы данных представляют собой подкаталоги, расположенные в каталоге данных C:\mysql5\data, имена которых совпадают с именами баз данных. Создание в этом каталоге нового подкаталога аналогично процедуре создания новой базы данных при помощи оператора `CREATE DATABASE`. На компакт-диске, прилагаемом к книге, содержится учебная база данных `shop` как в виде готового дампа (base\base.sql), так и в бинарном виде в каталоге base\shop. Для развертывания бинарных файлов достаточно

скопировать каталог base\shop в C:\mysql5\data, и в системе появится новая база данных, которая автоматически распознается сервером MySQL.

Замечание

Подробное описание учебной базы данных `shop` приведено в *главе 4*.

Теперь можно приступать к созданию дампа базы данных `shop`. Для этого в командной строке необходимо выполнить команду, представленную в листинге 3.3.

Листинг 3.3. Создание дампа базы данных `shop`

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

Как видно из листинга 3.3, утилита `mysqldump` принимает имя пользователя при помощи параметра `-u`. Следует отметить, что `mysqldump` принимает все параметры, перечисленные в разделе, посвященном утилите `mysql`. Кроме этого, после всех параметров указывается имя базы данных `shop`, для которой осуществляется создание дампа. Так как вывод данных осуществляется в стандартный поток (за которым по умолчанию закреплен экран монитора), его следует перенаправить в файл (в листинге 3.3 — это `shop.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.

Помимо параметров, представленных в табл. 3.2, `mysqldump` поддерживает несколько специфических параметров. К ним относится параметр `--databases` или в сокращенной форме `-B`. Данный параметр позволяет создать дамп сразу нескольких баз данных, которые можно перечислить через пробел после него.

Так команда, представленная в листинге 3.4, сохраняет дампы баз данных `shop` и `mysql` в файл shop_mysql.sql.

Листинг 3.4. Создание дампа нескольких баз данных

```
C:\mysql5\bin> mysqldump -u root -B shop mysql > shop_mysql.sql
```

Если необходимо сохранить дамп всех баз данных MySQL-сервера, следует воспользоваться параметром `--all-databases` или в сокращенной форме `-A` (листинг 3.5).

Листинг 3.5. Создание дампа всех баз данных MySQL-сервера

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

Полученный в результате дамп базы данных представляет собой текстовый файл с SQL-инструкциями, выполнить которые можно при помощи утилиты `mysql`.

При развертывании дампа базы данных удобно воспользоваться пакетным режимом (листинг 3.6).

Листинг 3.6. Развертывание дампа базы данных с использованием `mysql`

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

В листинге 3.6 данные из дампа `shop.sql` перенаправляются на стандартный вход утилиты `mysql`, которая размещает таблицы базы данных `shop` в базе данных `test`.

Полный список параметров, доступных для использования совместно с утилитой `mysqldump`, представлен в табл. 3.2 и 3.6.

> ***Замечание***
>
> Смысл многих параметров будет понятен лишь при более подробном рассмотрении операторов SQL в последующих главах. Поэтому при первом прочтении табл. 3.6 можно пропустить.

Таблица 3.6. *Параметры утилиты* `mysqldump`

Параметр	Описание
`--add-drop-database`	Добавляет оператор `DROP DATABASE` перед каждым оператором `CREATE DATABASE`
`--add-drop-table`	Добавляет оператор `DROP TABLE IF EXISTS` перед каждым оператором `CREATE TABLE`
`--add-locks`	Добавляет операторы `LOCK TABLE` и `UNLOCK TABLE` до и после блока операторов `INSERT`
`--all-databases`, `-A`	Сохраняет все таблицы из всех баз данных, которые находятся под управлением текущего сервера
`--allow-keywords`	Позволяет создать имена столбцов, которые совпадают с ключевыми словами
`--comments`, `-i`	Данный параметр позволяет добавить в дамп дополнительную информацию, такую, как версия `mysqldump`, версия MySQL, имя хоста, на котором расположен сервер MySQL
`--compact`	Данный параметр требует от `mysqldump` создать дамп, используя как можно более компактный формат. Параметр является противоположным `--comments`
`--compatible=`*`name`*	Параметр генерирует вывод, который совместим с другими СУБД или более старыми версиями MySQL. Вместо ключевого слова *`name`* можно использовать следующие значения: `ansi`, `mysql323`, `mysql40`, `postgresql`, `oracle`, `mssql`, `db2`, `maxdb`, `no_key_options`, `no_table_options`, `no_field_options`. Можно использовать несколько значений, разделив их запятыми

Таблица 3.6 *(продолжение)*

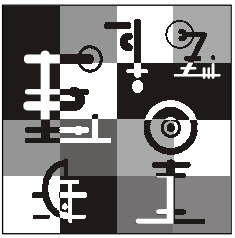
Параметр	Описание
`--complete-insert`, `-c`	Используется полная форма оператора `INSERT`, включая имена столбцов
`--create-options`	Добавляет дополнительную информацию в операторы `CREATE TABLE`. Это может быть тип таблицы, начальное значение `AUTO_INCREMENT` и другие параметры, которые описываются в *главе 12*. До версии MySQL 4.1.2 вместо этого параметра использовался параметр `--all`
`--databases`, `-B`	Параметр позволяет указать имена нескольких баз данных, для которых необходимо создать дамп
`--delayed`	Параметр добавляет ключевое слово `DELAYED` в оператор `INSERT`
`--delete-master-logs`	На главном сервере репликации автоматически удаляются бинарные log-файлы после того, как дамп был успешно создан при помощи `mysqldump`. Этот параметр автоматически включает параметр `--master-data`
`--disable-keys`, `-K`	Для каждой таблицы, окружить оператор `INSERT` фрагментами `/*!40000 ALTER TABLE` *`tbl_name`* `DISABLE KEYS */` и `/*!40000 ALTER TABLE` *`tbl_name`* `ENABLE KEYS */`. Это ускоряет загрузку данных из дампа на сервер для таблиц типа MyISAM
`--extended-insert`, `-e`	Используется многострочная форма оператора `INSERT`
`--fields-terminated-by=...`, `--fields-enclosed-by=...`, `--fields-optionally-enclosed-by=...`, `--fields-escaped-by=...`, `--lines-terminated-by=...`	Параметры этой группы имеют то же значение, что и соответствующие ключевые слова оператора `LOAD DATA INFILE`, синтаксис которого рассматривается в *разд. 6.3*
`--force`, `-f`	Продолжить работу, даже если в процессе создания дампа произошла ошибка
`--hex-blob`	Данный параметр позволяет представить бинарные данные в полях типа `BINARY`, `VARBINARY`, `BLOB` и `BIT` в шестнадцатеричном формате. Так последовательность `'abc'` будет заменена на `0x616263`
`--ignore-table=` *`db_name.tbl_name`*	Данный параметр позволяет игнорировать таблицу *`tbl_name`* из базы данных *`db_name`* при создании дампа. Если из дампа необходимо исключить несколько таблиц, необходимо использовать несколько параметров `--ignore-table`, указывая по одной таблице в каждом из параметров

Таблица 3.6 *(продолжение)*

Параметр	Описание
`--insert-ignore`	Параметр добавляет ключевое слово `IGNORE` в оператор `INSERT`
`--lock-all-tables`, `-x`	Указание этого параметра приводит к блокировке всех таблиц во всех базах данных на время создания полного дампа всех баз данных
`--lock-tables`, `-l`	Указание этого параметра приводит к блокировке таблиц базы данных, для которой создается дамп
`--no-autocommit`	Данный параметр включает все операторы `INSERT`, относящиеся к одной таблице, в одну транзакцию, что приводит к увеличению скорости загрузки данных
`--no-create-db`, `-n`	Параметр подавляет создание в дампе операторов `CREATE DATABASE`, которые автоматически добавляются при использовании параметров `--databases` и `--all-databases`
`--no-data`, `-d`	Параметр подавляет создание операторов `INSERT` в дампе, что может быть полезно при создании дампа структуры базы данных без самих данных
`--opt`	Параметр предназначен для оптимизации скорости резервирования данных и является сокращением, включающим следующие опции: `--add-drop-table`, `--add-locks`, `--create-options`, `--disable-keys`, `--extended-insert`, `--lock-tables`, `--quick`, `--set-charset`. Начиная с MySQL 4.1, параметр `--opt` используется по умолчанию, т. е. все вышеперечисленные параметры включаются по умолчанию, даже если они не указываются. Для того чтобы исключить такое поведение, необходимо воспользоваться параметром `--skip-opt`
`--order-by-primary`	Указание параметра приводит к тому, что каждая таблица сортируется по первичному ключу или первому уникальному индексу
`--protocol={TCP|SOCKET|PIPE|MEMORY}`	Параметр позволяет задать протокол подключения к серверу
`--quick`, `-q`	Обычно `mysqldump` перед формированием дампа предварительно дожидается окончания поступления их с сервера. Данный параметр позволяет начать формирование дампа, не дожидаясь их полной загрузки и экономя тем самым память
`--quote-names`, `-Q`	Помещает имена баз данных, таблиц и столбцов в обратные апострофы `. Начиная с MySQL 4.1, данный параметр включен по умолчанию

Таблица 3.6 *(окончание)*

Параметр	Описание
`--replace`	Параметр добавляет ключевое слово `REPLACE` в оператор `INSERT`. Данный параметр впервые появился в MySQL 5.1.3
`--result-file=`*`file`*, `-r` *`file`*	Параметр направляет дамп в файл *`file`*. Этот параметр особенно удобен в Windows, без использования командной строки, когда можно перенаправить результаты в файл при помощи последовательностей `>` и `>>`
`--routines`, `-R`	Данный параметр создает дамп хранимых процедур и функций. Параметр впервые появился в MySQL 5.1.2
`--single-transaction`	Параметр создает дамп в виде одной транзакции
`--skip-comments`	Данный параметр позволяет подавить вывод в дамп дополнительной информации
`--tab=path`, `-T path`	При использовании этого параметра в каталоге *`path`* для каждой таблицы создаются два отдельных файла: *имя_таблицы*.sql, содержащий оператор `CREATE TABLE`, и *имя_таблицы*.txt, который содержит данные таблицы, разделенные символом табуляции. Формат данных может быть переопределен явно с помощью параметров `--fields-xxx` и `--lines-xxx`
`--tables`	Перекрывает действие параметра `--databases`. Все аргументы, следующие за этим параметром, трактуются как имена таблиц
`--triggers`	При использовании параметра `--triggers` создается дамп триггеров. Этот параметр включен по умолчанию, для его отключения необходимо использовать `--skip-triggers`
`--tz-utc`	При использовании данного параметра в дамп будет добавлен оператор вида `SET TIME_ZONE='+00:00'`, который позволит обмениваться дампами в различных временных зонах
`--verbose`, `-v`	Подробный режим: `mysqldump` выводит подробную информацию о том, что делает
`--where='`*`where_condition`*`'`, `-w '`*`where_condition`*`'`	При использовании данного параметра, `mysqldump` помещает в дамп только те строки, которые удовлетворяют условию *`where_condition`*
`--xml`, `-X`	Результирующая таблица выводится в формате XML



Глава 4

Создание баз данных и таблиц. Типы данных

Данные в любой СУБД организованы в многоуровневые структуры. На верхнем уровне расположена база данных, в состав которой входят таблицы. Каждая таблица разделена на столбцы и строки, на пересечении которых содержатся значения данных. Каждый столбец имеет свой собственный тип.

> ***Замечание***
>
> MySQL не поддерживает каталоги баз данных, как это свойственно некоторым СУБД.

4.1. Создание базы данных

Как было сказано в *главе 3*, 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 4.1 приведен пример создания базы данных `wet`.

Листинг 4.1. Создание базы данных `wet`

```
mysql> CREATE DATABASE wet;
Query OK, 1 row affected (0.00 sec)
```

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 4.2).

Замечание

Более подробно оператор `SHOW` обсуждается в *главе 30*.

Листинг 4.2. Использование оператора `SHOW DATABASES`

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

Как видно из листинга 4.2, оператор `SHOW DATABASES` вернул имена пяти 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.

База данных `shop` была создана в *главе 3*, а база данных `wet` — при помощи SQL-запроса в листинге 4.1. База данных `test` является пустой и создается при установке MySQL вместе с системными базами данных `information_schema` и `mysql`.

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 4.3).

Листинг 4.3. Удаление базы данных `wet`

```
mysql> DROP DATABASE wet;
mysql> SHOW DATABASES;
```

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| shop               |
| test               |
+--------------------+
```

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

Если производится попытка создания уже существующей базы данных, возвращается ошибка (листинг 4.4).

> ***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 4.4. Сообщение об ошибке

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

Часто такое поведение нежелательно, особенно при работе с объемными дампами, размещенными в sql-файлах, 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 4.5).

Листинг 4.5. Использование конструкции `IF NOT EXISTS`

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

Как видно из листинга 4.5, после удаления базы данных `wet` первый оператор `CREATE DATABASE` возвращает ответ, в котором говорится, 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 4.6).

Листинг 4.6. Использование конструкции `IF EXISTS`

```
mysql> DROP DATABASE IF EXISTS wet;
Query OK, 1 rows affected (0.03 sec)
mysql> 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 переименовать каталог базы данных в каталоге данных C:\mysql5\data\. Пример использования оператора `RENAME DATABASE` приведен в листинге 4.7, где база данных `wet` переименовывается в `complete`.

Листинг 4.7. Использование оператора `RENAME DATABASE`

```
mysql> 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 ключевое слово `DEFAULT CHARACTER SET` *charset_name*, где *charset_name* — имя кодировки, например, `cp1251`, которая обозначает русскую Windows-кодировку (листинг 4.8).

Листинг 4.8. Назначение кодировки по умолчанию

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

Замечание

Подробнее кодировки обсуждаются в *главе 12*, а с полным списком кодировок можно ознакомиться в листинге 12.4.

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
```

Если при конфигурировании MySQL утилитой MySQL Server Instance Configuration Wizard или при ручном редактировании конфигурационного файла my.ini в качестве кодировки по умолчанию для MySQL-сервера была выбрана другая кодировка, например, `latain1`, то при отсутствии конструкции `DEFAULT CHARACTER SET` вместо `cp1251` будет использована именно `latain1`. Если затем кодировка не будет меняться при определении таблицы и столбцов, это может привести к неправильной сортировке русского текста.

Замечание

Подробное описание конфигурирования MySQL-сервера при помощи утилиты **MySQL Server Instance Configuration Wizard** приводится в *главе 2*.

Для изменения параметров базы данных можно воспользоваться оператором `ALTER DATABASE`, как это продемонстрировано в листинге 4.9.

Листинг 4.9. Изменение кодировки базы данных

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

В листинге 4.9 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, разрешено опускать имя базы данных, т. е. вполне допустима запись вида `ALTER DATABASE CHARACTER SET cp1251`.

4.2. Создание таблицы

Перед тем как создать таблицу, в клиентской программе следует выбрать базу данных, с которой будет производиться работа. Эта операция осуществляется при помощи команды `USE` (листинг 4.10).

Замечание

Команда `USE` не является оператором SQL и не применяется в СУБД, отличных от MySQL. С полным списком внутренних команд можно ознакомиться в табл. 3.4.

Листинг 4.10. Выбор базы данных

```
mysql> USE test;
Database changed
```

Замечание

Прежде чем работать с таблицами, всегда производится выбор базы данных. Это относится и к прикладным программам, работающим с базой данных. Обычно с этой целью применяется функция с суффиксом `_select_db`.

Для создания таблицы используется оператор `CREATE TABLE`, после которого следует имя создаваемой таблицы и в круглых скобках — структура таблицы (листинг 4.11).

Листинг 4.11. Создание таблицы `tbl`

```
mysql> CREATE TABLE tbl (number INT, name TEXT);
Query OK, 0 rows affected (0.06 sec)
```

В круглых скобках через запятую указываются имена столбцов и их тип. Так, созданная таблица `tbl` имеет два поля: `number`, куда можно помещать целочисленные значения (`INT`), и `name`, в котором размещаются строки (`TEXT`). Максимальная длина имен таблиц и столбцов составляет 64 знака и может включать литеры, цифры и символы _ и $. Имя может начинаться с цифры, но не должно полностью состоять из цифр.

Замечание

Более подробно типы столбцов обсуждаются далее в этой главе. Синтаксис SQL-оператора `CREATE TABLE` полностью рассматривается в *главе 12*.

Замечание

В операционной системе Windows имена таблиц и баз данных не зависят от регистра, тогда как в UNIX-подобных операционных системах — зависят. Это связано с тем, что база данных является каталогом, а таблицы — файлами, имена которых совпадают с именами баз данных и таблиц. В файловой системе Windows имена файлов не зависят от регистра, в то время как в UNIX имена, отличающиеся только регистром, считаются разными файлами.

Проконтролировать вновь созданную таблицу можно при помощи оператора `SHOW TABLES`, который возвращает список таблиц текущей базы данных (листинг 4.12).

Листинг 4.12. Использование оператора `SHOW TABLES`

```
mysql> SHOW TABLES;
+---------------+
| Tables_in_test |
+---------------+
| tbl            |
+---------------+
```

Для того чтобы просмотреть содержимое другой базы данных, следует предварительно выбрать ее с использованием оператора `USE` (листинг 4.13).

Листинг 4.13. Использование оператора `USE`

```
mysql> USE shop;
Database changed
mysql> SHOW TABLES;
+---------------+
| Tables_in_shop |
+---------------+
| catalogs       |
| orders         |
| products       |
| users          |
+---------------+
```

Как видно из листинга 4.13, учебная база данных содержит четыре таблицы: `catalogs`, `orders`, `products` и `users`.

> ***Замечание***
>
> Подробнее структура учебной базы данных `shop` рассматривается в *разд. 4.4*.

Удаление таблицы производится при помощи оператора `DROP TABLE` (листинг 4.14).

Листинг 4.14. Использование оператора `DROP TABLE`

```
mysql> USE test;
Database changed
mysql> DROP TABLE tbl;
Query OK, 0 rows affected (0.02 sec)
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

> ***Замечание***
>
> Наряду с обращением к таблице по ее имени, например, `tbl`, существует расширенное обращение, включающее в себя имя базы данных, например, `test.tbl`. Любой SQL-оператор может обращаться к базе данных по этому имени. Так, оператор `DROP TABLE` из листинга 4.14 может выглядеть следующим образом: `DROP TABLE test.tbl`. При использовании расширенного синтаксиса допускается не производить выбор базы данных оператором `USE`, т. к. база данных явно указывается в запросе.

Для того чтобы получить описание столбцов базы данных, можно воспользоваться оператором `DESCRIBE` (листинг 4.15).

Листинг 4.15. Использование оператора `DESCRIBE`

```
mysql> DESCRIBE test.tbl;
+--------+---------+------+-----+---------+-------+
| Field  | Type    | Null | Key | Default | Extra |
+--------+---------+------+-----+---------+-------+
| number | int(11) | YES  |     | NULL    |       |
| name   | text    | YES  |     | NULL    |       |
+--------+---------+------+-----+---------+-------+
```

Как видно из листинга 4.15, имя таблицы приведено в расширенном формате `test.tbl`, однако вполне допустимо указание одного имени таблицы `tbl`, если в качестве текущей базы данных выбрана `test`.

> ***Замечание***
>
> Оператор `DESCRIBE` не является стандартным оператором языка запросов SQL и характерен только для диалекта MySQL.

Оператор `DESCRIBE` возвращает таблицу, каждая строка которой соответствует столбцу таблицы `tbl`. Имена столбцов и их типы приводятся в первом и втором столбцах, соответственно. К остальным столбцам результирующей таблицы мы вернемся по мере освоения материала.

4.3. Типы данных

MySQL поддерживает несколько типов данных.

- 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.

4.3.1. Числовые данные

Числовые данные делятся на точечные (BIT, BOOLEAN, INTEGER и DECIMAL) и приближенные (FLOAT, REAL и DOUBLE PRECISION). Характеристики и занимаемый объем точечных типов приведены в табл. 4.1.

> **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 4.1. *Точечные типы*

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 65535 (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

Как видно из табл. 4.1, 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.

> **Замечание**
>
> Тип INT имеет синоним INTEGER.

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 нолями 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 число 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.

Для представления вещественных (приближенных) типов в СУБД MySQL имеются три типа: FLOAT, DOUBLE и DECIMAL.

Замечание

Тип DOUBLE имеет два синонима: PRECISION и REAL. Тип DECIMAL имеет синоним NUMERIC.

Характеристики вещественных типов представлены в табл. 4.2.

Таблица 4.2. *Приближенные типы*

Тип	Объем памяти	Диапазон
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 $\pm 1{,}797693134862315 \times 10^{308}$

Диапазон вещественных чисел помимо максимального значения имеет также минимальное значение, которое характеризует точность данного типа. Параметр M в табл. 4.2 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.

Замечание

Приближенные числовые данные могут задаваться в обычной форме, например, 45.67 и в так называемой научной нотации, например 5.456E-02 или 4.674E+04. Символ E указывает на то, что число перед ним следует умножить на 10 в степени, указанной после E. То есть приведенные в примерах числа можно записать как 0.05456 и 46740. Такой формат введен для удобства записи, т. к. числа в 300-й степени неудобно представлять в обычном десятичном виде.

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 4.16 представлен оператор CREATE TABLE, создающий таблицу tbl с двумя полями:

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

Листинг 4.16. Использование оператора `CREATE TABLE`

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

Как видно из листинга 4.16, после типа столбца указано ключевое слово `DEFAULT`. Оно позволяет задать значение поля по умолчанию. Для поля `num1` значение по умолчанию выбрано `0`, а для `num2` — `1.0`. Ключевое слово `DEFAULT` является необязательным и может не указываться, т. е. вполне допустимо создание таблицы при помощи запроса, приведенного в листинге 4.17.

Листинг 4.17. Использование оператора `CREATE TABLE` без ключевого слова `DEFAULT`

```
CREATE TABLE tbl (
  num1 INT,
  num2 FLOAT
)
```

4.3.2. Строковые данные

Для строковых типов данных максимальный размер и требования к памяти приведены в табл. 4.3. Здесь `L` — это длина хранимой в ячейке строки, а байты, приплюсованные к `L` — накладные расходы для хранения длины строки.

Таблица 4.3. *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
`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 выборе строкового типа данных для столбца следует принимать во внимание, что для строк переменной длины VARCHAR требуется количество символов, равное длине строки плюс один байт, в то время как тип CHAR(M), независимо от длины строки, использует для ее хранения все M символов. В то же время тип CHAR обрабатывается эффективнее переменных типов, т. к. всегда заранее известно, где заканчивается очередной блок данных (табл. 4.4).

Таблица 4.4. *Сравнение типов* CHAR *и* VARCHAR

Значение	CHAR(4)		VARCHAR(4)	
	представление	число байтов	представление	число байтов
''	' '	4	''	1
'ab'	' ab'	4	'ab'	3
'abcd'	'abcd'	4	'abcd'	5
'abcdefgh'	'abcd'	4	'abcd'	5

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.

> ***Замечание***
>
> Начиная с версии 4.1.2, типы CHAR и VARCHAR рассматривают строки как последовательности символов. Это означает, что при использовании многобайтных кодировок, например Unicode, размер строки в байтах будет больше, чем в символах. Для совместимости со старыми версиями в MySQL введены два специальных типа: BINARY и VARBINARY, которые эквивалентны типам CHAR и VARCHAR, но строка в них рассматривается как последовательность байтов, а не символов. К BINARY-строкам не применимы кодировки и сортируются они как обычные последовательности байтов.

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.

Замечание

Основным отличием типа TEXT от CHAR и VARCHAR является поддержка возможностей полнотекстового поиска, которые рассматриваются в *главе 20*.

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, значения для столбца SET('y','n') могут 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 4.18 представлен оператор CREATE TABLE, создающий таблицу tbl 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 4.18. 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 4.18, 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'`.

4.3.3. Календарные данные

СУБД MySQL имеет пять видов столбцов для хранения календарных типов данных: `DATE`, `DATETIME`, `TIME`, `TIMESTAMP` и `YEAR` (табл. 4.5). Тип `DATE` предназначен для хранения даты, `TIME` — для времени суток, а `TIMESTAMP` — 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.

Таблица 4.5. *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)`

Для значений, имеющих тип `DATE` и `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`).

> ***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.

Если время задается в недопустимом формате, то в поле записывается нулевое значение. Нулевое значение присваивается полям временно́го типа по умолчанию, когда им не присваивается инициирующее значение (табл. 4.6).

Таблица 4.6. *Нулевые значения календарных типов*

Тип	Нулевое значение
`DATE`	`'0000-00-00'`
`TIME`	`'00:00:00'`
`DATATIME`	`'0000-00-00 00:00:00'`
`TIMESTAMP`	`00000000000000`
`YEAR`	`0000`

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 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 4.19, абсолютно идентичны.

Листинг 4.19. Задание временных 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 4.20 поле puttime1 получает текущее время при создании новой записи, а поле puttime2 при ее модификации.

Листинг 4.20. Задание текущего времени

```
CREATE TABLE tbl(
  puttime1 TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  puttime2 TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

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.

4.3.4. 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 4.21.

Листинг 4.21. 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 него, является текстом комментария. C-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 прини-

мают 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 ни при каких обстоятельствах поле не должно принимать значение NULL (регистрация невозможна, если фамилия пользователя неизвестна), следует указать ключевое слово NOT NULL (листинг 4.22).

Листинг 4.22. 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 4.23).

Листинг 4.23. 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 'O', # 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` — `'O'`, `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.

Применив к таблице `users` оператор `DESCRIBE`, получим результат, представленный в листинге 4.24.

Листинг 4.24. Структура таблицы `users`

```
mysql> 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   |     | O       |       |
| 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. Назначение поля `Key` будет объяснено в *главе 5*.

4.3.5. Выбор типа данных

При выборе типа данных следует помнить, что обработка числовых данных происходит быстрее строковых. Так как типы данных `ENUM` и `SET` имеют внутреннее числовое представление, им следует отдавать предпочтение перед другими видами строковых данных, если предоставляется такая возможность.

Производительность можно увеличить за счет представления строк в виде чисел. Примером может служить преобразование IP-адреса из строковой нотации в `BIGINT`. Это позволит уменьшить размер таблицы и значительно увеличит скорость при сортировке и выборке данных, но потребует дополнительных преобразований.

Не следует забывать, что базы данных хранятся на жестком диске, и чем меньше места они занимают, тем быстрее происходит поиск и извлечение. Поэтому там, где есть возможность, предпочтение следует отдавать типам данных, занимающих меньше места.

Типы фиксированной длины обрабатываются быстрее типов переменной длины, т. к. в последнем случае при частых удалениях и модификациях таблицы происходит ее фрагментация.

> ***Замечание***
>
> Если применение столбцов с данными переменной длины неизбежно, для дефрагментации таблицы следует применять команду `OPTIMIZE TABLE`, синтаксис которой подробно обсуждается в *главе 31*.

4.4. Учебная база данных

На протяжении всей книги описание особенностей диалекта SQL в СУБД MySQL будет вестись на примере учебной базы данных. В качестве такой базы данных выбрана база электронного магазина, торгующего компьютерными комплектующими.

> ***Замечание***
>
> На компакт-диске, поставляемом вместе с данной книгой, можно найти как текстовый дамп учебной базы данных \base\base.sql, так и бинарные файлы базы данных \base\shop. О том, как подключить данные файлы к серверу MySQL, рассказывается в *главе 3*.

База данных компьютерного магазина будет состоять из четырех таблиц:

- `catalogs` — список торговых групп;
- `products` — товарные позиции;
- `users` — список зарегистрированных пользователей магазина;
- `orders` — список осуществленных сделок.

Таблица `catalogs` предназначена для хранения торговых групп, таких как "Материнские платы", "Процессоры", "Видеокарты" и т. п. Таблица состоит из двух полей:

- `id_catalog` — уникальный номер;
- `name` — имя раздела.

Оператор `CREATE TABLE`, создающий таблицу `catalogs`, представлен в листинге 4.25. Оба поля снабжены атрибутом `NOT NULL`, т. к. неопределенное значение для них недопустимо.

Листинг 4.25. Создание таблицы `catalogs`

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

Таблица products содержит конкретные товарные позиции, такие как "Celeron 2.0GHz", "Intel Pentium 4 3.2GHz" и т. п. Таблица состоит из семи полей:

- id_product — уникальный номер товарной позиции;
- name — название товарной позиции;
- price — цена;
- count — количество товарных позиций на складе;
- mark — относительная оценка товара;
- description — описание;
- id_catalog — номер торговой группы из таблицы catalogs, которой принадлежит товарная позиция.

Оператор CREATE TABLE, создающий таблицу products, представлен в листинге 4.26.

Листинг 4.26. Создание таблицы products

```
CREATE TABLE products (
  id_product INT(11) NOT NULL,
  name TINYTEXT NOT NULL,
  price DECIMAL(7,2) NULL default '0.00',
  count INT(11) NULL default '0',
  mark FLOAT(4,1) NOT NULL default '0.0',
  description TEXT NULL,
  id_catalog INT(11) NOT NULL
)
```

Цена товарной позиции price, количество товара на складе count и описание description снабжены атрибутом NULL. Несмотря на то, что в момент привоза зачастую бывает неизвестно количество товара на складе и его текущая цена, отразить факт наличия товара в прайс-листе необходимо. Точно так же и описание description может быть неизвестно из-за того, что товар слишком новый и имеет противоречивое описание в источниках.

Поле id_catalog устанавливает связь между таблицами catalogs и products. Зная уникальный номер каталога id_catalog, всегда можно выяснить, какие товарные позиции принадлежат данной товарной группе и наоборот, к какой товарной группе принадлежит товарная позиция с конкретным значением id_catalog. MySQL позволяет сделать это при помощи несложных SQL-запросов, которые будут рассмотрены в следующих главах.

Таблица users содержит записи с информацией о зарегистрированных покупателях и состоит из восьми полей:

- id_user — уникальный номер покупателя;

- `surname` — фамилия покупателя;
- `patronymic` — отчество покупателя;
- `name` — имя покупателя;
- `phone` — телефон покупателя (если имеется);
- `email` — e-mail покупателя (если имеется);
- `url` — домашняя страница покупателя (если имеется);
- `userstatus` — статус покупателя, поле типа `ENUM`, которое может принимать одно из четырех значений:
  - `active` — авторизованный покупатель, который может осуществлять покупки через Интернет;
  - `passive` — неавторизованный покупатель (значение по умолчанию), который осуществил процедуру регистрации, но не подтвердил ее и пока не может осуществлять покупки в магазине через Интернет, но ему доступны каталоги для просмотра;
  - `lock` — заблокированный покупатель, не может осуществлять покупки и просматривать каталоги магазина;
  - `gold` — активный покупатель с хорошей кредитной историей, которому предоставляется скидка при следующих покупках в магазине.

Оператор `CREATE TABLE`, создающий таблицу `users`, представлен в листинге 4.27.

Листинг 4.27. Создание таблицы `users`

```
CREATE TABLE users (
  id_user INT(11) NOT NULL,
  surname TINYTEXT,
  patronymic TINYTEXT,
  name TINYTEXT,
  phone VARCHAR(12) NULL,
  email TINYTEXT NULL,
  url TINYTEXT NULL,
  userstatus ENUM('active','passive','lock','gold') DEFAULT 'passive'
)
```

Как видно из листинга 4.27, поля `phone`, `email` и `url` снабжены атрибутом `NULL`, т. к. посетитель имеет право не указывать эту информацию. Если эта информация не указана, то она носит неопределенный характер: телефона, e-mail и домашней страницы у покупателя может просто не быть, с другой стороны, он мог не захотеть их указывать. Поэтому, чтобы указать неопределенность этой информации, в поле следует

поместить значение NULL. Все остальные поля получают атрибут NOT NULL, даже если он не указывается.

Таблица orders содержит информацию о покупках, совершенных в магазине, и включает пять полей:

- id_order — уникальный номер сделки;
- id_user — номер пользователя из таблицы users;
- ordertime — время совершения сделки;
- number — число приобретенных товаров;
- id_product — номер товарной позиции из таблицы products.

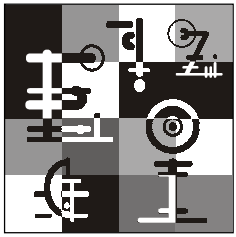
Оператор CREATE TABLE, создающий таблицу orders, представлен в листинге 4.28.

Листинг 4.28. Создание таблицы orders

```
CREATE TABLE orders (
  id_order INT(11) NOT NULL,
  id_user INT(11) NOT NULL,
  ordertime DATETIME NOT NULL,
  number INT(11) NOT NULL,
  id_product INT(11) NOT NULL
)
```

Как видно из листинга 4.28, все столбцы таблицы orders снабжены атрибутом NOT NULL, т. к. при совершении покупки вся информация должна быть доступна и занесена в таблицу.

В таблице orders устанавливается связь сразу с двумя таблицами: users за счет поля id_user, и products за счет поля id_product. Это позволит для каждой покупки восстановить покупателя и приобретенный товар. С другой стороны, можно узнать число покупок, совершенных каждым из покупателей, а также частоту покупки той или иной товарной позиции, что позволит увеличить или уменьшить закупки товара и тем самым произвести оптимизацию работы магазина.



Глава 5

Индексы

Индексы играют большую роль в базах данных, т. к. это основной способ ускорения их работы. Обычно записи в таблице располагаются в хаотическом порядке (рис. 5.1). 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.

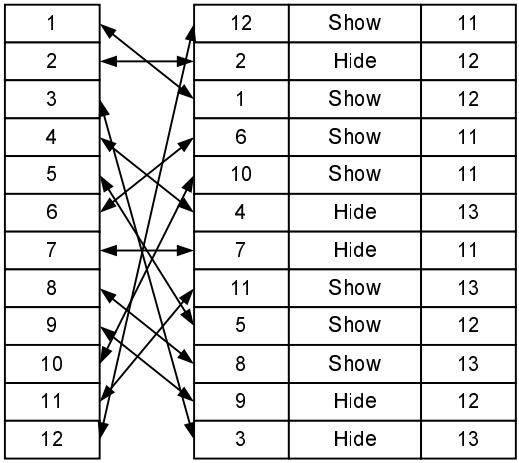


Рис. 5.1. Демонстрация работы индексированных столбцов

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.

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, подробно описывается в *главе 20*.

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`.

> ***Замечание***
>
> В таблице может быть не более 32 индексов, а многостолбцовый индекс может охватывать не более 16 столбцов.

5.1. 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 5.1 приведено определение таблицы `catalogs`, где в качестве первичного ключа назначен столбец `id_catalog`.

> ***Замечание***
>
> Первичный ключ не является обязательной конструкцией, вполне возможно создание таблиц, не содержащих первичных ключей и индексов.

Листинг 5.1. Создание таблицы `catalogs`

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

Просмотр структуры таблицы `catalogs` при помощи оператора `DESCRIBE` показывает, что в четвертой колонке результирующей таблицы напротив поля `id_catalog` появился флаг `PRI`, который отмечает поле первичного ключа (листинг 5.2).

Листинг 5.2. `DESCRIBE products`

```
mysql> DESCRIBE catalogs;
+------------+----------+------+-----+---------+-------+
| Field      | Type     | Null | Key | Default | Extra |
+------------+----------+------+-----+---------+-------+
| id_catalog | int(11)  |      | PRI | 0       |       |
| name       | tinytext |      |     |         |       |
+------------+----------+------+-----+---------+-------+
```

Существует альтернативный способ объявления первичного ключа, представленный в листинге 5.3.

Листинг 5.3. Альтернативный способ объявления первичного ключа

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

Как видно из листинга 5.3, 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 добавить в круглых скобках число символов, входящих в индекс (листинг 5.4).

Листинг 5.4. Создание первичного ключа на текстовом столбце

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL,
  name TINYTEXT NOT NULL,
  PRIMARY KEY(name(10))
)
```

Как видно из листинга 5.4, 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 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 5.5).

Листинг 5.5. Создание многостолбцового первичного ключа

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL,
  name TINYTEXT NOT NULL,
  PRIMARY KEY(id_catalog, name(10))
)
```

Просмотр структуры таблицы `catalogs` при помощи оператора `DESCRIBE` показывает, что в четвертой колонке результирующей таблицы оба поля: и `id_catalog`, и `name` имеют флаг `PRI`, который отмечает первичный ключ (листинг 5.6).

> ***Замечание***
>
> Значение полей `id_catalog` и `name` могут быть любыми, но их совокупность должна быть уникальна в пределах таблицы.

Листинг 5.6. Оператор `DESCRIBE catalogs`

```
mysql> DESCRIBE catalogs;
+------------+----------+------+-----+---------+-------+
| Field      | Type     | Null | Key | Default | Extra |
+------------+----------+------+-----+---------+-------+
| id_catalog | int(11)  | NO   | PRI |         |       |
| name       | tinytext | NO   | PRI |         |       |
+------------+----------+------+-----+---------+-------+
```

В качестве первичного ключа часто выступает столбец типа `INT` или `BIGINT`. В таблицах учебной базы данных `shop`, созданных в *главе 4*, все поля, начинающиеся с префикса `id_`, предназначены для первичного ключа. 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 5.7 приводится пример использования ключевого слова `AUTO_INCREMENT`.

Листинг 5.7. Использование атрибута `AUTO_INCREMENT`

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

Просматривая структуру вновь созданной таблицы `catalogs` при помощи оператора `DESCRIBE` (листинг 5.8), можно заметить, что для столбца `id_catalog` в шестой колонке `Extra` результирующей таблицы появилась запись `auto_increment`, сообщающая, что столбец снабжен этим атрибутом.

Листинг 5.8. Результат использования атрибута `AUTO_INCREMENT`

```
mysql> DESCRIBE catalogs;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| id_catalog | int(11)  | NO   | PRI | NULL    | auto_increment |
| name       | tinytext | NO   |     |         |                |
+------------+----------+------+-----+---------+----------------+
```

Как видно из листинга 5.8, для столбца `id_catalog` в качестве значения по умолчанию выставлено `NULL`, однако само поле не может принимать значение `NULL`. Передача `NULL` приведет к генерации уникального числа, которое и будет занесено в `id_catalog`. 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`, также не должно быть больше одного.

> ***Замечание***
>
> Значение `AUTO_INCREMENT` можно изменить и начать отсчет не с 1, а, например, с 1000. Более подробно об этом можно почитать в *главе 12*.

К атрибуту `AUTO_INCREMENT` мы еще вернемся в *главе 6* при рассмотрении оператора `INSERT` — вставки данных в таблицу.

5.2. 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 5.9 приводится оператор `CREATE TABLE`, создающий таблицу `orders` (см. листинг 4.28) из учебной базы данных `shop`.

Листинг 5.9. Создание таблицы `orders`

```
CREATE TABLE orders (
  id_order INT(11) NOT NULL AUTO_INCREMENT,
  id_user INT(11) NOT NULL DEFAULT '0',
  ordertime DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
  number INT(11) NOT NULL DEFAULT '0',
  id_product INT(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (id_order),
  KEY id_product (id_product),
  KEY id_user (id_user)
)
```

Как видно из листинга 5.9, в таблице `orders` вводятся два индекса по столбцам `id_product` и `id_user`. Имена индексов совпадают с именами индексируемых столбцов, но вполне допустимо присвоить им другие имена, например:

```
KEY first (id_product),
KEY second (id_user)
```

> ***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`.

Просмотр структуры таблицы `orders` при помощи оператора `DESCRIBE` приводит к результату, представленному в листинге 5.10.

Листинг 5.10. Оператор `DESCRIBE orders`

```
mysql> DESCRIBE orders;
+------------+----------+------+-----+------------+----------------+
| Field      | Type     | Null | Key | Default    | Extra          |
+------------+----------+------+-----+------------+----------------+
| id_order   | int(11)  | NO   | PRI | NULL       | auto_increment |
| id_user    | int(11)  | NO   | MUL | 0          |                |
| ordertime  | datetime | NO   |     | 0000-00-00 |                |
| number     | int(11)  | NO   |     | 0          |                |
| id_product | int(11)  | NO   | MUL | 0          |                |
+------------+----------+------+-----+------------+----------------+
```

Как видно из листинга 5.10, столбцы, проиндексированные обычным индексом, помечаются в поле `Key` флагом `MUL`.

> ***Замечание***
>
> Индексы, ссылающиеся на первичные ключи другой таблицы, такие как `id_user` и `id_product`, называют *внешними ключами*. Так `id_user` в таблице `orders` является внешним ключом таблицы `users`, а `id_products` — внешним ключом таблицы `products`. Подробнее с внешними ключами можно познакомиться в *главе 25*.

Объявление уникальных индексов в таблице `orders` приводит к тому, что проиндексированные столбцы помечаются флагом `UNI` (листинг 5.11).

Листинг 5.11. Оператор `DESCRIBE orders`

```
mysql> DESCRIBE orders;
+------------+----------+------+-----+------------+----------------+
| Field      | Type     | Null | Key | Default    | Extra          |
+------------+----------+------+-----+------------+----------------+
| id_order   | int(11)  | NO   | PRI | NULL       | auto_increment |
| id_user    | int(11)  | NO   | UNI | 0          |                |
| ordertime  | datetime | NO   |     | 0000-00-00 |                |
| number     | int(11)  | NO   |     | 0          |                |
| id_product | 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 5.12.

Листинг 5.12. Объявление индекса на двух столбцах

```
CREATE TABLE orders (
  id_order INT(11) NOT NULL AUTO_INCREMENT,
  id_user INT(11) NOT NULL DEFAULT '0',
  ordertime DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
  number INT(11) NOT NULL DEFAULT '0',
  id_product INT(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (id_order),
  KEY id_product (id_product, id_user)
)
```

Интересно отметить, что оператор `DESCRIBE` в этом случае будет считать проиндексированным только поле `id_product` (листинг 5.13).

Листинг 5.13. Результат объявления индекса на двух столбцах

```
mysql> DESCRIBE orders;
+------------+----------+------+-----+------------+----------------+
| Field      | Type     | Null | Key | Default    | Extra          |
+------------+----------+------+-----+------------+----------------+
| id_order   | int(11)  | NO   | PRI | NULL       | auto_increment |
| id_user    | int(11)  | NO   |     | 0          |                |
| ordertime  | datetime | NO   |     | 0000-00-00 |                |
| number     | int(11)  | NO   |     | 0          |                |
| id_product | int(11)  | NO   | MUL | 0          |                |
+------------+----------+------+-----+------------+----------------+
```

Часто решение об индексировании столбцов принимается после того, как таблица создана и заполнена данными. В этом случае прибегают к изменению структуры таблицы, создавая индекс при помощи оператора `CREATE INDEX` или удаляя его при помощи оператора `DROP INDEX`.

> ***Замечание***
>
> Для создания уникального индекса используется оператор `CREATE UNIQUE KEY`, в то же время удаление уникального индекса производится при помощи оператора `DROP INDEX` без дополнительного ключевого слова `UNIQUE`, т. к. характер индекса при удалении не имеет значения.

Замечание

Изменение структуры таблицы, в том числе и индексов, производится при помощи оператора ALTER TABLE, который рассматривается в *главе 13*. Операторы CREATE INDEX и DROP INDEX здесь являются, скорее, исключениями из правила, т. к. все остальные операции, изменение структуры индексов, создание, удаление, изменение столбцов и индексов производятся при помощи оператора ALTER TABLE.

Замечание

Создание и удаление первичного ключа при помощи операторов CREATE INDEX и DROP INDEX невозможно и осуществляется только при помощи оператора ALTER TABLE.

В листинге 5.14 демонстрируется создание индекса с именем id_catalog_index для столбца id_catalog таблицы products (см. листинг 4.26) учебной базы данных shop, рассмотренной в предыдущей главе.

Листинг 5.14. Создание индекса id_catalog_index в таблице products

```
CREATE INDEX id_catalog_index ON products (id_catalog)
```

Если необходимо создать индекс по нескольким столбцам, они перечисляются в круглых скобках через запятую.

Оператор DROP INDEX, который удаляет вновь созданный индекс, представлен в листинге 5.15.

Листинг 5.15. Удаление индекса id_catalog_index в таблице products

```
DROP INDEX id_catalog_index ON products
```

Точно так же, как и в случае первичного ключа, допускается индексирование текстовых столбцов, если указано число индексируемых символов (листинг 5.16).

Листинг 5.16. 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 5.17).

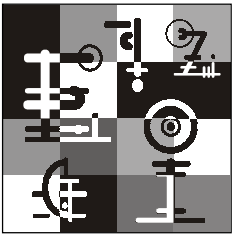
Замечание

Подробнее типы таблиц и их характеристики рассматриваются в *главе 11*. Создание индексов в таблицах, чей тип не предназначен для данного вида индекса, приведет к его игнорированию (предупреждений и сообщений об ошибке выдаваться не будет).

Листинг 5.17. Создание индексов типа `BTREE` и `HASH`

```
CREATE INDEX id_btree USING BTREE ON tbl (id)
CREATE INDEX id_hash USING HASH ON tbl (id)
```

Главное отличие этих двух видов индексов заключается в том, что `BTREE`-индексы построены на B-деревьях, а `HASH`-индексы построены на хэш-индексах.



Глава 6

Добавление данных

После успешного создания базы данных и таблиц перед разработчиком встает задача заполнения таблиц данными. Традиционно в реляционных базах данных для осуществления этой операции применяют три подхода:

- однострочный оператор `INSERT` добавляет в таблицу данных новую запись;
- многострочный оператор `INSERT` добавляет в таблицу данных несколько новых записей;
- пакетная загрузка данных служит для добавления в таблицу данных из внешнего файла.

6.1. Однострочный 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.

Данный оператор вставляет новую запись в таблицу *tbl*, значения записи перечисляются в списке *expression*, порядок следования столбцов может задаваться списком *col_name*. Значения передаются в списке после ключевого слова `VALUES`.

Рассмотрим процесс вставки на примере таблицы `catalogs` (см. листинг 4.25), которая имеет два поля:

- `id_catalog` — первичный ключ таблицы `catalogs`, снабженный атрибутом `AUTO_INCREMENT`, значение по умолчанию — `NULL`;
- `name` — название каталога, значение по умолчанию — пустая строка.

Замечание

Просмотр данных, помещенных в таблицу `catalogs`, осуществляется при помощи запроса `SELECT * FROM catalogs`. Оператор `SELECT` обсуждается подробнее в *главе 7*.

Добавить новую запись в таблицу `catalogs` можно при помощи запроса, представленного в листинге 6.1.

Листинг 6.1. Добавление новой записи в таблицу `catalogs`

```
mysql> INSERT INTO catalogs VALUES (1, 'Процессоры');
mysql> SELECT * FROM catalogs;
+------------+------------+
| id_catalog | name       |
+------------+------------+
|          1 | Процессоры |
+------------+------------+
```

Как видно из листинга 6.1, 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.

Замечание

Использование кавычек с числовыми значениями приведет к уменьшению производительности, т. к. на преобразование типа MySQL потребуется затратить дополнительное время.

Список столбцов *`col_name`*, размещенный после имени таблицы, позволяет изменить порядок следования столбцов при добавлении (листинг 6.2).

Листинг 6.2. Изменение порядка следования столбцов

```
mysql> INSERT INTO catalogs (name, id_catalog)
    -> VALUES ('Материнские платы', 2);
mysql> SELECT * FROM catalogs;
+------------+------------------+
| id_catalog | name             |
+------------+------------------+
|          1 | Процессоры       |
|          2 | Материнские платы |
+------------+------------------+
```

Как видно из листинга 6.2, порядок следования столбцов был изменен, сначала было добавлено название раздела `name` и лишь затем первичный ключ таблицы `id_catalog`. Следует помнить, что первичный ключ таблицы является уникальным значением, и добавление уже существующего значения приведет к ошибке (листинг 6.3).

Листинг 6.3. Значения первичного ключа должны быть уникальными

```
mysql> INSERT INTO catalogs VALUES (2, 'Видеоадаптеры');
ERROR 1062: Duplicate entry '2' for key 1
```

Если необходимо, чтобы новые записи с дублирующим ключом отбрасывались без генерации ошибки, необходимо добавить после оператора `INSERT` ключевое слово `IGNORE`.

Листинг 6.4. Использование ключевого слова `IGNORE`

```
mysql> INSERT IGNORE INTO catalogs VALUES (2, 'Видеоадаптеры');
mysql> SELECT * FROM catalogs;
+------------+------------------+
| id_catalog | name             |
+------------+------------------+
|          1 | Процессоры       |
|          2 | Материнские платы |
+------------+------------------+
```

Как видно из листинга 6.4, генерация ошибки не происходит, тем не менее, новая запись также не добавляется.

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 `id_catalog` атрибутом `AUTO_INCREMENT` *(см. главу 5)*. Тогда при создании новой записи в качестве значения `id_catalog` достаточно передать `NULL` или `0` — поле автоматически получит значение, равное максимальному значению столбца `id_catalog`, плюс единица (листинг 6.5).

Листинг 6.5. Действие механизма `AUTO_INCREMENT`

```
mysql> INSERT INTO catalogs VALUES (NULL, 'Видеоадаптеры');
mysql> SELECT * FROM catalogs;
+------------+------------------+
| id_catalog | name             |
```

```
+------------+------------------+
|          1 | Процессоры       |
|          2 | Материнские платы |
|          3 | Видеоадаптеры    |
+------------+------------------+
```

При вставке оператором `INSERT` не обязательно указывать все столбцы. Те столбцы, которые не указаны в списке, получат значение по умолчанию. Так, в листинге 6.6 значение `id_catalog` не передается оператору `INSERT`.

Листинг 6.6. Значение `id_catalog` отсутствует

```
mysql> INSERT INTO catalogs (name) VALUES ('Жесткие диски');
mysql> SELECT * FROM catalogs;
+------------+------------------+
| id_catalog | name             |
+------------+------------------+
|          1 | Процессоры       |
|          2 | Материнские платы |
|          3 | Видеоадаптеры    |
|          4 | Жесткие диски    |
+------------+------------------+
```

Так как `id_catalog` по умолчанию принимает значение `NULL`, которое вызывает механизм `AUTO_INCREMENT`, то поле `id_catalog` автоматически получает значение `4` (максимальное значение 3 плюс 1). Допускается добавление полностью пустых строк, как это продемонстрировано в листинге 6.7.

Листинг 6.7. Добавление пустых строк

```
mysql> INSERT INTO catalogs () VALUES ();
mysql> SELECT * FROM catalogs;
+------------+------------------+
| id_catalog | name             |
+------------+------------------+
|          1 | Процессоры       |
|          2 | Материнские платы |
|          3 | Видеоадаптеры    |
|          4 | Жесткие диски    |
|          5 |                  |
+------------+------------------+
```

Так как текстовое поле `name` по умолчанию принимает пустую строку, новая запись содержит пустую запись. Значение по умолчанию можно подставить, если передать в качестве значения ключевое слово `DEFAULT` (листинг 6.8).

Листинг 6.8. Использование ключевого слова `DEFAULT`

```
mysql> INSERT catalogs (id_catalog, name) VALUES (DEFAULT, DEFAULT);
mysql> SELECT * FROM catalogs;
+------------+------------------+
| id_catalog | name             |
+------------+------------------+
|          1 | Процессоры       |
|          2 | Материнские платы |
|          3 | Видеоадаптеры    |
|          4 | Жесткие диски    |
|          5 |                  |
|          6 |                  |
+------------+------------------+
```

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. 22.4.6)*, как это показано в листинге 6.9.

Листинг 6.9. Получение только что добавленного значения

```
mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                6 |
+------------------+
```

Например, если необходимо, чтобы следующее значение поля `id_catalog` приняло значение 16, а не 7, можно воспользоваться запросом, представленным в листинге 6.10.

> ***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 6.10. Использование функции LAST_INSERT_ID()

```
mysql> INSERT INTO catalogs (is_catalog, name)
    -> VALUES (LAST_INSERT_ID() + 10, 'Оперативная память');
mysql> SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 |                    |
|          6 |                    |
|         16 | 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.

Кроме констант, в списке значений *expression* могут выступать выражения с участием столбцов из списка *col_name*. Пример представлен в листинге 6.11.

Листинг 6.11. Использование выражений

```
mysql> INSERT catalogs (id_catalog, name) VALUES (7, id_catalog*3);
mysql> INSERT catalogs (id_catalog, name) VALUES (name*3, 7);
mysql> SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
```

```
|           4 | Жесткие диски      |
|           5 |                    |
|           6 |                    |
|          15 | Оперативная память |
|           7 | 21                 |
|          16 | 6                  |
+-------------+--------------------+
```

Так, в листинге 6.11 для первой записи поле `id_catalog` принимает значение `6`, а поле `name` значение `21` (7×3). Ссылаться в выражении можно только на столбцы, расположенные левее, ссылка на правые столбцы дает 0 (в случае столбца `id_catalog` это привело к автоматической генерации нового уникального значения `16` вместо ожидаемых `21`).

> ***Замечание***
>
> Оператор `*` является оператором умножения, более подробно операторы описываются в *главе 15*.

Если при вставке нового значения задействован механизм `AUTO_INCREMENT`, то воспользоваться вновь сгенерированным значением не удастся (листинг 6.12). Так сначала полю `id_catalog` присваивается значение `0`, и лишь после того, как запись добавлена — генерируется новое значение.

Листинг 6.12. Использование выражений совместно с `AUTO_INCREMENT`

```
mysql> INSERT catalogs (id_catalog, name) VALUES (NULL, id_catalog*3);
mysql> SELECT * FROM catalogs;
+-------------+--------------------+
| id_catalog | name                |
+-------------+--------------------+
|           1 | Процессоры         |
|           2 | Материнские платы  |
|           3 | Видеоадаптеры      |
|           4 | Жесткие диски      |
|           5 |                    |
|           6 |                    |
|          15 | Оперативная память |
|           7 | 21                 |
|          16 | 6                  |
|          17 | 0                  |
+-------------+--------------------+
```

Еще одной формой оператора INSERT является синтаксис с ключевым словом SET:

```
INSERT [IGNORE] [INTO] tbl SET col_name=expressin, ...
```

Оператор заносит в таблицу *tbl* новую запись, столбец *col_name* в котором получает значение *expression*.

Запрос в листинге 6.13 аналогичен запросу из листинга 6.14.

Листинг 6.13. Использование альтернативной формы оператора INSERT

```
mysql> INSERT INTO catalogs SET id_catalog = 18, name = 'Сетевые карты'
```

Листинг 6.14. Полная форма оператора INSERT

```
mysql> INSERT INTO catalogs VALUES(18, 'Сетевые карты');
```

Форма оператора INSERT с ключевым словом SET позволяет более гибко добавлять записи. Если имя столбца после SET не встречается, полю присваивается значение по умолчанию.

6.2. 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*. Так, в листинге 6.15 добавляется сразу пять записей при помощи одного оператора INSERT.

> ***Замечание***
>
> Так же как и в однострочной версии оператора INSERT, допускается использование ключевого слова IGNORE с целью игнорирования записей, которые имеют значения для уникальных индексов, совпадающие с одним из значений в таблице.

Листинг 6.15. Многострочный оператор INSERT

```
mysql> 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 6.16).

Листинг 6.16. Изменение состава столбцов в многострочном операторе INSERT

```
mysql> INSERT INTO catalogs (name) VALUES ('Процессоры'),
    -> ('Материнские платы'),
    -> ('Видеоадаптеры'),
    -> ('Жесткие диски'),
    -> ('Оперативная память');
```

6.3. 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 6.17 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 6.17. Использование ключевого слова DELAYED совместно с INSERT

```
mysql> INSERT DELAYED 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 *(см. главу 11)*, и ключевое слово DELAYED применяется только для операторов INSERT, в которых указаны списки значений.

6.4. Пакетная загрузка данных

Для пакетной загрузки данных из текстового файла в таблицу в СУБД MySQL предназначен специальный оператор LOAD DATA. Данный оператор работает быстрее, чем оператор INSERT, т. к. СУБД не требуется дополнительного времени на анализ синтаксиса оператора. Оператор 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.

> ***Замечание***
>
> Оператор LOAD DATA является дополнительным к оператору SELECT...INTO OUTFILE, который решает обратную задачу — формирование на основе таблицы текстового файла с данными. Синтаксис оператора SELECT ... INTO OUTFILE рассматривается подробнее в *главе 7*.

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 6.18, приводит к заполнению таблицы catalogs строками из файла catalogs.txt.

Листинг 6.18. Использование оператора LOAD DATA

```
mysql> LOAD DATA INFILE 'C:\\mysql5\\bin\\catalogs.txt'
    -> INTO TABLE catalogs;
mysql> 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 слеши необходимо экранировать, этого можно избежать, если использовать прямые слеши: '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` приведут к сообщению об ошибке (листинг 6.19). Однако применение оператора `REPLACE` осуществит полную замену содержимого таблицы `catalogs`.

Листинг 6.19. Использование ключевого слова `REPLACE`

```
mysql> LOAD DATA INFILE 'C:\\mysql5\\bin\\newcatalogs.txt'
    -> INTO TABLE catalogs;
ERROR 1062: Duplicate entry '1' for key 1
mysql> LOAD DATA INFILE 'C:\\mysql5\\bin\\newcatalogs.txt'
    -> REPLACE INTO TABLE catalogs;
mysql> 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 располагаемых на первой строке текстового файла (листинг 6.20).

Листинг 6.20. Использование конструкции `IGNORE` *`number`* `LINES`

```
mysql> 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 6.21 SQL-запрос при использовании его в операционной системе Windows может выглядеть следующим образом.

Листинг 6.21. Использование конструкции LINES TERMINATED BY

```
mysql> 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 6.22 идентичны.

Листинг 6.22. Варианты использования оператора LOAD DATA INFILE

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

Запросы, представленные в листинге 6.23, идентичны.

Листинг 6.23. Варианты использования оператора `LOAD DATA INFILE` с ключевым словом `FIELDS`

```
mysql> LOAD DATA INFILE 'C:\\mysql5\\bin\\catalogs.txt'
    -> REPLACE INTO TABLE catalogs;
mysql> 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 6.24.

Листинг 6.24. Загрузка данных из csv-файла

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

6.5. Утилита *mysqlimport*

Утилита `mysqlimport` является консольным интерфейсом для оператора `LOAD DATA INFILE`, который описывается в *разд. 6.3*. Все операции, которые позволяет осуществить оператор `LOAD DATA INFILE`, могут быть выполнены при помощи утилиты `mysqlimport`.

Порядок использования оператора `mysqlimport` следующий:

```
mysqlimport [parameters] db_name tbl1.txt [tbl2.text] [tbl3 ...]
```

Здесь *db_name* — имя базы данных, с таблицами которой будет работать утилита. После имени базы данных через пробел указываются имена текстовых файлов. Имена файлов должны совпадать с именами таблиц, в которые будет загружена информация из файлов. При этом расширение не имеет значения. Так файлы с именами *tbl.txt*, *tbl.text* и *tbl* все будут импортированы в таблицу `tbl`.

Как любая другая клиентская программа, утилита `mysqlimport` принимает общие для всех консольных клиентов параметры из табл. 3.2, связанные аутентификацией и параметрами соединения с сервером MySQL, а также ряд дополнительных параметров, представленных в табл. 6.1.

Таблица 6.1. *Параметры утилиты* `mysqlimport`

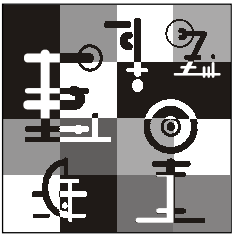
Параметр	Описание
`--columns=`*`col_list`*, `-c` *`col_list`*	Параметр задает список столбцов *`col_list`*, разделенных запятыми. Порядок столбцов определяет соответствие между столбцами таблицы и файла данных
`--delete`, `-D`	Использование данного параметра приводит к очистке таблицы перед импортом данных из файла
`--fields-terminated-by=...`, `--fields-enclosed-by=...`, `--fields-optionally-enclosed-by=...`, `--fields-escaped-by=...`, `--lines-terminated-by=...`	Параметры этой группы имеют то же значение, что и соответствующие ключевые слова оператора `LOAD DATA INFILE`, синтаксис которого рассматривается в *разд. 6.3*
`--force`, `-f`	Использование данного параметра приводит к игнорированию ошибок. Например, если таблица для одного из текстовых файлов не существует, утилита продолжит обрабатывать остальные файлы, вместо того, чтобы аварийно завершить работу
`--ignore`, `-i`	Данный параметр управляет поведением утилиты при загрузке записей, которые дублируют существующие уникальные ключи в таблице. Параметр аналогичен ключевому слову `IGNORE` в операторе `LOAD DATA INFILE` и приводит к игнорированию новых записей, которые дублируют старые
`--ignore-line=`*`n`*	Использование данного параметра приводит к игнорированию *`n`* первых строк текстового файла. Параметр аналогичен по действию ключевому слову `IGNORE` *`number`* `LINES` оператора `LOAD DATA INFILE`
`--local`, `-L`	Использование данного параметра приводит к поиску текстовых файлов на локальной машине клиента, а не сервера. Параметр аналогичен по действию ключевому слову `LOCAL` оператора `LOAD DATA INFILE`
`--lock-tables`, `-l`	Использование данного параметра приводит к блокированию всех таблиц базы данных перед началом импорта информации из тестовых файлов. Блокировка исключает нарушение ссылочной целостности базы данных, за счет обращения других клиентов к обновляемым таблицам

Таблица 6.1 *(окончание)*

Параметр	Описание
`--replace, -r`	Данный параметр управляет поведением утилиты при загрузке записей, которые дублируют существующие уникальные ключи в таблице. Параметр аналогичен ключевому слову `REPLACE` в операторе `LOAD DATA INFILE` и приводит к перезаписыванию старых записей новыми, если имеет место дублирование уникальных индексов

Используя утилиту `mysqlimport`, оператор `LOAD DATA INFILE` из листинга 6.23 можно переписать следующим образом:

```
mysqlimport -r --fields-terminated-by=','
               --fields-enclosed-by='"'
               --lines-terminated-by='\r\n' shop catalogs.txt
```



Глава 7

Выборка данных

В данной главе описывается оператор `SELECT`, позволяющий извлекать данные из таблиц базы данных. Упрощенный синтаксис оператора `SELECT` выглядит следующим образом:

```
SELECT select_expr FROM tbl
```

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*. Более того, как будет показано далее, результаты запроса, выполненного при помощи оператора `SELECT`, могут быть использованы для создания новой таблицы. Если результаты двух запросов к разным таблицам имеют одинаковый формат, их можно объединить в одну таблицу.

> ***Замечание***
>
> Таблица, полученная в результате запроса, может стать предметом дальнейших запросов. Техника создания таких запросов рассматривается в *главе 24*.

7.1. Изменение количества и порядка следования столбцов

Рассмотрим запросы на выборку данных на примере таблицы `catalogs` (см. листинг 4.25). Таблица имеет два поля: первичный ключ `id_catalog` и название элемента каталога `name`. Выбрать все записи таблицы `catalogs` можно при помощи запроса, представленного в листинге 7.1.

Листинг 7.1. Выборка данных при помощи оператора `SELECT`

```
mysql> 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 7.2).

Листинг 7.2. Использование символа * (все столбцы)

```
mysql> SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры         |
+------------+--------------------+
mysql> SELECT * FROM orders;
+----------+---------+---------------------+--------+------------+
| id_order | id_user | ordertime           | number | id_product |
+----------+---------+---------------------+--------+------------+
|        1 |       3 | 2005-01-04 10:39:38 |      1 |          8 |
|        2 |       6 | 2005-02-10 09:40:29 |      2 |         10 |
|        3 |       1 | 2005-02-18 13:41:05 |      4 |         20 |
|        4 |       3 | 2005-03-10 18:20:00 |      1 |         20 |
|        5 |       3 | 2005-03-17 19:15:36 |      1 |         20 |
+----------+---------+---------------------+--------+------------+
```

К списку столбцов в операторе SELECT прибегают в том случае, если необходимо изменить порядок следования столбов в результирующей таблице или выбрать только часть столбцов (листинг 7.3).

Листинг 7.3. Изменение числа и порядка выводимых столбцов

```
mysql> SELECT name, id_catalog FROM catalogs;
+--------------------+------------+
| name               | id_catalog |
+--------------------+------------+
| Видеоадаптеры      |          3 |
| Жесткие диски      |          4 |
| Материнские платы  |          2 |
| Оперативная память |          5 |
| Процессоры         |          1 |
+--------------------+------------+
mysql> SELECT id_order, id_user, ordertime name FROM orders;
+----------+---------+---------------------+
| id_order | id_user | name                |
+----------+---------+---------------------+
|        1 |       3 | 2005-01-04 10:39:38 |
|        2 |       6 | 2005-02-10 09:40:29 |
|        3 |       1 | 2005-02-18 13:41:05 |
|        4 |       3 | 2005-03-10 18:20:00 |
|        5 |       3 | 2005-03-17 19:15:36 |
+----------+---------+---------------------+
```

В листинге 7.3 в запросе к таблице catalogs изменяется порядок следования столбцов, а в запросе к таблице orders из пяти столбцов таблицы извлекаются только три.

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 7.4).

Листинг 7.4. Использование констант в списке столбцов

```
mysql> 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 |
+--------------------+-------------+---+----------+
```

7.2. Условия

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.

Так, в листинге 7.5 приводится пример запроса, извлекающего из таблицы `catalogs` записи, чей первичный ключ `id_catalog` больше (оператор >) 2.

Листинг 7.5. Использование конструкции `WHERE`

```
mysql> SELECT * FROM catalogs WHERE id_catalog > 2;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
```

Условие может быть составным и объединяться при помощи логических операторов. В листинге 7.6 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). Подробнее логические операторы описаны в *главе 15*.

Листинг 7.6. Использование составного логического условия

```
mysql> SELECT * FROM catalogs
    -> WHERE id_catalog > 2 AND id_catalog <= 4;
```

```
+------------+---------------+
| id_catalog | name          |
+------------+---------------+
|          3 | Видеоадаптеры |
|          4 | Жесткие диски |
+------------+---------------+
```

Как видно из листинга 7.6, в результирующей таблице возвращаются записи в диапазоне от 3 до 4. Такого же эффекта можно добиться применением конструкции `BETWEEN` *`min`* `AND` *`max`*, возвращающей записи, значения столбца `id_catalog` для которых лежат в диапазоне от *`min`* до *`max`* (листинг 7.7).

Листинг 7.7. Использование конструкции `BETWEEN`

```
mysql> SELECT * FROM catalogs WHERE id_catalog BETWEEN 3 AND 4;
+------------+---------------+
| id_catalog | name          |
+------------+---------------+
|          3 | Видеоадаптеры |
|          4 | Жесткие диски |
+------------+---------------+
```

Существует конструкция, противоположная конструкции `BETWEEN` — `NOT BETWEEN`, которая возвращает записи, не попадающие в интервал между *`min`* и *`max`* (листинг 7.8).

Листинг 7.8. Использование конструкции `NOT BETWEEN`

```
mysql> SELECT * FROM catalogs WHERE id_catalog NOT BETWEEN 3 AND 4;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры         |
+------------+--------------------+
```

Иногда требуется извлечь записи, удовлетворяющие не диапазону, а списку, например, записи с `id_catalog` из списка `(1,2,5)`, как показано в листинге 7.9. Для этого предназначена конструкция `IN`.

Листинг 7.9. Использование оператора IN

```
mysql> SELECT * FROM catalogs WHERE id_catalog IN (1,2,5);
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры         |
+------------+--------------------+
```

Замечание

По аналогии с конструкцией BETWEEN, для IN существует обратная конструкция NOT IN, которая позволяет извлечь из таблицы записи, не удовлетворяющие списку. Подробнее операторы рассматриваются в *главе 15*.

В конструкции WHERE могут использоваться не только числовые столбцы. В листинге 7.10 из таблицы catalogs извлекается запись, соответствующая элементу каталога "Процессоры".

Листинг 7.10. Работа с текстовыми полями

```
mysql> SELECT * FROM catalogs WHERE name = 'процессоры';
+------------+------------+
| id_catalog | name       |
+------------+------------+
|          1 | Процессоры |
+------------+------------+
```

Как видно из листинга 7.10, 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 *главе 14*.

Замечание

Для работы с текстом СУБД MySQL предлагает широкие возможности, которые рассматриваются в *главах 17*, *19* и *20*.

В листинге 7.11 из таблицы orders извлекаются записи, соответствующие покупкам, осуществленным за февраль 2005 года.

Листинг 7.11. Работа с датой

```
mysql> SELECT * FROM orders
    -> WHERE ordertime >= '2005-02-01' AND ordertime < '2005-03-01';
+----------+---------+---------------------+--------+------------+
| id_order | id_user | ordertime           | number | id_product |
+----------+---------+---------------------+--------+------------+
|        2 |       6 | 2005-02-10 09:40:29 |      2 |         10 |
|        3 |       1 | 2005-02-18 13:41:05 |      4 |         20 |
+----------+---------+---------------------+--------+------------+
```

В качестве логических выражений могут выступать и константы. Любое число, отличное от нуля, считается истинным выражением, т. е. удовлетворяющим запросу, а 0 — ложью, т. е. не удовлетворяющим запросу.

Пример представлен в листинге 7.12.

Листинг 7.12. Использование констант в качестве логических выражений

```
mysql> SELECT * FROM catalogs WHERE 1;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры         |
+------------+--------------------+
mysql> SELECT * FROM catalogs WHERE 0;
Empty set (0.00 sec)
```

Как видно, если первый запрос аналогичен `SELECT * FROM catalogs`, то второй запрос всегда возвращает пустую таблицу.

7.3. Сортировка

Как видно из листингов, 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, по которому следует сортировать данные.

Как видно из листинга 7.13, первый запрос сортирует результат выборки по полю `id_catalog`, а второй — по полю `name`.

Листинг 7.13. Использование конструкции `ORDER BY`

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

Сортировку записей можно производить и по нескольким столбцам. Пусть требуется извлечь из таблицы `products` записи, соответствующие товарным позициям, количество которых на складе `count` от 4 до 8, с сортировкой по полю `count`. Для краткости выведем только столбцы `count` и `mark` (оценка товарной позиции).

Листинг 7.14. Вывод столбцов `count` и `mark`

```
mysql> SELECT count, mark FROM products
    -> WHERE count BETWEEN 4 AND 8 ORDER BY count;
+-------+------+
| count | mark |
+-------+------+
|     4 |  3.9 |
|     4 |  4.0 |
```

```
|     4 |  3.9 |
|     4 |  4.1 |
|     5 |  4.0 |
|     5 |  4.0 |
|     5 |  4.5 |
|     6 |  3.7 |
|     6 |  4.1 |
|     6 |  4.1 |
|     6 |  4.5 |
|     6 |  3.6 |
|     6 |  4.0 |
|     6 |  4.0 |
|     8 |  4.2 |
|     8 |  4.5 |
+-------+------+
```

Как видно из листинга 7.14, сортировка по полю `count` прошла успешно, однако оценки `mark` во втором столбце расположены в хаотическом порядке. Для сортировки результатов сразу по двум столбцам их следует указать сразу после ключевого слова `ORDER BY` через запятую (листинг 7.15). Записи сортируются по полю `count` и, если встречается несколько записей с совпадающим значением в поле `count`, они сортируются по полю `mark`. Число столбцов, следующих за конструкцией `ORDER BY`, не ограничено.

Листинг 7.15. Сортировка по двум полям

```
mysql> SELECT count, mark FROM products
    -> WHERE count BETWEEN 4 AND 8 ORDER BY count, mark;
+-------+------+
| count | mark |
+-------+------+
|     4 |  3.9 |
|     4 |  3.9 |
|     4 |  4.0 |
|     4 |  4.1 |
|     5 |  4.0 |
|     5 |  4.0 |
|     5 |  4.5 |
|     6 |  3.6 |
|     6 |  3.7 |
```

```
|      6 |  4.0 |
|      6 |  4.0 |
|      6 |  4.1 |
|      6 |  4.1 |
|      6 |  4.5 |
|      8 |  4.2 |
|      8 |  4.5 |
+-------+------+
```

По умолчанию сортировка производится в прямом порядке, записи располагаются, начиная с наименьшего значения и заканчивая наибольшим (листинг 7.16).

Листинг 7.16. Сортировка по умолчанию

```
mysql> SELECT ordertime FROM orders ORDER BY ordertime;
+---------------------+
| ordertime           |
+---------------------+
| 2005-01-04 10:39:38 |
| 2005-02-10 09:40:29 |
| 2005-02-18 13:41:05 |
| 2005-03-10 18:20:00 |
| 2005-03-17 19:15:36 |
+---------------------+
```

Изменить порядок сортировки на обратный можно при помощи ключевого слова `DESC` (листинг 7.17).

Листинг 7.17. Обратная сортировка

```
mysql> SELECT ordertime FROM orders ORDER BY ordertime DESC;
+---------------------+
| ordertime           |
+---------------------+
| 2005-03-17 19:15:36 |
| 2005-03-10 18:20:00 |
| 2005-02-18 13:41:05 |
| 2005-02-10 09:40:29 |
| 2005-01-04 10:39:38 |
+---------------------+
```

Для прямой сортировки также существует ключевое слово ASC, но поскольку по умолчанию записи сортируются в прямом порядке, данное ключевое слово часто опускают.

7.4. 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 7.18 извлекаются первые пять записей, при этом одновременно осуществляется обратная сортировка по полю count.

Листинг 7.18. Использование ключевого слова LIMIT

```
mysql> SELECT id_product, count FROM products
    -> ORDER BY count DESC
    -> LIMIT 5;
+------------+-------+
| id_product | count |
+------------+-------+
|         28 |    20 |
|         25 |    20 |
|         26 |    15 |
|         29 |    12 |
|          9 |    12 |
+------------+-------+
```

Для того чтобы извлечь следующие пять записей, используется ключевое слово LIMIT с двумя цифрами, первая с двумя числами, первое указывает позицию, начиная с которой необходимо вернуть результат, а второе число — это количество извлекаемых записей (листинг 7.19).

Листинг 7.19. Извлечение записей, начиная с позиции 5

```
mysql> SELECT id_product, count FROM products
    -> ORDER BY count DESC
    -> LIMIT 5, 5;
```

```
+------------+-------+
| id_product | count |
+------------+-------+
|          1 |    10 |
|         27 |    10 |
|         24 |     8 |
|         30 |     8 |
|         20 |     6 |
+------------+-------+
```

Для извлечения следующих 5 записей необходимо использовать конструкцию `LIMIT 10, 5`.

7.5. Использование функций

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 функции 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 вместо вызова функции.

В листинге 7.20 приведен пример использования функции `VERSION()`, которая возвращает версию сервера MySQL.

Листинг 7.20. Использование функции `VERSION()`

```
mysql> SELECT VERSION();
+-----------------+
| VERSION()       |
+-----------------+
| 5.0.18-standard |
+-----------------+
```

Как видно из листинга 7.20, для использования функции `VERSION()` не требуется применение ключевого слова `FROM`, т. к. параметры таблицы не требуются. Однако использование его не возбраняется (листинг 7.21).

Листинг 7.21. Альтернативное использование функции VERSION()

```
mysql> 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 (см. листинг 7.4) 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 7.22).

Замечание

Таблица DUAL введена в MySQL, начиная с версии 4.1.

Листинг 7.22. Использование псевдотаблицы DUAL

```
mysql> SELECT VERSION(), 2+2 FROM DUAL;
+-----------------+-----+
| VERSION()       | 2+2 |
+-----------------+-----+
| 5.0.18-standard |   4 |
+-----------------+-----+
```

Чаще функция принимает параметры, например, имена столбцов. Функция COUNT() возвращает число записей в таблице и принимает в качестве аргумента имя столбца. Функция возвращает число строк в таблице, значения столбца для которых отличны от NULL (листинг 7.23).

Листинг 7.23. Использование функции COUNT()

```
mysql> SELECT COUNT(id_order) FROM orders;
+----------------+
| COUNT(id_order) |
```

```
+-----------------+
|               5 |
+-----------------+
mysql> SELECT COUNT(id_product) FROM products;
+-------------------+
| COUNT(id_product) |
+-------------------+
|                30 |
+-------------------+
```

В качестве параметра функции наряду с именами столбцов может выступать символ звездочки (*). При использовании символа * будет возвращено число строк таблицы не зависимо от того, принимают какие-то из них значение NULL или нет. Одной из особенностей использования функций является то, что название столбца в результирующей таблице совпадает с названием функции и ее параметрами. Часто это не удобно, особенно в прикладных программах, где после выполнения запроса обращение к результату происходит по имени столбца. В SELECT-запросе столбцу можно назначить новое имя, для этого предназначен оператор AS. В листинге 7.24 результату функции COUNT() присваивается новый псевдоним total.

Листинг 7.24. Использование оператора AS

```
mysql> SELECT COUNT(id_order) AS total FROM orders;
+-------+
| total |
+-------+
|     5 |
+-------+
```

Новое имя можно использовать в других частях SQL-запроса, в выражениях WHERE и ORDER BY. Данный прием часто используется при выполнении многотабличных запросов, рассмотренных в *главе 8*.

В качестве дополнительного примера можно привести функции MIN() и MAX(), возвращающие минимальное и максимальное значения столбца, имя которого передано в качестве параметра (листинг 7.25).

Листинг 7.25. Использование функций MIN() и MAX()

```
mysql> 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 7.26 показана попытка извлечения записи из таблицы `catalogs` с максимальным значением поля `id_catalog`.

Листинг 7.26. Использование функций `MIN()` и `MAX()` в `WHERE` недопустимо

```
mysql> 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 7.27 первый запрос извлекает запись с наименьшим значением поля `id_catalog`, а второй — с наибольшим.

Листинг 7.27. Использование функций `MIN()` и `MAX()` совместно с `ORDER BY`

```
mysql> SELECT * FROM catalogs ORDER BY id_catalog LIMIT 1;
+------------+------------+
| id_catalog | name       |
+------------+------------+
|          1 | Процессоры |
+------------+------------+
mysql> SELECT * FROM catalogs ORDER BY id_catalog DESC LIMIT 1;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          5 | Оперативная память |
+------------+--------------------+
```

СУБД MySQL имеет большое число встроенных функций, подробному их рассмотрению посвящена *часть II*.

7.6. Группировка записей

Пусть необходимо определить первичные ключи, соответствующие элементам каталога `id_catalog`, в которых есть хоть одна товарная позиция. Для этого достаточно осуществить запрос к таблице `products`, который извлечет поле `id_catalog` (листинг 7.28).

Листинг 7.28. Извлечение первичных ключей, соответствующих элементам каталога

```
mysql> SELECT id_catalog FROM products ORDER BY id_catalog;
+------------+
| id_catalog |
+------------+
|          1 |
|          1 |
|          1 |
|          1 |
|          1 |
|          1 |
|          1 |
|          1 |
|          1 |
|          2 |
|          2 |
|          2 |
|          2 |
|          2 |
|          2 |
|          3 |
|          3 |
|          3 |
|          3 |
|          4 |
|          4 |
|          4 |
|          4 |
|          4 |
|          5 |
|          5 |
|          5 |
|          5 |
|          5 |
|          5 |
+------------+
```

Как видно из листинга 7.28, 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 7.29).

Замечание

Ключевое слово `DISTINCT` имеет синоним — `DISTINCTROW`.

Листинг 7.29. Выборка уникальных значений

```
mysql> SELECT DISTINCT id_catalog FROM products ORDER BY id_catalog;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
```

Как показано в листинге 7.29, результат запроса не содержит ни одного повторяющегося значения. Использование ключевого слова `DISTINCT` допускается совместно с функцией `COUNT()`. В листинге 7.30 первый запрос возвращает общее число записей в таблице `products`, а второй — число уникальных значений `id_catalog`.

Листинг 7.30. Совместное использование ключевого слова `DISTINCT` и функции `COUNT()`

```
mysql> SELECT COUNT(id_catalog) FROM products;
+------------------+
| COUNT(id_catalog) |
+------------------+
|                30 |
+------------------+
mysql> SELECT COUNT(DISTINCT id_catalog) FROM products;
+---------------------------+
| COUNT(DISTINCT id_catalog) |
+---------------------------+
|                         5 |
+---------------------------+
```

Для ключевого слова DISTINCT имеется противоположное 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 7.31).

> ***Замечание***
>
> Конструкция GROUP BY располагается в SELECT-запросе перед конструкциями ORDER BY и LIMIT.

Листинг 7.31. Использование конструкции GROUP BY

```
mysql> SELECT id_catalog FROM products
    -> 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 7.32).

Листинг 7.32. Совместное использование GROUP BY и функции COUNT()

```
mysql> SELECT id_catalog, COUNT(id_catalog) FROM products
    -> GROUP BY id_catalog ORDER BY id_catalog;
+------------+-------------------+
| id_catalog | COUNT(id_catalog) |
+------------+-------------------+
|          1 |                 9 |
|          2 |                 6 |
|          3 |                 4 |
|          4 |                 5 |
|          5 |                 6 |
+------------+-------------------+
```

Как видно из листинга 7.32, каталог с первичным ключом `id_catalog = 1` содержит 9 товарных позиций, с `id_catalog = 2` — 6 и т. д.

> ***Замечание***
>
> Помимо функции `COUNT()`, в выражениях с использованием конструкции `GROUP BY` может применяться целый класс функций, описанию которых посвящена *глава 21*.

При использовании конструкции `GROUP BY` точно так же возможно применение условия `WHERE` (листинг 7.33).

Листинг 7.33. Использование ключевого слова `WHERE` совместно с `GROUP BY`

```
mysql> SELECT id_catalog, COUNT(id_catalog) FROM products
    -> WHERE id_catalog > 2
    -> GROUP BY id_catalog
    -> ORDER BY id_catalog;
+------------+------------------+
| id_catalog | COUNT(id_catalog) |
+------------+------------------+
|          3 |                4 |
|          4 |                5 |
|          5 |                6 |
+------------+------------------+
```

Чаще при составлении условий требуется ограничить выборку по результату функции, например, выбрать каталоги, где число товарных позиций больше пяти. Использование для этих целей конструкции `WHERE` приводит к ошибке.

Листинг 7.34. Ошибочное использование конструкции `WHERE`

```
mysql> SELECT id_catalog, COUNT(id_catalog) AS total FROM products
    -> WHERE total > 5
    -> GROUP BY id_catalog
    -> ORDER BY id_catalog;
ERROR 1054: 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 7.35).

Листинг 7.35. Использование ключевого слова `GROUP BY`

```
mysql> SELECT id_catalog, COUNT(id_catalog) AS total FROM products
    -> GROUP BY id_catalog
    -> HAVING total > 5
    -> ORDER BY id_catalog;
+------------+-------+
| id_catalog | total |
+------------+-------+
|          1 |     9 |
|          2 |     6 |
|          5 |     6 |
+------------+-------+
```

В условии `HAVING` можно использовать все столбцы результирующей таблицы, не только вычисляемые, например, в листинге 7.36 приводится пример запроса, извлекающего уникальные значения столбца `id_catalog`, большие двух.

Листинг 7.36. Использование в конструкции `HAVING` невычисляемых столбцов

```
mysql> 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 7.33 и 7.36, 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 7.37).

Листинг 7.37. Использование ключевого слова HAVING без GROUP BY

```
mysql> SELECT id_catalog FROM products
    -> HAVING id_catalog > 2
    -> ORDER BY id_catalog;
+------------+
| id_catalog |
+------------+
|          3 |
|          3 |
|          3 |
|          3 |
|          4 |
|          4 |
|          4 |
|          4 |
|          4 |
|          5 |
|          5 |
|          5 |
|          5 |
|          5 |
|          5 |
+------------+
```

В этом случае каждая строка таблицы рассматривается как отдельная группа.

7.7. 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 запроса, представленные в листинге 7.38.

Листинг 7.38. Одиночные SELECT-запросы

```
mysql> SELECT id_catalog FROM catalogs;
+------------+
| id_catalog |
+------------+
```

```
|           1 |
|           2 |
|           3 |
|           4 |
|           5 |
+-------------+
mysql> SELECT id_order + 5 FROM orders;
+--------------+
| 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 7.39.

Листинг 7.39. Использование ключевого слова `UNION`

```
mysql> SELECT id_catalog FROM catalogs
    -> UNION
    -> SELECT id_order + 5 FROM orders;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
|          6 |
|          7 |
|          8 |
|          9 |
|         10 |
+------------+
```

Во втором запросе SELECT, производящем выборку из таблицы orders, к значению первичного ключа id_orders добавляется значение 5, таким образом, все значения в результирующей таблице становятся уникальными. Однако если результирующая таблица содержит повторяющиеся строки, СУБД MySQL автоматически отбрасывает дубликаты (листинг 7.40).

Листинг 7.40. Исключение дублирующих записей при использовании конструкции UNION

```
mysql> SELECT id_catalog FROM catalogs
    -> UNION
    -> SELECT id_order FROM orders;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
```

Как видно из листинга 7.40, 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 7.41).

Листинг 7.41. Использование ключевого слова UNION ALL

```
mysql> SELECT id_catalog FROM catalogs
    -> UNION ALL
    -> SELECT id_order FROM orders;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
```

```
|           1 |
|           2 |
|           3 |
|           4 |
|           5 |
+-------------+
```

Для формы `UNION ALL` существует противоположный оператор `UNION DISTINCT`, но т. к. оператор `UNION` по умолчанию отбрасывает не уникальные столбцы, 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 7.42).

Листинг 7.42. Ошибочное использование конструкции `ORDER BY`

```
mysql> SELECT id_catalog FROM catalogs ORDER BY id_catalog
    -> UNION ALL
    -> SELECT id_order FROM orders;
ERROR 1221: Incorrect usage of UNION and ORDER BY
```

Это связано с тем, что сортировка производится только для объединенной таблицы и значение имеет лишь последняя конструкция `ORDER BY` (листинг 7.43).

Листинг 7.43. Сортировка результирующей таблицы

```
mysql> SELECT id_catalog FROM catalogs
    -> UNION ALL
    -> SELECT id_order FROM orders
    -> 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 7.43, объединению подвергаются данные из двух совершенно разных таблиц: catalogs и orders. Однако обращение к столбцу id_order приводит к ошибке (листинг 7.44) — столбца с таким именем в таблице просто нет, MySQL ориентируется на столбцы первого SELECT-запроса.

Листинг 7.44. Ошибочное обращение к столбцу id_order в конструкции ORDER BY

```
mysql> SELECT id_catalog FROM catalogs
    -> UNION ALL
    -> SELECT id_order FROM orders
    -> ORDER BY id_order;
ERROR 1054: Unknown column 'id_order' in 'order clause'
```

На том факте, что MySQL изменяет названия столбцов, но оставляет неизменным порядок их следования в UNION-запросах, основано большинство взломов баз данных при помощи SQL-инъекций. Эта проблема подробнее обсуждается в *главах 37—39*.

В данном разделе рассматривались в основном UNION-запросы, объединяющие два SELECT-запроса. Однако это вовсе не значит, что оператор UNION не может объединять больше запросов.

Листинг 7.45. Объединение четырех таблиц

```
mysql> SELECT id_catalog FROM catalogs
    -> UNION
    -> SELECT id_order FROM orders
    -> UNION
    -> SELECT id_catalog FROM products
    -> UNION
    -> SELECT id_user FROM users;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
```

```
|           3 |
|           4 |
|           5 |
|           6 |
+-------------+
```

Каждый из приведенных SELECT-запросов может использовать любые допустимые конструкции кроме конструкции ORDER BY (листинг 7.46).

Листинг 7.46. Объединение четырех таблиц

```
mysql> SELECT id_catalog FROM catalogs WHERE id_catalog != 4
    -> UNION
    -> SELECT id_order FROM orders WHERE id_order != 4
    -> UNION
    -> SELECT id_product FROM products WHERE id_product != 4 LIMIT 10
    -> UNION
    -> SELECT id_user FROM users WHERE id_user != 4;
+-------------+
| id_catalog |
+-------------+
|           1 |
|           2 |
|           3 |
|           5 |
|           6 |
|           7 |
|           8 |
|           9 |
|          10 |
|          11 |
+-------------+
```

7.8. Сохранение результатов во внешний файл

Результирующую таблицу, получаемую после выполнения оператора SELECT, можно сохранить в текстовый файл. Это позволяет более внимательно изучить результаты, подвергнуть их дальнейшей обработке при помощи внешней программы или вставить их в таблицу посредством оператора LOAD DATA INFILE, который описан в *разд. 6.3*.

Для сохранения результатов в текстовом файле предназначен оператор SELECT в форме SELECT...INTO OUTFILE *'file_name'*. Эта разновидность оператора осуществляет запись выбранных строк в файл, указанный в *file_name*.

Выполнение SQL-запроса, показанного в листинге 7.47, приведет к созданию файла text.sql в каталоге данных (в Windows — C:\mysql5\data\shop).

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

Листинг 7.47. Сохранение результирующей таблицы во внешнем файле text.sql

```
mysql> 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 7.48.

Листинг 7.48. Изменение формата внешнего файла при помощи LINES

```
mysql> 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 слеша `\`.

> ***Замечание***
>
> Конструкция `FIELDS` размещается перед конструкцией `LINES`.

Рассмотрим запрос из листинга 7.49.

Листинг 7.49. Изменение формата внешнего файла при помощи `FIELDS`

```
mysql> 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 7.49 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"
```

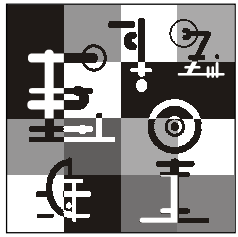
Часто удобно формировать текстовый файл в виде инструкций SQL, как это делает утилита `mysqldump` *(см. разд. 3.2)*. В листинге 7.50 приведен запрос, создающий текстовый файл, каждая строка в котором оформлена в виде запроса `INSERT`.

Листинг 7.50. Создание дампа таблицы

```
mysql> 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 7.50, 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");
```



Глава 8

Многотабличные запросы

Рассмотренные до этой главы SQL-запросы использовали одну таблицу. Но в реальных приложениях часто требуется осуществлять выборку сразу из нескольких таблиц базы данных. Запросы, которые обращаются одновременно к нескольким таблицам, называются *многотабличными запросами*. Именно в таких запросах проявляется одно из преимуществ реляционных баз данных — связь таблиц друг с другом.

8.1. Перекрестное объединение таблиц

Для обсуждения многотабличных запросов создадим две таблицы — `tbl1` и `tbl2`, каждая из которых будет содержать два столбца: числовой и текстовый. В листинге 8.1 представлены операторы `CREATE TABLE`, создающие указанные таблицы.

Листинг 8.1. 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, 'A');
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 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 8.2 представлена выборка из каждой таблицы, а также двухтабличный запрос с использованием оператора `SELECT` к таблицам `tbl1`, `tbl2`.

Листинг 8.2. Обычные и двухтабличный `SELECT`-запросы к таблицам `tbl1` и `tbl2`

```
mysql> SELECT * FROM tbl1;
+----+------+
| id | name |
+----+------+
|  1 | A    |
|  2 | Б    |
|  3 | B    |
+----+------+
mysql> SELECT * FROM tbl2;
+----+--------+
| id | letter |
+----+--------+
|  2 | Г      |
|  3 | Д      |
|  4 | E      |
+----+--------+
mysql> SELECT * FROM tbl1, tbl2;
+----+------+----+--------+
| id | name | id | letter |
+----+------+----+--------+
|  1 | A    |  2 | Г      |
|  2 | Б    |  2 | Г      |
|  3 | B    |  2 | Г      |
|  1 | A    |  3 | Д      |
|  2 | Б    |  3 | Д      |
|  3 | B    |  3 | Д      |
|  1 | A    |  4 | E      |
|  2 | Б    |  4 | E      |
|  3 | B    |  4 | E      |
+----+------+----+--------+
```

Двухтабличный запрос, представленный в листинге 8.2, 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 8.2 представлены на рис. 8.1.

SELECT * FROM tbl2	
id	letter
2	Г
3	Д
4	Е

×

SELECT * FROM tbl1	
id	name
1	А
2	Б
3	В

SELECT * FROM tbl1, tbl2			
tbl1.id	tbl1.name	tbl2.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

Рис. 8.1. Схематичное представление листинга 8.2

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 8.3).

Листинг 8.3. Управление столбцами результирующей таблицы

```
mysql> 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 8.4).

Листинг 8.4. Неоднозначность поля `id`

```
mysql> 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 8.4, прибегают к полным именам столбцов. Полное имя включает имена таблицы и столбца, разделенные точкой (листинг 8.5).

Листинг 8.5. Использование полных имен

```
mysql> SELECT tbl1.id, tbl1.name, tbl2.letter FROM tbl1, tbl2;
+----+------+--------+
| id | name | letter |
+----+------+--------+
|  1 | А    | Г      |
|  2 | Б    | Г      |
|  3 | В    | Г      |
|  1 | А    | Д      |
|  2 | Б    | Д      |
|  3 | В    | Д      |
|  1 | А    | 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 8.6).

Листинг 8.6. Использование полного имени совместно с символом `*`

```
mysql> SELECT tbl1.*, tbl2.* 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      |
+----+------+----+--------+
mysql> SELECT tbl1.* FROM tbl1, tbl2;
+----+------+
| id | name |
+----+------+
|  1 | А    |
|  2 | Б    |
|  3 | В    |
|  1 | А    |
|  2 | Б    |
|  3 | В    |
|  1 | А    |
|  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 8.7 производится выборка из таблиц `tbl1` и `tbl2` при условии совпадения полей `id` этих таблиц.

Листинг 8.7. Ограничение числа строк при помощи условия `WHERE`

```
mysql> 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 8.8).

Листинг 8.8. Ограничение числа строк при помощи конструкции `GROUP BY`

```
mysql> 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 8.9 полному имени `tbl1.id` назначается псевдоним `id`, имени `tbl1.name` — `name`, а `tbl2.letter` — `letter`.

Листинг 8.9. Назначение псевдонимов

```
mysql> 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 8.10 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 8.10. Назначение псевдонимов таблицам

```
mysql> 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.

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 8.11).

Листинг 8.11. Многотабличный запрос к одной таблице

```
mysql> SELECT t1.id, t2.name FROM tbl1 AS t1, tbl1 AS t2
    -> WHERE t1.id = t2.id;
+----+------+
| id | name |
+----+------+
|  1 | A    |
```

```
|  2 | Б    |
|  3 | В    |
+----+------+
```

Рассмотрим более реальный пример. Пусть требуется вывести названия и цены всех товарных позиций каталога из таблицы `catalogs`, которому принадлежит товарная позиция "Maxtor 6Y120P0". То есть, обнаружив в нашем магазине один жесткий диск, мы хотим выяснить, какие еще жесткие диски имеются в продаже, и сравнить их по цене. Данную задачу решает запрос, представленный в листинге 8.12.

Листинг 8.12. Список жестких дисков

```
mysql> SELECT products.name, products.price
    -> FROM products, products AS prd
    -> WHERE products.id_catalog = prd.id_catalog AND
    ->       prd.name = 'Maxtor 6Y120P0';
+-----------------------------+---------+
| name                        | price   |
+-----------------------------+---------+
| Maxtor 6Y120P0              | 2456.00 |
| Maxtor 6B200P0              | 3589.00 |
| Samsung SP0812C             | 2093.00 |
| Seagate Barracuda ST3160023A | 3139.00 |
| Seagate ST3120026A          | 2468.00 |
+-----------------------------+---------+
```

В листинге 8.12 происходит самообъединение таблицы `products`, при этом в полных именах списка столбцов после ключевого слова `SELECT` следует использовать имя таблицы, не участвующей в сравнении со строкой "Maxtor 6Y120P0", иначе будет возвращено пять строк с данным названием (листинг 8.13).

Листинг 8.13. Ошибочное выполнение самообъединения

```
mysql> SELECT prd.name, prd.price FROM products, products AS prd
    -> WHERE products.id_catalog = prd.id_catalog AND
    -> prd.name = 'Maxtor 6Y120P0';
+---------------+---------+
| name          | price   |
+---------------+---------+
| Maxtor 6Y120P0 | 2456.00 |
| Maxtor 6Y120P0 | 2456.00 |
```

```
| Maxtor 6Y120P0 | 2456.00 |
| Maxtor 6Y120P0 | 2456.00 |
| Maxtor 6Y120P0 | 2456.00 |
+----------------+---------+
```

Другим применением самообъединения является вычисление разницы между последовательными строками. В листинге 8.14 представлено содержимое таблицы `orders`.

Листинг 8.14. Содержимое таблицы `orders`

```
mysql> SELECT id_order, ordertime FROM orders ORDER BY ordertime;
+----------+---------------------+
| id_order | ordertime           |
+----------+---------------------+
|        1 | 2005-01-04 10:39:38 |
|        2 | 2005-02-10 09:40:29 |
|        3 | 2005-02-18 13:41:05 |
|        4 | 2005-03-10 18:20:00 |
|        5 | 2005-03-17 19:15:36 |
+----------+---------------------+
```

Пусть требуется вычислить разницу в днях между заказами в электронном магазине. Решить эту задачу можно при помощи запроса, представленного в листинге 8.15.

Листинг 8.15. Вычисления разницы в днях между заказами

```
mysql> SELECT ord1.id_order AS id1,
    ->     ord2.id_order AS id2,
    ->     ord1.ordertime, ord2.ordertime,
    ->     TO_DAYS(ord2.ordertime) - TO_DAYS(ord1.ordertime) AS different
    -> FROM orders AS ord2, orders AS ord1
    -> WHERE ord1.id_order + 1 = ord2.id_order
    -> ORDER BY ord1.ordertime;
+-----+-----+---------------------+---------------------+-----------+
| id1 | id2 | ordertime           | ordertime           | different |
+-----+-----+---------------------+---------------------+-----------+
|   1 |   2 | 2005-01-04 10:39:38 | 2005-02-10 09:40:29 |        37 |
|   2 |   3 | 2005-02-10 09:40:29 | 2005-02-18 13:41:05 |         8 |
|   3 |   4 | 2005-02-18 13:41:05 | 2005-03-10 18:20:00 |        20 |
|   4 |   5 | 2005-03-10 18:20:00 | 2005-03-17 19:15:36 |         7 |
+-----+-----+---------------------+---------------------+-----------+
```

Как видно из листинга 8.15, при самообъединении таблицы `orders`, копии таблицы смещаются друг относительно друга на одну строку за счет условия `ord1.id_order + 1 = ord2.id_order`. При помощи функции `TO_DAYS()` время приводится к дням, после чего производится вычитание полученных результатов. Полученное значение `different` и представляет разницу в днях между заказами.

> ***Замечание***
>
> Синтаксис функции `TO_DAYS()` рассматривается в *главе 16*.

Рассмотрим еще несколько запросов на примере учебной базы данных `shop`. Пусть требуется решить задачу подсчета числа товарных позиций, а также общего числа товара на складе для каждого из каталогов и вывести результирующую таблицу (листинг 8.16).

> ***Замечание***
>
> Такого рода запросы обсуждаются в *главе 21* и осуществляются при помощи специальных агрегатных функций `COUNT()` и `SUM()` с группировкой (`GROUP BY`) по полю `id_catalog`.

Листинг 8.16. Вывод числа товарных позиций и запасов на складе

```
mysql> SELECT id_catalog, COUNT(id_product), SUM(count) FROM products
    -> GROUP BY id_catalog;
+------------+------------------+------------+
| id_catalog | COUNT(id_product) | SUM(count) |
+------------+------------------+------------+
|          1 |                9 |         47 |
|          2 |                6 |         27 |
|          3 |                4 |         17 |
|          4 |                5 |         26 |
|          5 |                6 |         85 |
+------------+------------------+------------+
```

Первый столбец представляет собой внешние ключи таблицы `catalogs`, по которым можно восстановить названия каталогов. При помощи многотабличного запроса к таблицам `products` и `catalogs` вместо данного столбца можно вывести названия каталогов, тем самым представив результат запроса в более удобном виде (листинг 8.17).

Листинг 8.17. Замена первичного ключа `id_catalog` на имя каталога

```
mysql> SELECT catalogs.name, COUNT(id_product), SUM(count)
    -> FROM products, catalogs
    -> WHERE products.id_catalog = catalogs.id_catalog
```

```
    -> GROUP BY products.id_catalog;
+-------------------+-------------------+------------+
| name              | COUNT(id_product) | SUM(count) |
+-------------------+-------------------+------------+
| Процессоры        |                 9 |         47 |
| Материнские платы |                 6 |         27 |
| Видеоадаптеры     |                 4 |         17 |
| Жесткие диски     |                 5 |         26 |
| Оперативная память |                6 |         85 |
+-------------------+-------------------+------------+
```

Обязательным условием в данном запросе является равенство полей `id_catalog` таблиц `products` и `catalogs` (`products.id_catalog = catalogs.id_catalog`). Именно при помощи данного условия осуществляется связь этих двух таблиц.

> ***Замечание***
>
> Если имя столбца является уникальным, то при перечислении его в списке столбцов и конструкциях `WHERE`, `GROUP BY`, `ORDER BY` можно не использовать полное имя столбца.

Среди таблиц учебной базы данных `shop` примечательной является таблица `orders` (листинг 8.18), которая содержит внешний ключ `id_product` для связи с таблицей `products` и внешний ключ `id_user` для связи с таблицей `users`. Поэтому, чтобы результирующая таблица содержала данные заказа, фамилию покупателя и название товара, требуется осуществить трехтабличный запрос.

Листинг 8.18. Содержимое таблицы `orders`

```
mysql> SELECT * FROM orders;
+----------+---------+---------------------+--------+------------+
| id_order | id_user | ordertime           | number | id_product |
+----------+---------+---------------------+--------+------------+
|        1 |       3 | 2005-01-04 10:39:38 |      1 |          8 |
|        2 |       6 | 2005-02-10 09:40:29 |      2 |         10 |
|        3 |       1 | 2005-02-18 13:41:05 |      4 |         20 |
|        4 |       3 | 2005-03-10 18:20:00 |      1 |         20 |
|        5 |       3 | 2005-03-17 19:15:36 |      1 |         20 |
+----------+---------+---------------------+--------+------------+
```

Запрос, после выполнения которого результирующая таблица содержит вместо внешних ключей `id_user` и `id_product` фамилию покупателя и название товарной позиции, представлен в листинге 8.19.

Листинг 8.19. Трехтабличный запрос

```
mysql> SELECT id_order, users.surname, ordertime, products.name
    -> FROM users, orders, products
    -> WHERE orders.id_user = users.id_user AND
    ->       orders.id_product = products.id_product;
+----------+----------+---------------------+-----------------------+
| id_order | surname  | ordertime           | name                  |
+----------+----------+---------------------+-----------------------+
|        1 | Симдянов | 2005-01-04 10:39:38 | Intel Pentium 4 3.0GHz |
|        2 | Корнеев  | 2005-02-10 09:40:29 | Gigabyte GA-8I848P-RS  |
|        3 | Иванов   | 2005-02-18 13:41:05 | Maxtor 6Y120P0         |
|        4 | Симдянов | 2005-03-10 18:20:00 | Maxtor 6Y120P0         |
|        5 | Симдянов | 2005-03-17 19:15:36 | Maxtor 6Y120P0         |
+----------+----------+---------------------+-----------------------+
```

Помимо того, что после ключевого слова `FROM` требуется указать три таблицы: `users`, `orders` и `products`, `WHERE`-условие должно включать две связи: таблицы `orders` с `users` и `orders` с `products`, которые объединяются при помощи оператора `AND` (И).

Если кроме представленной в листинге 8.19 информации, в результирующую таблицу необходимо поместить столбец с названием элемента каталога, к которому относится выбранный товар, потребуется осуществить четырехтабличный запрос, представленный в листинге 8.20.

Листинг 8.20. Четырехтабличный запрос

```
mysql> SELECT id_order, users.surname, products.name, catalogs.name
    -> FROM users, orders, products, catalogs
    -> WHERE orders.id_user = users.id_user AND
    ->       orders.id_product = products.id_product AND
    ->       products.id_catalog = catalogs.id_catalog;
+----------+----------+-----------------------+------------------+
| id_order | surname  | name                  | name             |
+----------+----------+-----------------------+------------------+
|        1 | Симдянов | Intel Pentium 4 3.0GHz | Процессоры       |
|        2 | Корнеев  | Gigabyte GA-8I848P-RS  | Материнские платы |
|        3 | Иванов   | Maxtor 6Y120P0         | Жесткие диски    |
|        4 | Симдянов | Maxtor 6Y120P0         | Жесткие диски    |
|        5 | Симдянов | Maxtor 6Y120P0         | Жесткие диски    |
+----------+----------+-----------------------+------------------+
```

Здесь, помимо связи таблицы `orders` с таблицами `users` и `products`, добавляется третья связь таблицы `products` с таблицей `catalogs` (`products.id_catalog = catalogs.id_catalog`).

С увеличением количества таблиц в запросе резко возрастает объем работы, необходимой для выполнения запроса, поэтому по возможности в запросе не следует использовать больше трех-четырех таблиц.

8.2. Объединение таблиц при помощи *JOIN*

Оператор `JOIN` позволяет объединять таблицы и имеет многочисленные варианты использования, которые будут рассмотрены далее в этом разделе. 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, рассмотренному в *разд. 8.1*.

Листинг 8.21. Использование ключевого слова `JOIN`

```
mysql> SELECT * FROM tbl1, tbl2 WHERE tbl1.id = tbl2.id;
+----+------+----+--------+
| id | name | id | letter |
+----+------+----+--------+
|  2 | Б    |  2 | Г      |
|  3 | В    |  3 | Д      |
+----+------+----+--------+
mysql> 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 8.21, полностью идентичны. Для формирования условия в запросах, использующих объединение `JOIN`, вместо ключевого слова `WHERE` предпочтительно указывать ключевое слово `ON`, как это продемонстрировано в листинге 8.22.

Листинг 8.22. Использование ключевого слова ON

```
mysql> 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 объединения таблиц, которые осуществляются при помощи конструкций LEFT JOIN и RIGHT JOIN, соответственно.

В листинге 8.21 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 8.23).

Листинг 8.23. Левое объединение таблиц tbl1 и tbl2

```
mysql> 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 8.23, записи в таблице tbl1 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 заметить, что для задания условия вместо ключевого слова WHERE при левом и правом объединениях используется ключевое слово ON.

В листинге 8.24 демонстрируется "правое" объединение при помощи конструкции RIGHT JOIN.

Листинг 8.24. Правое объединение таблиц `tbl1` и `tbl2`

```
mysql> 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 8.24, 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, приведенные в листингах 8.23 и 8.24, можно представить так, как это показано в листинге 8.25.

Листинг 8.25. Использование ключевого слова `USING`

```
mysql> SELECT * FROM tbl1 LEFT JOIN tbl2 USING(id);
+----+------+------+--------+
| id | name | id   | letter |
+----+------+------+--------+
|  1 | А    | NULL | NULL   |
|  2 | Б    |    2 | Г      |
|  3 | В    |    3 | Д      |
+----+------+------+--------+
```

```
mysql> 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.

Возвращаясь к учебной базе данных `shop`, рассмотрим несколько запросов к таблицам, входящим в ее состав. В листинге 8.26 при помощи конструкции `JOIN...USING()` извлекается количество товарных позиций в каталоге.

Листинг 8.26. Извлечение количества товарных позиций в каталоге

```
mysql> SELECT catalogs.name, COUNT(id_product)
    -> FROM catalogs JOIN products USING(id_catalog)
    -> GROUP BY products.id_catalog;
+-------------------+-------------------+
| name              | COUNT(id_product) |
+-------------------+-------------------+
| Процессоры        |                 9 |
| Материнские платы |                 6 |
| Видеоадаптеры     |                 4 |
| Жесткие диски     |                 5 |
| Оперативная память |                6 |
+-------------------+-------------------+
```

Допустим, происходит расширение ассортимента товарных позиций, и в списке каталога появляется новый элемент "Периферия". SQL-запрос, добавляющий данную позицию, представлен в листинге 8.27.

Листинг 8.27. Добавление нового элемента "Периферия"

```
mysql> INSERT INTO catalogs VALUES (NULL, 'Периферия');
mysql> SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
|          6 | Периферия          |
+------------+--------------------+
```

Однако запрос из листинга 8.27 не отражает наличие нового элемента каталога в электронном магазине, т. к. таблица `products` еще не содержит ни одной записи, относящейся к новой позиции каталога. Для того чтобы название элемента каталога "Периферия" появилось в результирующей таблице, необходимо провести левое объединение таблиц `catalogs` и `products`, причем таблица `catalogs` должна выступать в качестве "левой" таблицы (листинг 8.28).

Листинг 8.28. Полный список каталогов

```
mysql> SELECT catalogs.name, COUNT(id_product)
    -> FROM catalogs LEFT JOIN products USING(id_catalog)
    -> GROUP BY products.id_catalog;
+--------------------+-------------------+
| name               | COUNT(id_product) |
+--------------------+-------------------+
| Периферия          |                 0 |
| Процессоры         |                 9 |
| Материнские платы  |                 6 |
| Видеоадаптеры      |                 4 |
| Жесткие диски      |                 5 |
| Оперативная память |                 6 |
+--------------------+-------------------+
```

Как видно из листинга 8.28, левое объединение позволило включить элемент каталога "Периферия", для которого товарные позиции пока отсутствуют.

Пусть требуется вывести список покупателей и число осуществленных ими покупок, причем покупателей необходимо отсортировать в порядке убывания числа оплаченных ими заказов. Для решения этой задачи можно воспользоваться запросом, представленным в листинге 8.29.

Листинг 8.29. Список покупателей и число оплаченных ими заказов

```
mysql> SELECT users.surname, users.name,
    -> users.patronymic, COUNT(orders.id_order) AS total
    -> FROM users JOIN orders USING(id_user)
    -> GROUP BY users.id_user
    -> ORDER BY total DESC;
+----------+----------+--------------+-------+
| surname  | name     | patronymic   | total |
+----------+----------+--------------+-------+
| Симдянов | Игорь    | Вячеславович |     3 |
| Корнеев  | Александр | Александрович |     1 |
| Иванов   | Александр | Валерьевич   |     1 |
+----------+----------+--------------+-------+
```

Как видно из листинга 8.29, в списке присутствуют только те покупатели, которые оплатили хотя бы одну покупку, покупатели, на счету у которых нет ни одной покупки, в список не входят. Для того чтобы вывести полный список покупателей, необходимо вместо перекрестного объединения таблиц `users` и `orders` воспользоваться левым объединением, где в качестве "левой" таблицы выступит таблица `users` (листинг 8.30).

Листинг 8.30. Полный список покупателей

```
mysql> SELECT users.surname, users.name,
    -> users.patronymic, COUNT(orders.id_order) AS total
    -> FROM users LEFT JOIN orders USING(id_user)
    -> GROUP BY users.id_user
    -> ORDER BY total DESC;
+----------+----------+--------------+-------+
| surname  | name     | patronymic   | total |
+----------+----------+--------------+-------+
| Симдянов | Игорь    | Вячеславович |     3 |
| Иванов   | Александр | Валерьевич   |     1 |
| Корнеев  | Александр | Александрович |     1 |
```

```
| Лосев     | Сергей    | Иванович      |     0 |
| Кузнецов  | Максим    | Валерьевич    |     0 |
| Нехорошев | Анатолий  | Юрьевич       |     0 |
+-----------+-----------+---------------+-------+
```

Помимо условия `ON` или `USING`, в запросах на объединение могут использоваться традиционные конструкции условия. Например, запрос, представленный в листинге 8.31, извлекает список покупателей и число их покупок при условии, что покупателя зовут Александр.

Листинг 8.31. Использование ключевого слова `WHERE`

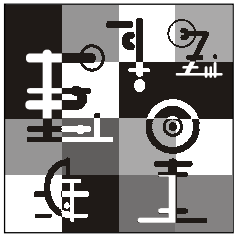
```
mysql> SELECT users.surname, users.name,
    -> users.patronymic, COUNT(orders.id_order) AS total
    -> FROM users LEFT JOIN orders USING(id_user)
    -> WHERE name = 'Александр'
    -> GROUP BY users.id_user
    -> ORDER BY total DESC;
+---------+-----------+---------------+-------+
| surname | name      | patronymic    | total |
+---------+-----------+---------------+-------+
| Иванов  | Александр | Валерьевич    |     1 |
| Корнеев | Александр | Александрович |     1 |
+---------+-----------+---------------+-------+
```

Для этого применяется условие `WHERE name = 'Александр'`. Однако использовать столбец `total` в условии `WHERE` уже не получится, т. к. это групповой столбец, сформированный агрегатной функцией `COUNT()` и конструкцией `GROUP BY`. Для формирования условия с его участием необходимо использовать условие `HAVING`. Пусть требуется извлечь всех покупателей магазина, число покупок (`total`) у которых меньше трех (листинг 8.32).

Листинг 8.32. Использование ключевого слова `HAVING`

```
mysql> SELECT users.surname, users.name,
    -> users.patronymic, COUNT(orders.id_order) AS total
    -> FROM users LEFT JOIN orders USING(id_user)
    -> GROUP BY users.id_user
    -> HAVING total < 3
```

```
    -> ORDER BY total DESC;
+-----------+-----------+---------------+-------+
| surname   | name      | patronymic    | total |
+-----------+-----------+---------------+-------+
| Иванов    | Александр | Валерьевич    |     1 |
| Корнеев   | Александр | Александрович |     1 |
| Лосев     | Сергей    | Иванович      |     0 |
| Кузнецов  | Максим    | Валерьевич    |     0 |
| Нехорошев | Анатолий  | Юрьевич       |     0 |
+-----------+-----------+---------------+-------+
```



Глава 9

Удаление данных

Время от времени возникает задача удаления записей из базы данных, например, комплектующие могут устаревать и быть снятыми с производства. Для того чтобы отразить этот факт в учебной базе данных, необходимо удалить соответствующую запись в таблице `products`.

Для удаления записей из таблиц, предусмотрены два оператора:

- `DELETE`;
- `TRUNCATE TABLE`.

9.1. 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 9.1 из таблицы `catalogs` удаляются записи, имеющие значение первичного ключа `id_catalog` больше двух.

Листинг 9.1. Удаление записей из таблицы `catalogs`

```
mysql> DELETE FROM catalogs WHERE id_catalog > 2;
mysql> SELECT * FROM catalogs;
+------------+------------------+
| id_catalog | name             |
+------------+------------------+
|          2 | Материнские платы |
|          1 | Процессоры       |
+------------+------------------+
```

Если в операторе DELETE отсутствует условие WHERE, из таблицы удаляются все записи (листинг 9.2).

Листинг 9.2. Удаление всех записей таблицы `catalogs`

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

Ограничение LIMIT позволяет задать максимальное число записей, которые могут быть уничтожены. В листинге 9.3 удаляются не более 3 записей таблицы orders.

Листинг 9.3. Удаление записей из таблицы `orders`

```
mysql> DELETE FROM orders LIMIT 3;
mysql> SELECT * FROM orders;
+----------+---------+---------------------+--------+------------+
| id_order | id_user | ordertime           | number | id_product |
+----------+---------+---------------------+--------+------------+
|        4 |       3 | 2005-03-10 18:20:00 |      1 |         20 |
|        5 |       3 | 2005-03-17 19:15:36 |      1 |         20 |
+----------+---------+---------------------+--------+------------+
```

Так как таблица содержит 5 записей, то в результате в таблице остаются две записи. Инструкция ORDER BY обычно применяется совместно с ключевым словом LIMIT. Например, если необходимо удалить 20 первых записей таблицы, то производится сортировка по полю DATETIME — это гарантирует, что удаляться будут в первую очередь самые старые данные.

9.2. Оператор *TRUNCATE TABLE*

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 9.4.

Замечание

Оптимизатор запросов СУБД MySQL автоматически использует оператор TRUNCATE TABLE, если оператор DELETE не содержит WHERE-условия или конструкции LIMIT.

Листинг 9.4. Удаление таблицы products

```
mysql> TRUNCATE TABLE products;
```

9.3. Удаление из нескольких таблиц

Помимо оператора SELECT, в многотабличных запросах можно использовать оператор DELETE. Для этого сразу после ключевого слова FROM имена таблиц перечисляются через запятую. Пусть требуется удалить из таблицы catalogs все записи, чей первичный ключ id_catalog больше 3. Для этого можно использовать запрос, представленный в листинге 9.5.

> ***Замечание***
>
> В многотабличных запросах с участием SQL-оператора DELETE не допускается использование конструкций ORDER BY и LIMIT.

Листинг 9.5. Удаление записей из таблицы catalogs

```
mysql> DELETE FROM catalogs WHERE id_catalog > 3;
mysql> SELECT * FROM catalogs;
+------------+------------------+
| id_catalog | name             |
+------------+------------------+
|          1 | Процессоры       |
|          2 | Материнские платы |
|          3 | Видеоадаптеры    |
+------------+------------------+
```

Запрос из листинга 9.5 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. В этом можно убедиться, запросив число товарных позиций для каждого из каталогов при помощи SQL-запроса, представленного в листинге 9.6.

Листинг 9.6. Товарные позиции не удалились из таблицы products

```
mysql> SELECT id_catalog, COUNT(*) FROM products GROUP BY id_catalog;
+------------+----------+
| id_catalog | COUNT(*) |
```

```
+------------+----------+
|          1 |        9 |
|          2 |        6 |
|          3 |        4 |
|          4 |        5 |
|          5 |        6 |
+------------+----------+
```

При использовании многотабличных запросов с помощью оператора `DELETE` допустимы две формы, представленные в листинге 9.7, где из таблицы `products` удаляются все записи, для которых имеется соответствие в таблице `catalogs`.

Листинг 9.7. Многотабличное удаление из таблицы `products`

```
mysql> DELETE products FROM products, catalogs
    -> WHERE products.id_catalog = catalogs.id_catalog;
mysql> 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 9.8, затронут уже обе таблицы.

Листинг 9.8. Многотабличное удаление из таблиц `products` и `catalogs`

```
mysql> DELETE products, catalogs FROM products, catalogs
    -> WHERE products.id_catalog = catalogs.id_catalog;
mysql> 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 9.9).

Листинг 9.9. Удаление позиций каталога с первичным ключом больше 3

```
mysql> DELETE products, catalogs FROM products, catalogs
    -> WHERE products.id_catalog = catalogs.id_catalog AND
    -> catalogs.id_catalog > 3;
```

```
mysql> SELECT * FROM catalogs;
+------------+------------------+
| id_catalog | name             |
+------------+------------------+
|          1 | Процессоры       |
|          2 | Материнские платы |
|          3 | Видеоадаптеры    |
+------------+------------------+
mysql> SELECT id_catalog, COUNT(*) FROM products GROUP BY id_catalog;
+------------+----------+
| id_catalog | COUNT(*) |
+------------+----------+
|          1 |        9 |
|          2 |        6 |
|          3 |        4 |
+------------+----------+
```

Как видно из листинга 9.9, многотабличное удаление позволяет удалить позиции каталога, не нарушая ссылочной целостности базы данных.

> ***Замечание***
>
> СУБД MySQL обладает автоматическими средствами поддержания ссылочной целостности, когда удаление записей приводят либо к блокировке удаления, либо к каскадному удалению ссылающихся записей. Более подробно этот вопрос обсуждается в *главе 25*.

Рассмотренные выше примеры используют перекрестное объединение, хотя многотабличные запросы с участием оператора `DELETE` допускают любой тип объединения, рассмотренный ранее в *главе 8* на примере оператора `SELECT`.

Для достижения совместимости с MS Access в многотабличных операторах `DELETE` допускается использование символов `.*` после имени таблицы (листинг 9.10).

Листинг 9.10. Использование символов .*

```
mysql> DELETE products.*, catalogs.* FROM products, catalogs
    -> WHERE products.id_catalog = catalogs.id_catalog;
mysql> 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 9.11).

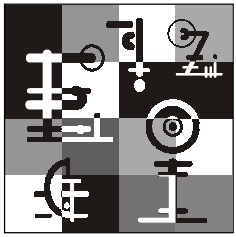
Листинг 9.11. Использование ключевого слова AS в СУБД MySQL 4.0

```
mysql> 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 9.12).

Листинг 9.12. Использование ключевого слова AS в СУБД MySQL 4.1 и выше

```
mysql> DELETE p, c FROM products AS p, catalogs AS c
    -> WHERE p.id_catalog = c.id_catalog;
```



Глава 10

Обновление данных

Операция обновления позволяет менять значения полей в уже существующих записях, это может понадобиться при изменении цен комплектующих, их количества на складе и т. д. 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.

10.1. 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`*.

В листинге 10.1 запись "Процессоры" в таблице `catalogs` изменяется на "Процессоры (Intel)".

Листинг 10.1. Обновление таблицы `catalogs`

```
mysql> SELECT * FROM catalogs ORDER BY name;
+------------+--------------------+
| id_catalog | name               |
```

```
+------------+--------------------+
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры         |
+------------+--------------------+
mysql> UPDATE catalogs SET name = 'Процессоры (Intel)'
    -> WHERE name = 'Процессоры';
mysql> SELECT * FROM catalogs ORDER BY name;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры (Intel) |
+------------+--------------------+
```

Изменения можно производить не только над выборочными записями, но и над всей таблицей в целом. Пусть требуется изменить цену комплектующих в учебной базе данных в связи с изменением курса доллара. Для этого следует разделить цену в рублях на старый курс (`28.5`), чтобы получить цену в долларах, и умножить результат на новый курс (`27.0`) (листинг 10.2).

Листинг 10.2. Изменение курса доллара

```
mysql> UPDATE products SET price = (price/28.5)*27.0;
mysql> SELECT name, price FROM products ORDER BY price;
+-----------------------------+-----------+
| name                        | price     |
+-----------------------------+-----------+
| DDR-400 256MB PQI           |  851.6842 |
| DDR-400 256MB Kingston      | 1027.8947 |
| DDR-400 256MB Hynix Original| 1116.9474 |
| Celeron 1.8                 | 1511.0526 |
| ASUSTEK V9520X              | 1517.6842 |
| DDR-400 512MB PQI           | 1601.0526 |
| DDR-400 512MB Hynix         | 1626.6316 |
```

```
| Celeron D 315 2.26GHz          | 1781.0526 |
| Gigabyte GA-8I848P-RS          | 1796.2105 |
| DDR-400 512MB Kingston         | 1830.3158 |
| Celeron D 320 2.4GHz           | 1858.7368 |
| Celeron 2.0GHz                 | 1865.3684 |
| Samsung SP0812C                | 1982.8421 |
| Celeron 2.4GHz                 | 1998.0000 |
| Gigabyte GA-8IPE1000G          | 2168.5263 |
| Epox EP-4PDA3I                 | 2168.5263 |
| Gigabyte GA-8IG1000            | 2292.6316 |
| Maxtor 6Y120P0                 | 2326.7368 |
| Seagate ST3120026A             | 2338.1053 |
| Asustek P4P800-VM\L i865G      | 2385.4737 |
| SAPPHIRE 256MB RADEON 9550     | 2586.3158 |
| Celeron D 325 2.53GHz          | 2602.4211 |
| Seagate Barracuda ST3160023A   | 2973.7895 |
| Maxtor 6B200P0                 | 3400.1053 |
| ASUSTEK A9600XT/TD             | 4884.6316 |
| Asustek P4C800-E Delux         | 5111.0526 |
| Intel Pentium 4 3.0GHz         | 5374.4211 |
| GIGABYTE AGP GV-N59X128D       | 5576.2105 |
| Intel Pentium 4 3.0GHz         | 5823.4737 |
| Intel Pentium 4 3.2GHz         | 6876.9474 |
+--------------------------------+-----------+
```

Точно так же, как и в случае оператора `DELETE`, инструкции `LIMIT` и `ORDER BY` позволяют ограничить число записей, подвергающихся изменению. Пусть стоит задача для десяти самых дешевых товарных позиций вернуть цену с учетом старого курса доллара, в этом случае запрос к таблице `products` может выглядеть так, как это продемонстрировано в листинге 10.3.

Листинг 10.3. Изменение цены для 10 самых дешевых товарных позиций

```
mysql> UPDATE products SET price = (price/27.0)*28.5
    -> ORDER BY price LIMIT 10;
mysql> SELECT name, price, count FROM products ORDER BY price;
+--------------------------------+---------+
| name                           | price   |
+--------------------------------+---------+
| DDR-400 256MB PQI              |  899.00 |
```

```
| DDR-400 256MB Kingston         | 1085.00 |
| DDR-400 256MB Hynix Original   | 1179.00 |
| Celeron 1.8                    | 1595.00 |
| ASUSTEK V9520X                 | 1602.00 |
| DDR-400 512MB PQI              | 1690.00 |
| DDR-400 512MB Hynix            | 1717.00 |
| Celeron D 320 2.4GHz           | 1858.74 |
| Celeron 2.0GHz                 | 1865.37 |
| Celeron D 315 2.26GHz          | 1880.00 |
| Gigabyte GA-8I848P-RS          | 1896.00 |
| DDR-400 512MB Kingston         | 1932.00 |
| Samsung SP0812C                | 1982.84 |
| Celeron 2.4GHz                 | 1998.00 |
| Gigabyte GA-8IPE1000G          | 2168.53 |
| Epox EP-4PDA3I                 | 2168.53 |
| Gigabyte GA-8IG1000            | 2292.63 |
| Maxtor 6Y120P0                 | 2326.74 |
| Seagate ST3120026A             | 2338.11 |
| Asustek P4P800-VM\L i865G      | 2385.47 |
| SAPPHIRE 256MB RADEON 9550     | 2586.32 |
| Celeron D 325 2.53GHz          | 2602.42 |
| Seagate Barracuda ST3160023A   | 2973.79 |
| Maxtor 6B200P0                 | 3400.11 |
| ASUSTEK A9600XT/TD             | 4884.63 |
| Asustek P4C800-E Delux         | 5111.05 |
| Intel Pentium 4 3.0GHz         | 5374.42 |
| GIGABYTE AGP GV-N59X128D       | 5576.21 |
| Intel Pentium 4 3.0GHz         | 5823.47 |
| Intel Pentium 4 3.2GHz         | 6876.95 |
+--------------------------------+---------+
```

За один SQL-запрос `UPDATE` обновлению может подвергаться несколько столбцов таблицы. Пусть теперь необходимо для десяти самых дешевых товаров уменьшить число товарных позиций на складе на одну единицу, а цену — на 5%. Решить эту задачу позволяет запрос, представленный в листинге 10.4.

Замечание

Вычисления в листинге 10.4 производятся без учета изменений в листингах 10.2 и 10.3.

Листинг 10.4. Уменьшение числа товарных позиций на складе, 5% скидка на товары

```
mysql> UPDATE products
    -> SET price = price*0.95,
    ->     count = count - 1
    -> ORDER BY price LIMIT 10;
mysql> SELECT name, price, count FROM products ORDER BY price;
+------------------------------+---------+-------+
| name                         | price   | count |
+------------------------------+---------+-------+
| DDR-400 256MB PQI            | 854.05  |     9 |
| DDR-400 256MB Kingston       | 1030.75 |    19 |
| DDR-400 256MB Hynix Original | 1120.05 |    14 |
| Celeron 1.8                  | 1515.25 |     9 |
| ASUSTEK V9520X               | 1521.90 |     5 |
| DDR-400 512MB PQI            | 1605.50 |    11 |
| DDR-400 512MB Hynix          | 1631.15 |     7 |
| Celeron D 315 2.26GHz        | 1786.00 |     5 |
| Gigabyte GA-8I848P-RS        | 1801.20 |     3 |
| Celeron D 320 2.4GHz         | 1831.20 |     0 |
| DDR-400 512MB Kingston       | 1932.00 |    20 |
| Celeron 2.0GHz               | 1934.46 |     2 |
| Samsung SP0812C              | 2056.28 |     5 |
| Celeron 2.4GHz               | 2072.00 |     4 |
| Gigabyte GA-8IPE1000G        | 2248.84 |     6 |
| Epox EP-4PDA3I               | 2248.84 |     5 |
| Gigabyte GA-8IG1000          | 2377.54 |     2 |
| Maxtor 6Y120P0               | 2412.91 |     6 |
| Seagate ST3120026A           | 2424.70 |     8 |
| Asustek P4P800-VM\L i865G    | 2473.82 |     6 |
| SAPPHIRE 256MB RADEON 9550   | 2682.11 |     3 |
| Celeron D 325 2.53GHz        | 2698.81 |     6 |
| Seagate Barracuda ST3160023A | 3083.93 |     3 |
| Maxtor 6B200P0               | 3526.04 |     4 |
| ASUSTEK A9600XT/TD           | 5065.54 |     2 |
| Asustek P4C800-E Delux       | 5300.35 |     4 |
| Intel Pentium 4 3.0GHz       | 5573.47 |    12 |
| GIGABYTE AGP GV-N59X128D     | 5782.74 |     6 |
| Intel Pentium 4 3.0GHz       | 6039.16 |     1 |
| Intel Pentium 4 3.2GHz       | 7131.65 |     5 |
+------------------------------+---------+-------+
```

Если указывается необязательное ключевое слово IGNORE, то команда обновления не будет прервана, даже если при обновлении возникнет ошибка дублирования ключей. Строки, из-за которых возникают конфликтные ситуации, обновлены не будут.

10.2. Многотабличный оператор *UPDATE*

Использование многотабличного оператора UPDATE сходно с использованием оператора DELETE. 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 10.5.

Листинг 10.5. Редактирование таблицы `catalogs`

```
mysql> 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 10.6.

Листинг 10.6. Обновление сразу двух таблиц

```
mysql> UPDATE catalogs, products
    -> SET catalogs.id_catalog = 10, products.id_catalog = 10
    -> WHERE catalogs.id_catalog = 5 AND products.id_catalog = 5;
mysql> SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры (Intel) |
|          2 | Материнские платы  |
|          3 | Видеоадаптары      |
|          4 | Жесткие диски      |
|         10 | Оперативная память |
+------------+--------------------+
mysql> SELECT id_catalog FROM products GROUP BY id_catalog;
+------------+
| id_catalog |
+------------+
|          1 |
```

```
|           2 |
|           3 |
|           4 |
|          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 10.7), синтаксис которого подробно рассматривается в *главе 13*.

Листинг 10.7. Добавление нового столбца `catalog` в таблицу `products`

```
mysql> ALTER TABLE products ADD catalog TINYTEXT AFTER id_catalog;
mysql> DESCRIBE products;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| id_product  | int(11)      |      | PRI | NULL    | auto_increment |
| name        | tinytext     |      | MUL |         |                |
| price       | decimal(7,2) |      |     | 0.00    |                |
| count       | int(11)      |      |     | 0       |                |
| mark        | float(4,1)   |      |     | 0.0     |                |
| description | text         |      |     |         |                |
| id_catalog  | int(11)      |      | 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 10.8).

Листинг 10.8. Заполнение поля `catalog`

```
mysql> SELECT id_catalog, catalog FROM products GROUP BY id_catalog;
+------------+---------+
| id_catalog | catalog |
+------------+---------+
|          1 | NULL    |
|          2 | NULL    |
|          3 | NULL    |
```

```
|          4 | NULL    |
|         10 | NULL    |
+------------+---------+
mysql> UPDATE products, catalogs SET products.catalog = catalogs.name
    -> WHERE products.id_catalog = catalogs.id_catalog;
mysql> SELECT id_catalog, catalog FROM products GROUP BY id_catalog;
+------------+--------------------+
| id_catalog | catalog            |
+------------+--------------------+
|          1 | Процессоры (Intel) |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|         10 | Оперативная память |
+------------+--------------------+
```

Как видно из листинга 10.8, таблица `catalogs` не подвергается изменениям, но активно участвует в многотабличном запросе с участием 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, рассмотренный ранее на примере оператора `SELECT`. В листинге 10.9 приводятся три запроса, идентичные представленному в листинге 10.8.

Листинг 10.9. Многотабличные запросы с использованием ключевого слова JOIN

```
mysql> UPDATE products JOIN catalogs SET products.catalog = catalogs.name
    -> WHERE products.id_catalog = catalogs.id_catalog;
mysql> UPDATE products JOIN catalogs
    -> ON products.id_catalog = catalogs.id_catalog
    -> SET products.catalog = catalogs.name;
mysql> UPDATE products JOIN catalogs USING(id_catalog)
    -> SET products.catalog = catalogs.name;
```

Точно так же, как и в случае оператора `SELECT`, допускается использование псевдонимов для таблиц, которые назначаются при помощи оператора `AS` (листинг 10.10).

Листинг 10.10. Использование псевдонимов

```
mysql> UPDATE products AS p, catalogs AS c SET p.catalog = c.name
    -> WHERE p.id_catalog = c.id_catalog;
```

10.3. 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,...),(...),...
```

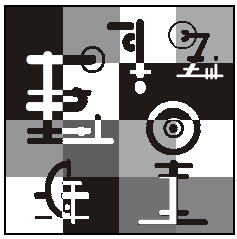
В таблицу *tbl* вставляются значения, определяемые в списке после ключевого слова VALUES. Задать порядок столбцов можно при помощи необязательного списка *col_name*, следующего за именем таблицы *tbl*. Точно так же, как и в случае оператора INSERT, оператор REPLACE допускает многострочный формат. Пример использования оператора приведен в листинге 10.11, где в таблицу catalogs добавляется пять новых записей.

> ***Замечание***
>
> Многотабличный синтаксис для операторов REPLACE и INSERT не предусмотрен.

Листинг 10.11. Использование оператора REPLACE

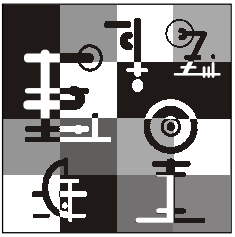
```
mysql> SELECT * FROM catalogs ORDER BY id_catalog;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры (Intel) |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
mysql> REPLACE INTO catalogs VALUES
    -> (4, 'Сетевые адаптеры'),
    -> (5, 'Программное обеспечение'),
    -> (6, 'Мониторы'),
```

```
    -> (7, 'Периферия'),
    -> (8, 'CD-RW/DVD');
mysql> 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              |
+------------+------------------------+
```



ЧАСТЬ II

СЛОЖНЫЕ ВОПРОСЫ MYSQL



Глава 11

Типы и структура таблиц

В *главе 4* упоминалось, что СУБД MySQL поддерживает несколько видов таблиц, каждая из которых имеет свои возможности и ограничения. В данной главе типы таблиц будут рассмотрены более подробно.

> ***Замечание***
>
> По мере рассмотрения различных типов таблиц, будут обсуждаться элементы оператора `CREATE TABLE`, характерные для того или иного типа таблиц. Полное описание синтаксиса оператора `CREATE TABLE` приводится в *главе 12*.

> ***Замечание***
>
> Не все типы таблиц поддерживаются по умолчанию. Для того чтобы узнать, какие типы таблиц поддерживаются вашей версией MySQL, а какие нет, необходимо воспользоваться оператором `SHOW ENGINES`, который рассматривается в *главе 30*.

11.1. MyISAM

Тип таблиц MyISAM является "родным" для базы СУБД MySQL. Если в операторе `CREATE TABLE` *(см. главу 4)* тип таблицы не указывается — назначается именно этот тип.

> ***Замечание***
>
> Долгое время наряду с MyISAM в MySQL присутствовал устаревший тип таблиц ISAM, но в версии MySQL 4.1 он был объявлен официально устаревшим, а в MySQL 5.0 его поддержка была исключена. Тип таблиц MyISAM является модифицированной и улучшенной версией ISAM.

База данных в MySQL организуется как подкаталог каталога mysql\data. Таблицы базы данных организуются как файлы данного каталога. Каждая MyISAM-таблица хранится на диске в трех файлах, имена которых совпадают с названием таблицы, а расширение может принимать одно из следующих значений:

- frm — содержит структуру таблицы, в файле данного типа хранится информация об именах и типах столбцов и индексов;

- myd — файл, в котором содержатся данные таблицы (MYData);
- myi — файл, в котором содержатся индексы таблицы (MYIndex).

Таблицы MyISAM обладают рядом особенностей.

- Данные хранятся в кросс-платформенном формате, это позволяет переносить базы данных с сервера непосредственным копированием файлов, минуя промежуточные форматы.
- Максимальное число индексов в таблице составляет 64 (32 до появления версии 4.1.2). Каждый индекс может состоять, максимум, из 16 столбцов. Начиная с версии 5.1, можно увеличить число индексов в таблице до 128 при помощи параметра `--with-max-indexes=N`.
- Начиная с версии MySQL 4.1, для каждого из текстовых столбцов может быть назначена своя кодировка.
- Допускается индексирование текстовых столбцов, в том числе и переменной длины.
- Поддерживается полнотекстовый поиск.
- Каждая таблица имеет специальный флаг, указывающий правильность закрытия таблиц. Если сервер останавливается аварийно, то при его повторном старте незакрытые флаги сигнализируют о возможных сбойных таблицах, сервер автоматически проверяет их и пытается восстановить.
- Расширенная поддержка кодировок.

При создании таблиц с использованием оператора `CREATE TABLE` *(см. главу 12)* можно указать тип таблицы при помощи ключевого слова `ENGINE` или `TYPE` (листинг 11.1)

Листинг 11.1. Создание MyISAM-таблицы

```
CREATE TABLE tbl (i INT) ENGINE = MyISAM;
CREATE TABLE tbl (i INT) TYPE = MyISAM;
```

Использование ключевого слова `ENGINE` является более предпочтительным, но в отличие от `TYPE` не может быть применено в MySQL до версии 4.0.18. Для создания MyISAM-таблиц можно не указывать ключевое слово `ENGINE`, т. к. данный тип таблиц назначается по умолчанию.

11.2. MERGE

Тип таблиц MERGE позволяет сгруппировать несколько таблиц типа MyISAM в одну. Такой тип таблиц применяется главным образом 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 11.2 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 11.2. 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 11.3.

Листинг 11.3. 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 11.2, 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 11.4 демонстрируется вставка нового значения в таблицу `total`, в определении которой опущен параметр `INSERT_METHOD`.

Листинг 11.4. Ошибочное добавление новой записи в таблицу MERGE

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

11.3. MEMORY (HEAP)

Тип таблиц MEMORY хранится в оперативной памяти, поэтому все запросы к такой таблице выполняются очень быстро. Недостатком 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.

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 11.5).

Листинг 11.5. Создание MEMORY-таблицы

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

11.4. EXAMPLE

Данный тип таблиц является заглушкой: можно создать таблицу данного типа, но сохранить или получить из нее данные нельзя (см. листинг 11.6). При создании таблиц данного типа, точно так же как и в случае MEMORY, создается один файл с расширением frm, в котором определяется структура таблицы.

Тип EXAMPLE был введен для удобства сторонних разработчиков и демонстрирует, каким образом следует создавать собственные типы таблиц. Не следует забывать, что MySQL является базой данных с открытым кодом, и любой желающий может моди-

фицировать ее код для своих нужд. Так типы таблиц InnoDB и BDB, рассматриваемые далее, были разработаны сторонними компаниями.

Замечание

Тип таблиц EXAMPLE был введен в MySQL, начиная с версии 4.1.3.

Для создания EXAMPLE-таблицы параметру `ENGINE` в операторе `CREATE TABLE` необходимо присвоить значение `EXAMPLE` (листинг 11.6).

Листинг 11.6. Создание EXAMPLE-таблицы

```
CREATE TABLE test (i INT) ENGINE = EXAMPLE;
Query OK, 0 rows affected (0.09 sec)
INSERT INTO test VALUES(1),(2),(3);
ERROR 1031 (HY000): Table storage engine for 'test' doesn't have this
option
SELECT * FROM test;
Empty set (0.03 sec)
```

Как видно из листинга 11.6, попытки вставки записи в таблицу типа `EXAMPLE` заканчиваются неудачей. СУБД MySQL сообщает, что тип таблицы является тестовым и не поддерживает оператор `INSERT`.

11.5. BDB (BerkeleyDB)

Таблицы типа BDB (сокращение от BerkeleyDB) обслуживаются транзакционным обработчиком Berkeley DB, разработанным компанией Sleepycat. При создании таблиц данного типа формируются два файла: первый с расширением frm, в котором определяется структура базы данных, а второй с расширением db, в котором размещаются данные и индексы.

Далее перечислены особенности таблиц BDB.

- Для каждой из таблиц ведется журнал. Это позволяет значительно повысить устойчивость базы и увеличить вероятность успешного восстановления после сбоя.
- Таблицы BDB хранятся в виде бинарных деревьев (B-деревьев). Такое представление замедляет сканирование таблицы (например, если извлекаются все строки) и увеличивает занимаемое место на жестком диске по сравнению с другими типами таблиц. С другой стороны, поиск отдельных значений в таких таблицах осуществляется быстрее. Так, для сравнения, остальные типы таблиц хранят в виде бинарных деревьев свои индексы.
- Каждая таблица BDB должна иметь первичный ключ, в случае его отсутствия создается скрытый первичный ключ, снабженный атрибутом `AUTO_INCREMENT`.
- Поддерживаются транзакции на уровне страниц.

- Подсчет числа строк в таблице при помощи встроенной функции `COUNT()` осуществляется медленнее, чем для MyISAM-таблиц, т. к. в отличие от последних, для BDB-таблиц не поддерживается подсчет количества строк в таблице, и СУБД MySQL вынуждена каждый раз сканировать таблицу заново.
- Ключи не являются упакованными, как в MyISAM, т. е. ключи в таблицах BDB занимают больше места по сравнению с ключами таблиц MyISAM.
- Если таблица BDB займет все пространство на диске, то будет выведено сообщение об ошибке и выполнен откат транзакции. В отличие от BDB, таблицы MyISAM будут ожидать, пока не появится свободное место, а потом продолжат работу, т. е. произойдет зависание.
- Для обеспечения блокировок таблиц на уровне операционной системы в файл db в момент создания таблицы записывается путь к файлу. Это приводит к тому, что, в отличие от MyISAM-таблиц, файлы нельзя перемещать из текущего каталога в другой.
- При создании резервных копий таблиц BDB необходимо использовать утилиту `mysqldump` или создать резервные копии всех db-файлов и файлов журналов BDB. Файлы журналов BDB — это файлы в каталоге mysql\data с именами log.*XXXXXX* (6 цифр). Обработчик таблицы BDB хранит незавершенные транзакции в файлах журналов, их наличие требуется при запуске сервера MySQL.

Для создания BDB-таблицы необходимо указать в качестве значения параметра `ENGINE` в операторе `CREATE TABLE` константу `BDB` (листинг 11.7).

Листинг 11.7. Создание BDB-таблиц

```
CREATE TABLE t (i INT) ENGINE = BDB;
CREATE TABLE t (i INT) TYPE = BDB;
```

Замечание

В параметрах `ENGINE` и `TYPE` в качестве синонима `BDB` можно применять `BerkeleyDB`.

11.6. InnoDB

Данный тип таблиц обслуживается обработчиком InnoDB, разработанным компанией Innobase Oy. Таблицы этого типа обеспечивают высокую производительность и устойчивое хранение данных в таблицах объемом вплоть до 1 Тбайт и нагрузкой на сервер до 800 вставок/обновлений в секунду.

Замечание

Начиная с версии 4.1.5, автоматическая программа установки MySQL под Windows (Windows installer) может назначить таблицы InnoDB типом по умолчанию.

К особенностям таблиц InnoDB относят следующие.

- В отличие от MyISAM и BDB, таблицы для InnoDB не создаются в базах данных, и для каждой из таблиц не выделяется отдельный файл данных. Исключение — файл определения с расширением frm, который все же создается (по умолчанию в базе данных `test`). Все таблицы хранятся в едином табличном пространстве, поэтому имена таблиц должны быть уникальными.
- Хранение данных в едином табличном пространстве позволяет снять ограничения на объем таблиц, т. к. файл с таблицами может быть разбит на несколько частей и распределен по нескольким дискам или даже хостам.
- Данный тип таблиц поддерживает автоматическое восстановление после сбоев.
- Обеспечивается поддержка транзакций.
- Это единственный тип таблиц в MySQL, поддерживающий внешние ключи и каскадное удаление.
- Выполняется блокировка на уровне отдельных записей.
- Расширенная поддержка кодировок.
- Таблицы типа InnoDB надежнее MyISAM и рушатся при достижении объема в несколько гигабайт, однако заметно уступают в скорости MyISAM (иногда в разы) и не поддерживают полнотекстовый поиск.

Для создания InnoDB-таблицы необходимо указать в качестве значения параметра `ENGINE` в операторе `CREATE TABLE` константу `InnoDB` (листинг 11.8).

Листинг 11.8. Создание InnoDB-таблиц

```
CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A)) ENGINE = InnoDB;
CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A)) TYPE = InnoDB;
```

11.7. NDB Cluster

Тип таблиц `NDB Cluster` предназначен для организации кластеров `MySQL`, когда таблицы распределены между несколькими компьютерами, объединенными в локальную сеть.

Замечание

Тип таблиц NDB Cluster был введен в MySQL, начиная с версии 4.1.2.

11.8. ARCHIVE

Тип таблиц ARCHIVE введен для хранения большого объема данных в сжатом формате. При создании таблиц данного типа MySQL, так же как и для таблиц любого другого типа, создает файл с именем, совпадающим с именем таблицы, и расшире-

нием frm. В этом файле хранится определение структуры таблицы. Помимо этого, создаются два файла с расширениями arz и arm, в которых хранятся данные и метаданные, соответственно. Кроме этого, при оптимизации таблицы может появиться файл с расширением arn.

Замечание

Тип таблиц ARCHIVE был введен в MySQL, начиная с версии 4.1.3.

Замечание

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

ARCHIVE-таблицы поддерживают только два SQL-оператора: `INSERT` и `SELECT`. Причем оператор `SELECT` выполняется по методу полного сканирования таблицы. Любые другие операторы, включая `DELETE`, `UPDATE` и `REPLACE`, не поддерживаются.

11.9. CSV

Формат CSV, представляющий собой обычный текстовый файл, записи в котором хранятся в строках, а поля разделены точкой с запятой, широко распространен в компьютерном мире. Следует отметить, что MS Excel позволяет сохранять таблицы в этом формате.

Замечание

Тип таблиц CSV был введен в MySQL, начиная с версии 4.1.4.

Замечание

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

При создании CSV-таблицы в каталоге с текущей базой данных формируются два файла с именами, совпадающими с именем таблицы, и расширениями frm и csv. В первом файле хранится определение структуры таблицы, а во втором — данные в CSV-формате. Любая другая программа, поддерживающая CSV-формат, например, MS Excel, может открыть данный файл.

В листинге 11.9 приведены SQL-запросы, создающие и заполняющие CSV-таблицу.

Листинг 11.9. Создание CSV-таблицы

```
CREATE TABLE test(i INT, c CHAR(10)) ENGINE = CSV;
INSERT INTO test VALUES(1,'record one'),(2,'record two');
```

Результат выборки при помощи оператора `SELECT` представлен в листинге 11.10.

Листинг 11.10. Выборка из таблицы

```
SELECT * FROM test;
+------+------------+
| i    | c          |
+------+------------+
|    1 | record one |
|    2 | record two |
+------+------------+
2 rows in set (0.00 sec)
```

11.10. FEDERATED

Тип таблиц FEDERATED позволяет хранить данные в удаленных таблицах, расположенных на другой машине сети. 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` (см. листинг 11.11).

Листинг 11.11. 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 11.12).

Листинг 11.12. 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 11.12, параметр ENGINE принимает значение FEDERATED, а в параметре таблицы CONNECTION указываются параметры соединения с удаленным сервером. В результате выполнения SQL-запроса из листинга 11.12 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.

11.11. BLACKHOLE

Таблица типа BLACKHOLE дословно переводится как "черная дыра", ее назначение совпадает с назначением специального вида файла в UNIX-подробных операционных системах — /dev/null. Любые данные, помещаемые в таблицы этого типа, уничтожаются. Основное применение таблицы — для проверки синтаксиса дампов, когда необходимо проверить дамп на наличие ошибок, однако не производить реальное развертывание базы данных. Кроме этого данный тип таблиц удобно использовать для оптимизации производительности и поиска "узких" мест в программах.

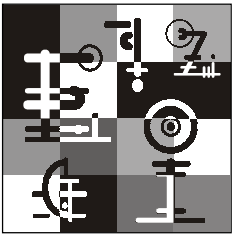
> ***Замечание***
>
> Тип таблиц BLACKHOLE был введен в MySQL, начиная с версии 4.1.1.

Для создания таблиц BLACKHOLE необходимо указать в качестве значения параметра `ENGINE` в операторе `CREATE TABLE` константу `BLACKHOLE` (листинг 11.13).

Листинг 11.13. Создание BLACKHOLE-таблиц

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = BLACKHOLE;
mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql> SELECT * FROM test;
Empty set (0.00 sec)
```

Как видно из листинга 11.13, в отличие от таблицы типа EXAMPLE, при вставке данных при помощи оператора `INSERT` не возникают никакие сообщения об ошибке, однако данные в таблицу не попадают. Тем не менее, в бинарных журналах все операции фиксируются, и в случае сбоя MySQL-сервер даже осуществит попытку восстановить данные. В любом случае таблица останется пустой.



Глава 12

Создание таблиц и удаление таблиц

В *главе 4* уже затрагивался оператор `CREATE TABLE`, позволяющий создать таблицу в текущей базе данных, в данной главе процесс создания новых таблиц будет рассмотрен более подробно. Помимо этого будет также рассмотрен оператор `DROP TABLE`, позволяющий уничтожить таблицу в текущей базе данных.

12.1. Оператор *CREATE TABLE*

Полный синтаксис оператора `CREATE TABLE` выглядит так:

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

> ***Замечание***
>
> Элементы в квадратных скобках являются необязательными и могут быть опущены.

Необязательное ключевое слово `TEMPORARY`: позволяет создавать временные таблицы, которые более подробно рассматриваются в *главе 23*.

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 12.1).

Листинг 12.1. Создание уже существующей таблицы

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

Как видно из листинга 12.1, 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.

> ***Замечание***
>
> Создание таблицы с использованием вложенного запроса *`select_statement`* подробно рассматривается в *главе 23*.

12.1.1. 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. Типы столбцов подробно рассматриваются в *главе 4*, здесь же будут описаны ключевые слова, которые могут следовать за типом столбца.

- `NULL` — означает, что столбец может содержать значение `NULL` (по умолчанию).
- `NOT NULL` — столбец не может содержать значение `NULL`.
- `DEFAULT` *`default_value`* — определяет значение столбца, который 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. повторение в пределах одной таблицы не допускается. 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. Данный параметр подробно рассмотрен в *главе 25*.
- 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).

Рассмотрим несколько примеров, демонстрирующих различные примеры таблиц. В листинге 12.2 представлено определение таблицы пользователей форума.

Листинг 12.2. Оператор CREATE TABLE для создания таблицы authors

```
CREATE TABLE authors (
  id_author INT(6) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name TINYTEXT NOT NULL,
  email TINYTEXT,
  about TEXT,
  photo MEDIUMBLOB,
  puttime DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
  kod INT(10) NOT NULL DEFAULT '20' UNIQUE KEY,
```

```
  id BIGINT(20) NOT NULL DEFAULT '0',
  statususer ENUM('user','moderator','admin') NOT NULL DEFAULT 'user'
);
```

Таблица `authors` состоит из 9 столбцов.

- `id_author` — целочисленный столбец типа `INT`, который снабжен атрибутом `AUTO_INCREMENT` и является первичным ключом таблицы (`PRIMARY KEY`). При вставке новой записи при помощи оператора `INSERT` поле `id_author` можно не заполнять или передавать ему значение `NULL` — MySQL автоматически присвоит ему увеличенное на единицу максимальное значение столбца `id_author`. В таблице больше не может быть первичных ключей и столбцов, снабженных атрибутом `AUTO_INCREMENT`. Кроме того, поле `id_author` обязательно должно быть помечено атрибутом `NOT NULL`.
- `name` — имя пользователя, поле типа `TINYTEXT`, способное принимать в качестве значения строку до 255 символов. Поле помечено атрибутом `NOT NULL`, т. е. не может принимать значение `NULL`. Если значение для данного поля не указывается, оно принимает в качестве значения по умолчанию пустую строку. Как было сказано ранее, применять ключевое слово `DEFAULT` для типов `TEXT/BLOB` нельзя.
- `email` — адрес электронной почты пользователя, поле типа `TINYTEXT`, по умолчанию, если никаких дополнительных параметров ни указывается, оно может принимать значение `NULL`.
- `about` — "несколько слов о себе", поле типа `TEXT`, способное принимать в качестве значения строку до 65 535 символов.
- `photo` — фотография пользователя, поле типа `MEDIUMBLOB`, способное принимать в качестве значения бинарную информацию объемом до 16 777 215 байтов. Поля типа `BLOB`, в отличие от `TEXT`, не имеют кодировки, поэтому данные, помещенные в них, не искажаются автоматическими преобразованиями.
- `puttime` — время регистрации пользователя, поле типа `DATETIME`, в котором хранятся дата и время суток. Ключевое слово `DEFAULT` назначает полю по умолчанию значение `'0000-00-00 00:00:00'`, можно выставить любую другую дату, например, `'2005-12-01 00:00:00'`.
- `kod` — поле типа `INT`, принимающее по умолчанию значение 20. Ключевое слово `UNIQUE KEY` требует, чтобы в пределах таблицы не было двух записей с полями `kod`, имеющими одинаковое значение.
- `id` — поле типа `BIGINT`, принимающее по умолчанию значение 0.
- `statususer` — поле типа `ENUM`, принимающее три значения: `'user'`, `'moderator'` и `'admin'`. Поле данного типа не может иметь значение `NULL`, поэтому снабжается ключевым словом `NOT NULL` и `default`, которое определяет в качестве значения по умолчанию строку `'user'`.

В листинге 12.3 представлен оператор `CREATE TABLE` для таблицы `test`, который демонстрирует применение конструкций `COMMENT` и `CHARACTER SET`.

Листинг 12.3. 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 12.3, таблица `test` состоит из четырех полей типа `TEXT`:

- `dos_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 12.4). Для различных версий их число может быть различным, например, в тестовые версии MySQL обычно не включают национальные кодировки и поддерживаются только кодировки `latain`.

> ***Замечание***
>
> Оператор `SHOW` более подробно рассматривается в *главе 30*.

Листинг 12.4. Результат работы оператора `SHOW CHARACTER SET`

```
mysql> 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 12.4, 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 (см. листинг 12.5). Последний столбец содержит максимальное число байтов, отводимых под одну букву в кодировке.

Листинг 12.5. 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
);
```

> ***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. Более подробно кодировки и сортировки рассматриваются в *главе 14*.

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

Комментарий столбца, 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 12.6).

Листинг 12.6. Просмотр комментария таблицы

```
mysql> 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.

Помимо представленного в листинге 12.3 синтаксиса определения индексов, допускается их определение после перечисления всех столбцов. Такой синтаксис применяется в том случае, если индексу требуется задать имя, отличное от имени столбца или индекс должен быть определен по нескольким столбцам. К ключевым словам для определения индексов, приведенных в начале данного раздела, добавляется ключевое слово `FULLTEXT`, объявление которого допустимо только в конце таблицы (см. листинг 12.7).

Листинг 12.7. Оператор `CREATE TABLE` для создания таблицы `authors`

```
CREATE TABLE authors (
  id_author INT(6) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  email TINYTEXT,
  about TEXT,
  photo MEDIUMBLOB,
  puttime DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
  kod INT(10) NOT NULL DEFAULT '20',
  id BIGINT(20) NOT NULL DEFAULT '0',
  statususer ENUM('user', 'moderator', 'admin') NOT NULL DEFAULT 'user',
  PRIMARY KEY (id_author, id),
  UNIQUE KEY kod (kod),
  FULLTEXT KEY name (name),
  FULLTEXT KEY mailsearch (email),
  FULLTEXT KEY search (about, name)
) TYPE = MyISAM;
```

Как видно из листинга 12.7, оператор `CREATE TABLE` создает таблицу `authors`, аналогичную представленной в листинге 12.2. Отличие заключается в том, что определение ключей вынесено из определения столбцов в конец таблицы — такой синтаксис позволяет переименовать ключи или определить ключи сразу по нескольким столбцам. Так, первичный ключ определен по двум столбцам: `id_author` и `id`, комбинация которых должна быть уникальной. В таблице определены три ключа `FULLTEXT` для полнотекстового поиска по столбцам типа `TEXT`. Первый ключ `name`, позволяет осуществлять поиск по столбцу `name`, второй `mailsearch` по полю `email`, а индекс `search` позволяет выполнять поиск сразу по двум полям: `about` и `name`.

> ***Замечание***
>
> Следует помнить, что ключевое слово `PRIMARY KEY` должно появляться в определении таблицы только один раз: либо в качестве дополнительных ключевых слов столбца, либо в конце таблицы в списке индексов.

> ***Замечание***
>
> Полнотекстовый поиск подробно рассмотрен в *главе 20*.

12.1.2. Параметры таблицы

Параметры таблицы, обозначенные в схеме оператора `CREATE TABLE` как *`table_options`*, были введены в MySQL, начиная с версии 3.23. Далее приводятся ключевые слова, которые могут быть перечислены после структуры 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 12.8.

Листинг 12.8. 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. 12.1.

Таблица 12.1. *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

Замечание

Более подробно типы таблиц описаны в *главе 11*.

AUTO_INCREMENT

Параметр AUTO_INCREMENT позволяет задать число, начиная с которого будет происходить увеличение счетчика для столбца, снабженного этим атрибутом (листинг 12.9).

Листинг 12.9. 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.

AVG_ROW_LENGTH

Приблизительная средняя длина записи для таблиц типа MyISAM. Установка данного параметра имеет смысл только для таблиц с записями переменной длины, т. е. содержащих типы столбцов VARCHAR, TEXT, BLOB и т. п.

Во время создания таблицы MySQL оценивает ее размер по произведению AVG_ROW_LENGTH и максимального числа записей MAX_ROWS в таблице. Чем больше таблица, тем большую разрядность должен иметь дескриптор для адресации данных в таблице. Так по умолчанию применяется 32-разрядный дескриптор (4 байта), который позволяет адресовать 4 Гбайт данных. Именно на столько рассчитана по умолчанию таблица MyISAM, если не указаны параметры AVG_ROW_LENGTH и MAX_ROWS. Если по оценке MySQL планируемый размер таблицы превысит 4 Гбайт, будет автоматически выбран дескриптор большей разрядности.

[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 12.10.

Листинг 12.10. 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 12.10 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 12.11). Это приводит к небольшому снижению скорости обновления таблицы, но значительно повышает вероятность ее восстановления в случае сбоя.

Листинг 12.11. 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 12.12).

Листинг 12.12. Использование параметра `COMMENT`

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

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

CONNECTION

Параметр `CONNECTION` используется для указания параметров соединения с удаленным сервером в таблицах типа FEDERATED *(см. разд. 11.10)*. До версии 5.0.13 вместо параметра `CONNECTION` для этих целей использовался параметр `COMMENT`.

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 12.13).

Листинг 12.13. 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.

DELAY_KEY_WRITE

Параметр `DELAY_KEY_WRITE` принимает значение либо 1, либо 0. Если установлено значение 1, кэш индекса периодически обновляется для всей таблицы, а не после каждой операции вставки. Данный параметр применяется только для таблиц типа MyISAM и игнорируется для таблиц другого типа.

INDEX DIRECTORY

Параметр `INDEX DIRECTORY` используется только для таблиц типа MyISAM и определяет каталог, куда помещается индексный файл (myi). Путь должен быть абсолютным (листинг 12.14).

Листинг 12.14. 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.

INSERT_METHOD

Данный параметр используется в таблицах типа MERGE для определения правил добавления строк. Подробное описание данного параметра приведено в *разд. 11.2*.

MAX_ROWS

Параметр MAX_ROWS определяет максимально число записей, которое планируется хранить в таблице (листинг 12.15). Использование данного параметра имеет смысл только в таблицах MyISAM.

Листинг 12.15. Использование параметра MAX_ROWS

```
CREATE TABLE test (
  name TINYTEXT
) ENGINE = MyISAM MAX_ROWS = 10000000 AVG_ROW_LENGTH = 100;
```

Следует отметить, что параметр MAX_ROWS не ограничивает число записей в таблице. Он лишь служит для подсказки оптимизатору MySQL, как наиболее эффективно хранить данные.

MIN_ROWS

Параметр MIN_ROWS определяет минимальное число записей, которое планируется хранить в таблице. Использование данного параметра имеет смысл только в таблицах MyISAM и HEAP. Точно так же, как и MAX_ROWS, данный параметр не несет ограничительных функций и служит лишь для оценки объема таблицы.

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.

PASSWORD

Данная опция, показанная в листинге 12.16, задает пароль для шифрования файла описания таблицы (с расширением frm).

Листинг 12.16. Использование параметра PASSWORD

```
CREATE TABLE test (
  name TINYTEXT
) ENGINE = MyISAM PASSWORD = 'root';
```

К сожалению, в текущих версиях MySQL данная опция лишь зарезервирована, но не работает.

ROW_FORMAT

Параметр ROW_FORMAT применим только к таблицам типа MyISAM и определяет способ хранения записей (листинг 12.17). 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 12.17. Использование параметра ROW_FORMAT

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

UNION

Параметр UNION определяет список таблиц MyISAM, которые составляют MERGE-таблицу. Синтаксис данного параметра рассмотрен в *разд. 11.2*.

12.2. Оператор DROP TABLE

Оператор DROP TABLE предназначен для удаления таблиц и имеет следующий синтаксис:

```
DROP [TEMPORARY] TABLE [IF EXISTS]
     tbl_name [, tbl_name] ...
     [RESTRICT | CASCADE]
```

> **Замечание**
>
> Элементы в квадратных скобках являются необязательными и могут быть опущены.

Оператор позволяет удалить одну или более таблиц *tbl_name*, так в листинге 12.18 первый оператор DROP TABLE удаляет таблицу catalogs, а второй оператор — все остальные таблицы учебной базы данных shop.

> **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 12.18. Использование оператора `DROP TABLE`

```
mysql> DROP TABLE catalogs;
mysql> DROP TABLE products, orders, users;
```

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 12.19).

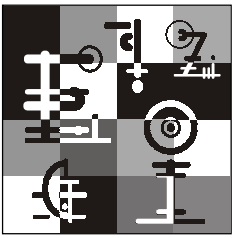
Листинг 12.19. Использование ключевого слова `IF EXISTS` совместно с `DROP TABLE`

```
mysql> DROP TABLE catalogs;
ERROR 1051 (42S02): Unknown table 'catalogs'
mysql> 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 *главе 23*.

Ключевые слова `RESTRICT` и `CASCADERESTRICT` и `CASCADE` введены для дальнейших расширений и облегчения переносимости SQL-запросов из других СУБД. В текущих версиях MySQL данные ключевые слова ничего не делают.



Глава 13

Редактирование структуры таблиц

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`, которому посвящена данная глава.

Оператор `ALTER TABLE` 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. Более подробное описание оператора приведено в соответствующих разделах данной главы.

> ***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.

13.1. Добавление и удаление столбцов

Для добавления столбца указывается спецификатор оператора `ALTER TABLE — ADD [COLUMN]`, использование которого демонстрируется в листинге 13.1, где в таблицу `products` (см. листинг 4.26) учебной базы данных `shop` добавляется целочисленный столбец `rebate`, в который можно поместить скидку на товар в процентах.

Замечание

Ключевое слово COLUMN является необязательным и может быть опущено как 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 13.1. Добавление нового столбца `rebate` в таблицу `products`

```
mysql> DESCRIBE products;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| id_product  | int(11)      | NO   |     |         |       |
| name        | tinytext     | NO   |     |         |       |
| price       | decimal(7,2) | YES  |     | 0.00    |       |
| count       | int(11)      | YES  |     | 0       |       |
| mark        | float(4,1)   | NO   |     | 0.0     |       |
| description | text         | YES  |     | NULL    |       |
| id_catalog  | int(11)      | NO   |     |         |       |
+-------------+--------------+------+-----+---------+-------+
mysql> ALTER TABLE products ADD COLUMN rebate INT;
mysql> DESCRIBE products;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| id_product  | int(11)      | NO   |     |         |       |
| name        | tinytext     | NO   |     |         |       |
| price       | decimal(7,2) | YES  |     | 0.00    |       |
| count       | int(11)      | YES  |     | 0       |       |
| mark        | float(4,1)   | NO   |     | 0.0     |       |
| description | text         | YES  |     | NULL    |       |
| id_catalog  | int(11)      | NO   |     |         |       |
| rebate      | int(11)      | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
```

Удаление столбца производится при помощи спецификатора DROP [COLUMN] оператора ALTER TABLE. В листинге 13.2 демонстрируется удаление вновь созданного столбца rebate.

Листинг 13.2. Удаление столбца `rebate` из таблицы `products`

```
mysql> ALTER TABLE products DROP COLUMN rebate;
mysql> DESCRIBE products;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| id_product  | int(11)      | NO   |     |         |       |
| name        | tinytext     | NO   |     |         |       |
| price       | decimal(7,2) | YES  |     | 0.00    |       |
| count       | int(11)      | YES  |     | 0       |       |
| mark        | float(4,1)   | NO   |     | 0.0     |       |
| description | text         | YES  |     | NULL    |       |
| id_catalog  | int(11)      | NO   |     |         |       |
+-------------+--------------+------+-----+---------+-------+
```

Замечание

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

Как видно из листинга 13.1, 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 13.3 демонстрируется добавление столбца `id_first` в начало таблицы и столбца `rebate` — после `price`.

Листинг 13.3. Добавление сразу двух столбцов

```
mysql> ALTER TABLE products ADD id_first INT FIRST,
    -> ADD rebate INT AFTER price;
mysql> DESCRIBE products;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| id_first    | int(11)      | YES  |     | NULL    |       |
| id_product  | int(11)      | NO   |     |         |       |
| name        | tinytext     | NO   |     |         |       |
| price       | decimal(7,2) | YES  |     | 0.00    |       |
| rebate      | int(11)      | YES  |     | NULL    |       |
```

```
| count       | int(11)      | YES  |     | 0       |       |
| mark        | float(4,1)   | NO   |     | 0.0     |       |
| description | text         | YES  |     | NULL    |       |
| id_catalog  | int(11)      | NO   |     |         |       |
+-------------+--------------+------+-----+---------+-------+
```

13.2. 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 спецификация `MODIFY`, после которой указывается имя модифицируемого столбца и его новый тип. В листинге 13.4 целочисленный тип столбца `id_first`, добавленного в листинге 13.3, меняется на текстовый.

Листинг 13.4. Изменение типа столбца `id_first`

```
mysql> ALTER TABLE products MODIFY id_first TEXT;
mysql> DESCRIBE products;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| id_first    | text         | YES  |     | NULL    |       |
| id_product  | int(11)      | NO   |     |         |       |
| name        | tinytext     | NO   |     |         |       |
| price       | decimal(7,2) | YES  |     | 0.00    |       |
| rebate      | int(11)      | YES  |     | NULL    |       |
| count       | int(11)      | YES  |     | 0       |       |
| mark        | float(4,1)   | NO   |     | 0.0     |       |
| description | text         | YES  |     | NULL    |       |
| id_catalog  | int(11)      | NO   |     |         |       |
+-------------+--------------+------+-----+---------+-------+
```

Точно так же, как и при создании столбца, использование ключевых слов `FIRST` и `AFTER` позволяет изменить порядок следования столбцов. В листинге 13.5 столбцы `id_first` и `rebate` меняются местами.

Листинг 13.5. Изменение порядка следования столбцов

```
mysql> ALTER TABLE products MODIFY id_first TEXT AFTER price,
    -> MODIFY rebate INT FIRST;
mysql> DESCRIBE products;
```

```
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| rebate      | int(11)      | YES  |     | NULL    |       |
| id_product  | int(11)      | NO   |     |         |       |
| name        | tinytext     | NO   |     |         |       |
| price       | decimal(7,2) | YES  |     | 0.00    |       |
| id_first    | text         | YES  |     | NULL    |       |
| count       | int(11)      | YES  |     | 0       |       |
| mark        | float(4,1)   | NO   |     | 0.0     |       |
| description | text         | YES  |     | NULL    |       |
| id_catalog  | int(11)      | NO   |     |         |       |
+-------------+--------------+------+-----+---------+-------+
```

Спецификация `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 13.6 столбец `id_first` переименовывается в `id_second`.

Листинг 13.6. Переименование столбца

```
mysql> ALTER TABLE products CHANGE id_first id_second TEXT;
mysql> DESCRIBE products;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| rebate      | int(11)      | YES  |     | NULL    |       |
| id_product  | int(11)      | NO   |     |         |       |
| name        | tinytext     | NO   |     |         |       |
| price       | decimal(7,2) | YES  |     | 0.00    |       |
| id_second   | text         | YES  |     | NULL    |       |
| count       | int(11)      | YES  |     | 0       |       |
| mark        | float(4,1)   | NO   |     | 0.0     |       |
| description | text         | YES  |     | NULL    |       |
| id_catalog  | int(11)      | NO   |     |         |       |
+-------------+--------------+------+-----+---------+-------+
```

Как и в случае `MODIFY`, совместно со спецификацией `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. В листинге 13.7 решается задача смены типа столбца id_second с TEXT на INT, а типа столбца rebate с INT на TEXT, при этом столбцы размещаются в конце таблицы.

Листинг 13.7. Использование ключевого слова CHANGE

```
mysql> ALTER TABLE products
    -> CHANGE id_second id_second INT AFTER id_catalog,
    -> CHANGE rebate rebate TEXT AFTER id_catalog;
mysql> DESCRIBE products;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| id_product  | int(11)      | NO   |     |         |       |
| name        | tinytext     | NO   |     |         |       |
| price       | decimal(7,2) | YES  |     | 0.00    |       |
| count       | int(11)      | YES  |     | 0       |       |
| mark        | float(4,1)   | NO   |     | 0.0     |       |
| description | text         | YES  |     | NULL    |       |
| id_catalog  | int(11)      | NO   |     |         |       |
| rebate      | text         | YES  |     | NULL    |       |
| id_second   | int(11)      | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
```

13.3. Добавление и удаление индексов

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 20*.

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 13.8 создается первичный ключ для таблицы `products` (см. листинг 4.26) и индексируется столбец `id_catalog` с созданием индекса `id_catalog_index`.

Листинг 13.8. Создание индексов при помощи оператора `ALTER TABLE`

```
mysql> ALTER TABLE products ADD PRIMARY KEY (id_product),
    -> ADD INDEX id_catalog_index (id_catalog);
mysql> DESCRIBE products;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| id_product  | int(11)      | NO   | PRI |         |       |
| name        | tinytext     | NO   |     |         |       |
| price       | decimal(7,2) | YES  |     | 0.00    |       |
| count       | int(11)      | YES  |     | 0       |       |
| mark        | float(4,1)   | NO   |     | 0.0     |       |
| description | text         | YES  |     | NULL    |       |
| id_catalog  | int(11)      | NO   | MUL |         |       |
+-------------+--------------+------+-----+---------+-------+
```

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 отдельная 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 13.9 происходит удаление индексов, созданных SQL-запросом из листинга 13.8.

Листинг 13.9. Удаление индексов при помощи оператора `ALTER TABLE`

```
mysql> ALTER TABLE products DROP PRIMARY KEY,
    -> DROP INDEX id_catalog_index;
mysql> DESCRIBE products;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| id_product  | int(11)      | NO   |     |         |       |
| name        | tinytext     | NO   |     |         |       |
| price       | decimal(7,2) | YES  |     | 0.00    |       |
| count       | int(11)      | YES  |     | 0       |       |
| mark        | float(4,1)   | NO   |     | 0.0     |       |
| description | text         | YES  |     | NULL    |       |
| id_catalog  | int(11)      | NO   |     |         |       |
+-------------+--------------+------+-----+---------+-------+
```

13.4. Преобразование параметров таблицы

Помимо изменения структуры таблицы, оператор `ALTER TABLE` позволяет изменять параметры таблицы, например, ее название. Для этого используется конструкция `RENAME [TO] new_table`, в результате применения которой таблица получает новое имя *new_table*. В листинге 13.10 таблица `products` базы данных `shop` переименовывается в `components`.

> ***Замечание***
>
> Необязательное ключевое слово `TO` в спецификации `RENAME [TO]` может быть опущено.

Листинг 13.10. Переименование таблицы `products` в `components`

```
mysql> SHOW TABLES;
+---------------+
| Tables_in_shop |
+---------------+
| catalogs       |
| orders         |
| products       |
| users          |
+---------------+
mysql> ALTER TABLE products RENAME TO components;
```

```
mysql> SHOW TABLES;
+---------------+
| Tables_in_shop |
+---------------+
| catalogs       |
| components     |
| orders         |
| users          |
+---------------+
```

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 13.11 таблицы `users` и `orders` меняются именами.

Листинг 13.11. Обмен именами таблиц `users` и `orders`

```
mysql> RENAME TABLE users TO backup_table,
    -> orders TO users,
    -> backup_table TO orders;
mysql> DESCRIBE users;
+------------+----------+------+-----+------------+----------------+
| Field      | Type     | Null | Key | Default    | Extra          |
+------------+----------+------+-----+------------+----------------+
| id_order   | int(11)  | NO   | PRI | NULL       | auto_increment |
| id_user    | int(11)  | NO   | MUL | 0          |                |
| ordertime  | datetime | NO   |     | 0000-00-00 |                |
| number     | int(11)  | NO   |     | 0          |                |
| id_product | int(11)  | NO   | MUL | 0          |                |
+------------+----------+------+-----+------------+----------------+
```

Так как в одной базе данных не может быть двух таблиц с одинаковыми именами, в листинге 13.11 таблице `users` потребовалось назначить временное имя `backup_table`, которое потом заменяется на `orders`.

Спецификация `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 13.12 записи таблицы `catalogs` сортируются по полю `name`.

Листинг 13.12. Сортировка записей таблицы `catalogs` по полю `name`

```
mysql> SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
mysql> ALTER TABLE catalogs ORDER BY name;
mysql> 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. Спецификация `ORDER BY` в основном полезна, 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 13.13 для таблицы `catalogs` выставляется значение `AUTO_INCREMENT`, равное 1000, в результате чего добавление нового значения приведет к тому, что первичный ключ таблицы `id_catalog` получит значение 1000.

> ***Замечание***
>
> Полный список параметров и их назначение рассматривается в *главе 12*, каждый из них можно изменять так же, как и параметр `AUTO_INCREMENT`.

Листинг 13.13. Изменение значений параметра `AUTO_INCREMENT`

```
mysql> ALTER TABLE catalogs AUTO_INCREMENT = 1000;
mysql> INSERT INTO catalogs () VALUES ();
mysql> 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 13.14.

Листинг 13.14. Изменение значения параметра `[DEFAULT] CHARACTER SET`

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

Как видно из листинга 13.14, помимо кодировки, можно указать тип сортировки (сопоставления), который задается ключевым словом `COLLATE`. Так в последнем SQL-запросе для таблицы `users` указывается сортировка кодировки `cp1251` с учетом регистра.

Однако параметр `[DEFAULT] CHARACTER SET` изменяет кодировку и сортировку только для самой таблицы, если необходимо 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 13.14 следует переписать так, как это представлено в листинге 13.15.

Замечание

Более подробно кодировки и сортировки рассматриваются в *главе 14*.

Листинг 13.15. Изменение кодировки и сортировки для всей таблицы, включая столбцы

```
mysql> ALTER TABLE catalogs CONVERT TO CHARACTER SET cp1251;
mysql> ALTER TABLE products CONVERT TO CHARACTER SET cp1251;
mysql> ALTER TABLE users CONVERT TO CHARACTER SET cp1251
    ->                   COLLATE cp1251_general_cs;
```

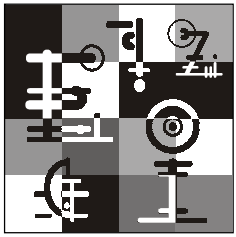
Еще два спецификатора `DISABLE KEYS` и `ENABLE KEYS` оператора `ALTER TABLE` позволяют управлять актуальностью неуникальных индексов таблицы. Спецификатор `DISABLE KEYS` отключает обновление индексов при добавлении и удалении записей из таблицы, а спецификатор `ENABLE KEYS` восстанавливает режим обновления. Использование данных спецификаторов необходимо в том случае, когда требуется ускорить процесс вставки или удаления записей таблицы при помощи операторов `INSERT` и `DELETE` (листинг 13.16). Если отключен режим автоматического обновления индексов, СУБД MySQL не будет тратить время на сортировку индексов.

Замечание

Подробнее индексы рассматриваются в *главе 5*.

Листинг 13.16. Использование спецификаторов `DISABLE KEYS` и `ENABLE KEYS`

```
mysql> ALTER TABLE products DISABLE KEYS;
mysql> -- многочисленные операции вставки новых значений
mysql> ALTER TABLE products ENABLE KEYS;
```



Глава 14

Приведение типов

В этой главе рассматривается приведение типов данных в СУБД MySQL, а также встроенные функции, предназначенные для приведения типов. В большинстве случаев приведение типов осуществляется автоматически (листинг 14.1).

Листинг 14.1. Автоматическое приведение типов

```
mysql> SELECT 1 + '1';
+---------+
| 1 + '1' |
+---------+
|       2 |
+---------+
1 row in set (0.01 sec)
```

Тем не менее, отдельные ситуации требуют специального приведения типов.

14.1. Ключевое слово *BINARY*

Сравнение текстовых строк в MySQL не зависит от регистра сопоставляемых строк. Это означает, что строки "Hello" и "HELLO" с точки зрения MySQL являются идентичными (листинг 14.2).

Листинг 14.2. Сравнение текстовых строк

```
mysql> SELECT "Hello" = "HELLO";
+-------------------+
| "Hello" = "HELLO" |
+-------------------+
```

```
|                 1 |
+-------------------+
1 row in set (0.02 sec)
```

Ключевое слово `BINARY` позволяет сравнивать строки как двоичные последовательности. В этом случае строки будут равны только при условии полной эквивалентности (листинг 14.3).

Листинг 14.3. Двоичное сравнение строк

```
mysql> SELECT BINARY "Hello" = "HELLO";
+--------------------------+
| BINARY "Hello" = "HELLO" |
+--------------------------+
|                        0 |
+--------------------------+
1 row in set (0.01 sec)
```

Использование ключевого слова `BINARY` допускается как слева от оператора сравнения:

```
SELECT BINARY "Hello" = "HELLO";
```

так и справа:

```
SELECT "Hello" = BINARY "HELLO";
```

Аналогичного результата можно добиться, если в качестве типа столбцов использовать `BLOB`.

14.2. Функция *CAST()*

Встроенная функция `CAST()` предназначена для преобразования выражения из одного типа в другой. Функция имеет следующий синтаксис:

```
CAST(expr AS type)
```

Результатом работы функции является преобразование выражения *expr* в тип *type*. Значением *type* может быть одно из следующих:

- `BINARY` (двоичная строка);
- `CHAR`;
- `DATE`;
- `DATETIME`;
- `SIGNED [INTEGER]`;

- `TIME`;
- `UNSIGNED [INTEGER]`.

В листинге 14.4 продемонстрирована работа функции `CAST()`.

Листинг 14.4. Функция `CAST()`

```
mysql> SELECT 20051201 AS first, CAST(20051201 AS DATE) AS second;
+----------+------------+
| first    | second     |
+----------+------------+
| 20051201 | 2005-12-01 |
+----------+------------+
1 row in set (0.00 sec)
```

Замечание

Между именем функции и открывающей круглой скобкой не должно быть пробела. Это связано с тем, что имена функций не являются зарезервированными словами и могут выступать в качестве имени столбца. В такой ситуации MySQL может отличить столбец от функции, только если за именем функции сразу следует открывающая круглая скобка.

Замечание

Конструкция `CAST(`*`str`* `AS BINARY)` аналогична конструкции `BINARY` *`str`*.

Функция `CAST()` часто применяется для сортировки полей типа `ENUM` не по численному значению, а по алфавиту. Возвращаясь к учебной базе данных, отсортируем поля таблицы `users` по полю `status` (листинг 14.5).

Листинг 14.5. Сортировка по `ENUM`-полю `status`

```
mysql> SELECT surname, status FROM users ORDER BY CAST(status AS CHAR);
+-----------+---------+
| surname   | status  |
+-----------+---------+
| Иванов    | active  |
| Симдянов  | active  |
| Кузнецов  | active  |
| Корнеев   | gold    |
| Нехорошев | lock    |
| Лосев     | passive |
+-----------+---------+
6 rows in set (0.00 sec)
```

Еще один пример использования функции `CAST()` приведен в листинге 14.6.

Листинг 14.6. Преобразование текущего времени в формат `DATE`

```
mysql> SELECT NOW(), CAST(NOW() AS DATE);
+---------------------+---------------------+
| NOW()               | CAST(NOW() AS DATE) |
+---------------------+---------------------+
| 2005-04-10 02:01:47 | 2005-04-10          |
+---------------------+---------------------+
1 row in set (0.00 sec)
```

В листинге 14.6 демонстрируется приведение текущего времени в полном формате, возвращаемого функцией `NOW()` к усеченному формату типа `DATE`.

14.3. Функция *CONVERT()*

Встроенная функция `CONVERT()` имеет следующий синтаксис:

`CONVERT(`*`expr`*`, `*`type`*`)`

`CONVERT(`*`expr`* `USING` *`charset`*`)`

Первая форма функции `CONVERT()` аналогична по действию функции `CAST()`. Функция `CAST()` является стандартной функцией SQL и поддерживается другими базами данных. Функция `CONVERT()` в первом варианте (без `USING`) является функцией интерфейса ODBC, который получил широкое распространение. Аргументы *`expr`* и *`type`* имеют то же значение, что и для функции `CAST()`.

Вторая форма функции `CONVERT()` (с использованием `USING`) предназначена для преобразования текста из одной кодировки в другую (листинг 14.7).

Листинг 14.7. Преобразование кодировки

```
SELECT CONVERT('База данных MySQL' USING koi8r);
+----------------------------------------+
| CONVERT('База данных MySQL' USING koi8r) |
+----------------------------------------+
| аЮГЮ ДЮММШУ MySQL                      |
+----------------------------------------+
1 row in set (0.00 sec)
```

Иногда преобразовывать строку не требуется, а необходимо просто указать ее кодировку, это осуществляется при помощи так называемого *представителя* (листинг 14.8).

Замечание

Представители рассматриваются подробнее в *разд. 14.5*.

Листинг 14.8. Альтернативное преобразование кодировки

```
mysql> SELECT _koi8r'База данных MySQL';
+------------------+
| аЮГЮ ДЮММШУ MySQL |
+------------------+
| аЮГЮ ДЮММШУ MySQL |
+------------------+
1 row in set (0.00 sec)
```

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.

14.4. Поддержка кодировок

Начиная с версии 4.1, MySQL предоставляет расширенную поддержку кодировок. Существует несколько наборов символов, один набор символов может поддерживать несколько языков (например, в случае `latin1` поддерживается несколько европейских языков). Однако для полноценной поддержки языка следует определить порядок соответствия кодов-символов. В табл. 14.1 приводятся ASCII-коды английского алфавита.

Как видно из табл. 14.1, буквы в верхнем регистре соответствуют ASCII-кодам от 65 до 90, а в нижнем регистре — от 97 до 122. Такое расположение символов достаточно удобно, особенно при сравнении строк, т. к. можно просто сравнивать ASCII-коды каждого из символов. Так, например, слово "hello" будет больше "allow", поскольку символ "h" имеет код 104, а символ "a" — 97, таким образом, "a" < "h". Именно благодаря тому, что символам поставлены в соответствие ASCII-коды, СУБД имеет возможность производить корректную сортировку по алфавиту. Сравнение строк по их ASCII-кодам называется *бинарным сравнением*.

Замечание

Первоначально ASCII-символы применялись для передачи текста в телеграфии и кодировались 7-ю битами (8-й бит байта использовался для контроля четности), что позволяет закодировать 128 символов, которых вполне хватало на английский язык и управляющие символы. В связи с развитием компьютеров и сетей, появилась потребность кодировать алфавиты других языков, поэтому таблица ASCII была расширена за счет восьмого бита до 256 символов. Поэтому английские символы кодируются кодами до 128, а русские после 128.

Таблица 14.1. *ASCII-коды английского алфавита*

Символ	ASCII-код	Символ	ASCII-код
A	65	a	97
B	66	b	98
C	67	c	99
D	68	d	100
E	69	e	101
F	70	f	102
G	71	g	103
H	72	h	104
I	73	i	105
J	74	j	106
K	75	k	107
L	76	l	108
M	77	m	109
N	78	n	110
O	79	o	111
P	80	p	112
Q	81	q	113
R	82	r	114
S	83	s	115
T	84	t	116
U	85	u	117
V	86	v	118
W	87	w	119
X	88	x	120
Y	89	y	121
Z	90	z	122

Не все языки можно закодировать при помощи монотонно-возрастающих кодов, например, русский алфавит в кодировке KOI8-R выглядит так, как это представлено в табл. 14.2.

Таблица 14.2. *Коды русского алфавита в KOI8-R*

Символ	ASCII-код	Символ	ASCII-код
А	225	а	193
Б	226	б	194
В	247	в	215
Г	231	г	199
Д	228	д	196
Е	229	е	197
Ё	179	ё	163
Ж	246	ж	214
З	250	з	218
И	233	и	201
Й	234	й	202
К	235	к	203
Л	236	л	204
М	237	м	205
Н	238	н	206
О	239	о	207
П	240	п	208
Р	242	р	210
С	243	с	211
Т	244	т	212
У	245	у	213
Ф	230	ф	198
Х	232	х	200
Ц	227	ц	195
Ч	254	ч	222
Ш	251	ш	219
Щ	253	щ	221
Ъ	255	ъ	223
Ы	249	ы	217
Ь	248	ь	216
Э	252	э	220
Ю	224	ю	192
Я	241	я	209

Как видно из табл. 14.2, русский алфавит в кодировке KOI8-R закодирован не монотонно, и при бинарном сравнении отсортировать русский текст уже не получится, требуется дополнительная таблица преобразования, которая сообщит СУБД, что символ "а" меньше символа "б", а "б" в свою очередь меньше символа "в". Кроме того, даже для английского алфавита в том случае, если не учитывается регистр символа, необходимо составлять дополнительную таблицу, чтобы показать СУБД, что A = a <= B = b <= C = c <= ...

Для этих целей используются сортировки, которые уже встречались нам и обозначались во всех операторах ключевым словом `COLLATION`. Одна кодировка может использоваться сразу для нескольких языков. Сортировка определяет порядок сопоставления символов, одна кодировка может иметь несколько сортировок, но одна сортировка всегда назначается по умолчанию. Так в листинге 12.5 извлекается список всех доступных для использования в СУБД MySQL кодировок, третий столбец указывает, какой тип сортировки назначен по умолчанию для каждой из кодировок.

При помощи оператора `SHOW COLLATION` можно определить полный список сортировок. По умолчанию оператор выводит большое количество информации, поэтому уменьшим его при помощи ключевого слова `LIKE`, ограничив вывод только кодировкой `cp1251`, которая соответствует русской Windows-кодировке (листинг 14.9).

> ***Замечание***
>
> Синтаксис оператора `SHOW` более подробно рассматривается в *главе 30*.

Листинг 14.9. Сортировки, применяемые совместно с кодировкой `latin1`

```
mysql> 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 14.9, кодировка `cp1251` обладает 5-ю сортировками, которые имеют следующие значения:

- `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).

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 14.9). 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 14.10.

Листинг 14.10. 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 14.11, т. к. в противном случае будет выставлена кодировка по умолчанию и все операции сортировки текста будут осуществляться некорректно.

Листинг 14.11. 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.

Приоритет увеличивается сверху вниз по списку, 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 14.10), то не смотря 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`. Впрочем, можно установить кодировку и сортировку на уровне столбца, как это демонстрируется в листинге 12.3. Работа с кодировками и сортировками на уровне базы данных, таблицы и столбцов рассматривались в предыдущих главах. Здесь будут рассмотрены кодировки на уровне сервера и соединения, т. к. именно это вызывает множество вопросов.

Сразу после установки, если не проводились дополнительные настройки сервера, на уровне сервера устанавливается кодировка `latin1` со шведской кодировкой `latin1_swedish_ci`. Это означает, что при создании баз данных и таблиц, если явно не указывать кодировку и сортировку при помощи ключевых слов `CHARACTER SET` и `COLLATION`, то по умолчанию будет выставляться кодировка `latin1` со шведской сортировкой. Для того чтобы переопределить кодировку на уровне сервера, необходимо при запуске серверу передать параметр `--default-character-set` с указанием кодировки. При необходимости при помощи параметра `--default-collation` можно указать сортировку. В листинге 14.12 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 14.12. 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 прописать в конфигурационном файле my.cnf (my.ini) в секции `[mysqld]` (листинг 14.13).

> ***Замечание***
>
> Конфигурационный файл my.cnf в Windows располагается в папке C:\Windows или в корне диска C:\, в операционной системе Linux при установке бинарных файлов конфигурационный файл находится в папке /etc.

Листинг 14.13. Установка кодировки и сортировки в конфигурационном файле my.ini

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

При указании сортировки следует следить, чтобы она соответствовала кодировке. Если совместно с кодировкой будет указана сортировка из другого кодового набора, север 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 и порядок работы с ними более подробно рассматриваются в *главе 30*.

При установке соединения клиента с сервером последний ожидает от клиента запросы в кодировке `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`.

> ***Замечание***
>
> Преобразованию из `character_set_client` в `character_set_connection` не подвергаются лишь строки, представленные при помощи модификаторов `_latin1` или `_utf8`.

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 14.14.

Листинг 14.14. Установка системных переменных для работы с русским текстом

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

Вместо трех операторов, представленных в листинге 14.14, можно воспользоваться специальным оператором `SET NAMES 'имя_кодировки'`. Оператор в листинге 14.15 по действию полностью эквивалентен операторам из листинга 14.14.

Листинг 14.15. Использование оператора SET NAMES

```
mysql> SET NAMES 'cp1251';
```

Помимо оператора `SET NAMES`, существует схожий оператор `SET CHARACTER SET`, использование которого демонстрируется в листинге 14.16.

Листинг 14.16. Использование оператора SET CHARACTER SET

```
mysql> 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. Забегая вперед, можно сказать, что листинг 14.16 полностью эквивалентен листингу 14.17.

Листинг 14.17. Установка системных переменных для работы с русским текстом

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

При работе с консольным клиентом `mysql` нет надобности всякий раз устанавливать кодировку при помощи оператора `SET NAMES`, т. к. при вызове клиента можно передать ему кодировку посредством параметра `--default-character-set` *(см. разд. 3.1)*. Более того, можно избежать и этого, добавив в конфигурационный файл my.ini опцию `default-character-set` в раздел `[mysql]`. Например, конфигурационный файл my.ini, представленный в листинге 14.13, можно переписать так, как это представлено в листинге 14.18.

Листинг 14.18. Установка кодировки и сортировки в конфигурационном файле my.ini

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

Если кодировкой соединения не управлять, то в результирующих таблицах вместо русского текста будут знаки вопроса (листинг 14.19).

Листинг 14.19. Искажение результатов из-за неправильно выбранной кодировки

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

14.5. Преобразование кодировок

После обстоятельного рассмотрения кодировок можно вернуться к указанию кодировки для строкового литерала, пример которого приводится в листинге 14.8. Такого рода выражения называют *представителем*. Его полная форма такова:

```
[_имя_кодировки]'строка' [COLLATE имя_сортировки]
```

Строку предваряет название кодировки с символом подчеркивания в качестве префикса. После строки при помощи ключевого слова `COLLATE` может быть указана сортировка. В листинге 14.20 приводится пример использования различных форм представителя.

Листинг 14.20. Использование представителя

```
mysql> SELECT 'База данных MySQL';
mysql> SELECT _cp1251'База данных MySQL';
mysql> SELECT _cp1251'База данных MySQL' COLLATE cp1251_general_ci;
```

Представитель позволяет явно указать кодировку строки, следующей за ним. Нужно помнить, что представитель не осуществляет преобразований, а лишь указывает серверу, в какой кодировке следует воспринимать строку. Разумеется, если потом она будет читаться в другой кодировке — получится эффект преобразования, как это демонстрируется в листинге 14.8, где указывается, что строка имеет кодировку `koi8-r`, но т. к. системная переменная `character_set_results` имеет значение `'cp1251'`, результат искажается.

14.6. Применение ключевого слова *COLLATE*

Ключевое слово `COLLATE` может применяться не только совместно с представителем, но и совместно с рядом других ключевых слов, например, `ORDER BY` (листинг 14.21).

Листинг 14.21. Использование ключевого слова `COLLATE` совместно с `ORDER BY`

```
mysql> SELECT * FROM catalogs
    -> ORDER BY name COLLATE cp1251_ukrainian_ci;
```

Использование ключевого слова, таким образом, особенно удобно, если в одной таблице хранятся одновременно и русские, и украинские строки. Например, можно добавить в каждую таблицу столбец, который будет определять предпочитаемый пользователем язык. Если пользователь укажет русский язык, нужно выбирать только русские записи, если украинский — украинские. При этом смена сортировки при помощи ключевого слова `COLLATE` позволит корректно сортировать записи для каждого из языков.

Ключевое слово `COLLATE` не обязательно помещать непосредственно в конструкцию `ORDER BY`, можно переопределить сортировку столбца, назначить ему при помощи оператора `AS` псевдоним и передать псевдоним конструкции `ORDER BY`, как это демонстрируется в листинге 14.22.

Листинг 14.22. Использование ключевого слова `COLLATE` совместно с `AS`

```
mysql> SELECT name COLLATE cp1251_ukrainian_ci AS sort FROM catalogs
    -> ORDER BY sort;
```

В листинге 14.22 для столбца `name` переопределяется кодировка. Чтобы корректно его использовать, в конструкции `ORDER BY` для столбца вводится псевдоним `sort`.

Помимо конструкции `ORDER BY`, ключевое слово `COLLATE` допускается использовать совместно с конструкцией `GROUP BY` (листинг 14.23).

Листинг 14.23. Использование ключевого слова `COLLATE` совместно с `GROUP BY`

```
mysql> SELECT name FROM catalogs
    -> GROUP BY name COLLATE cp1251_general_cs;
```

В листинге 14.23 группировка производится с учетом регистра. Это означает, что записи "процессоры" и "Процессоры" будут восприниматься как две разных записи.

Помимо этого ключевое слово допускается использовать с так называемыми агрегатными функциями (листинг 14.24), которые рассматриваются в *главе 21*.

Листинг 14.24. Использование ключевого слова `COLLATE` в агрегатных функциях

```
mysql> SELECT MAX(name COLLATE cp1251_general_cs) FROM catalogs;
```

С альтернативным `GROUP BY` ключевым словом `DISTINCT` также можно использовать `COLLATE` (листинг 14.25).

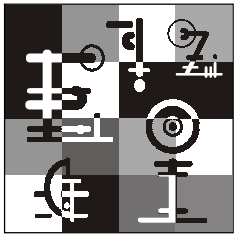
Листинг 14.25. Использование ключевого слова `COLLATE` совместно с `DISTINCT`

```
mysql> SELECT DISTINCT name COLLATE cp1251_ukrainian_ci FROM catalogs;
```

Наконец, ключевое слово `COLLATE` допускается использовать при сравнениях в `WHERE`- и `HAVING`-условиях (листинг 14.26).

Листинг 14.26. Использование ключевого слова `COLLATE` в условиях

```
mysql> SELECT * FROM catalogs
    -> WHERE _cp1251 'Процессоры' COLLATE cp1251_general_cs = name;
mysql> SELECT * FROM catalogs
    -> GROUP BY name
    -> HAVING name = _cp1251 'Процессоры' COLLATE cp1251_general_cs;
```



Глава 15

Операторы и математические функции

В данной главе будут рассмотрены основные функции для работы с числовыми данными. Здесь же подробно описываются операторы, которые можно использовать при составлении запросов в MySQL.

15.1. Операторы

Под *операторами* подразумеваются конструкции языка, которые производят преобразование данных, например, операция сложения +, вычитания – и т. п. Данные, над которыми совершается операция, называются *операндами*.

15.1.1. Арифметические операторы

К арифметическим операциям относят сложение +, вычитание –, умножение *, деление / и целочисленное деление DIV. Если в качестве аргументов используются числа типа INT, то результатом является число типа BIGINT. В то же время если аргументы имеют тип UNSIGNED INT, то результат также будет иметь тип UNSIGNED INT.

> ***Замечание***
>
> Операция целочисленного деления DIV введена в СУБД MySQL, начиная с версии 4.1.0.

Применение операции сложения + демонстрируется в листинге 15.1.

Листинг 15.1. Сложение

```
mysql> SELECT 3 + 5;
+-------+
| 3 + 5 |
+-------+
|     8 |
+-------+
```

Применять операцию сложения можно не только к обычным числам, но и к столбцам. Возвращаясь к учебной базе данных, смоделируем ситуацию привоза, когда на склад поступает продукция от поставщиков. Пусть партия комплектующих включает по 10 штук каждой товарной позиции. Тогда изменить число товара можно при помощи оператора `UPDATE`, представленного в листинге 15.2.

Листинг 15.2. Обновление числа товарных позиций на складе

```
UPDATE products SET count = count + 10;
```

В результате выполнения запроса число товарных позиций для каждой записи будет увеличено на 10. Если необходимо изменить число только для одной или нескольких товарных позиций, необходимо применить конструкцию `WHERE`, которая ограничит число затрагиваемых запросом записей.

Важно отметить, что операция сложения числа с `NULL` дает `NULL` (листинг 15.3).

Листинг 15.3. Сложение с `NULL`

```
mysql> SELECT 3 + NULL;
+----------+
| 3 + NULL |
+----------+
|     NULL |
+----------+
```

Такое поведение вполне оправдано. `NULL` обозначает данные, значение которых не определено, поэтому прибавление к такому значению числа приводит опять к неопределенному значению, т. е. к `NULL`.

MySQL автоматически приводит типы данных. Так, если в качестве слагаемых будут выступать строки, они автоматически окажутся приведенными к числам. Результатом в этом случае также будет число (листинг 15.4).

Листинг 15.4. Приведение строк к числам при сложении

```
mysql> SELECT '3' + '5';
+-----------+
| '3' + '5' |
+-----------+
|         8 |
+-----------+
```

При этом если строка не может быть приведена к числу, она интерпретируется как 0 (листинг 15.5).

Листинг 15.5. Сложение строк

```
mysql> SELECT 'abc' + 'dfe';
+---------------+
| 'abc' + 'dfe' |
+---------------+
|             0 |
+---------------+
```

Замечание

Для объединения строк в MySQL используется встроенная функция `CONCAT()`, ознакомится с синтаксисом которой можно в *главе 17*.

Операция вычитания –, продемонстрированная в листинге 15.6, имеет те же особенности и ограничения, что и операция сложения.

Листинг 15.6. Вычитание

```
mysql> SELECT 8 - 3;
+-------+
| 8 - 3 |
+-------+
|     5 |
+-------+
```

Помимо бинарного оператора –, который производит вычитание, существует унарный оператор, который меняет знак операнда (листинг 15.7).

Листинг 15.7. Смена знака

```
mysql> SELECT -7;
+----+
| -7 |
+----+
| -7 |
+----+
```

Операции умножения * и деления / продемонстрированы в листинге 15.8. Пусть требуется изменить цену комплектующих в учебной базе данных в связи с изменением

курса доллара. Для этого следует разделить цену в рублях на старый курс (`27.5`), чтобы получить цену в долларах, и затем умножить на новый (`27.0`).

Листинг 15.8. Изменение цены в таблице `products`

```
UPDATE products SET price = (price/27.5)*27.0;
```

Как видно из листинга 15.8, точно так же как и в арифметике, порядок вычисления можно задавать при помощи круглых скобок. В приведенном запросе круглые скобки вполне можно опустить — `price/27.5*27.0` — результат будет тот же: каждое значение из столбца `price` будет разделено на `28.5`, а потом умножено на `28.0`.

Если результат умножения выходит за пределы типа `BIGINT`, то возвращается 0 (листинг 15.9).

Листинг 15.9. Выход за границы типа `BIGINT` при умножении

```
mysql> SELECT 18014398509481984*18014398509481984;
+-------------------------------------+
| 18014398509481984*18014398509481984 |
+-------------------------------------+
|                                   0 |
+-------------------------------------+
```

В отличие от других языков программирования, деление на ноль не вызывает ошибки синтаксиса и остановки вычислений. В качестве результата возвращается `NULL` (листинг 15.10).

Листинг 15.10. Деление на ноль

```
mysql> SELECT 8/0;
+------+
| 8/0  |
+------+
| NULL |
+------+
```

Кроме обычного деления, начиная с версии 4.1.0, в MySQL введен оператор целочисленного деления `DIV`, работа которого представлена в листинге 15.11.

Листинг 15.11. Целочисленное деление при помощи DIV

```
mysql> SELECT 5 DIV 2, 5/2;
+---------+------+
| 5 DIV 2 | 5/2  |
+---------+------+
|       2 | 2.50 |
+---------+------+
```

Как видно из листинга 15.11, результат деления при помощи DIV является целочисленным. Дробная часть просто отбрасывается, и округление результата не производится. Для того чтобы получить остаток от деления, необходимо воспользоваться оператором % (листинг 15.12).

Листинг 15.12. Взятие остатка от деления

```
mysql> SELECT 5 % 2;
+-------+
| 5 % 2 |
+-------+
|     1 |
+-------+
```

Результат запроса равен 1, т. к. без остатка на 2 делится только 4 (5 – 4 = 1). Оператор взятия остатка от деления имеет еще две альтернативные формы написания:

- замена % на MOD;
- встроенная функция MOD().

Пример использования встроенной функции MOD() приводится в листинге 15.13.

Листинг 15.13. Альтернативная запись %

```
mysql> SELECT MOD(5,2), 5 MOD 2;
+----------+---------+
| MOD(5,2) | 5 MOD 2 |
+----------+---------+
|        1 |       1 |
+----------+---------+
```

Точно так же, как и в случае обычного деления, при использовании DIV и MOD, если делитель равен нулю, возвращается NULL.

15.1.2. Операторы сравнения

Данная группа операторов интенсивно используется для создания условных запросов с применением инструкций `WHERE` и `HAVING`. Результатом операции сравнения может быть 0 (ложь), 1 (истина) и `NULL`. Сравнивать можно как числа, так и строки.

> ***Замечание***
>
> При сравнении со значением `NULL` результат всегда равен `NULL`. Исключение составляет оператор `<=>`, который специально введен для сравнения `NULL`.

> ***Замечание***
>
> Если один из операндов имеет тип `TIMESTAMP` или `DATETIME`, а другой является константой, операнды сравниваются как `TIMESTAMP`-типы.

Равенство =

Результат равенства равен 1 (истина), если операнды равны, и 0 (ложь) в противном случае. В листинге 15.14 демонстрируется сравнение строк и чисел.

Листинг 15.14. Сравнение строк и чисел

```
mysql> SELECT 1=1, 1=2, '1'=1, 'ab'='ab', 'ab' = 'AB', 'ab' = 0;
+-----+-----+-------+-----------+-------------+----------+
| 1=1 | 1=2 | '1'=1 | 'ab'='ab' | 'ab' = 'AB' | 'ab' = 0 |
+-----+-----+-------+-----------+-------------+----------+
|   1 |   0 |     1 |         1 |           1 |        1 |
+-----+-----+-------+-----------+-------------+----------+
```

Как видно из листинга 15.14, при сравнении строк с числами строки автоматически преобразуются в числа, если же такое преобразование выполнить невозможно, то строка имеет значение 0, поэтому сравнение `'ab' = 0` дает истину — 1. Строка `'14L'` будет преобразована в `'14'`, а `'L14'` в `'0'`.

Сравнение строк по умолчанию происходит без учета регистра. Если необходимо, чтобы регистр учитывался, строки следует сравнивать как бинарные (сравнение производится не посимвольно, а побайтно). Для этого строка предваряется ключевым словом `BINARY` (листинг 15.15) или указывается зависимая от регистра сортировка *(см. разд. 14.6)*.

Листинг 15.15. Бинарное сравнение строк

```
mysql> SELECT BINARY 'abc'='ABC', BINARY 'abc'='abc';
+--------------------+--------------------+
| BINARY 'abc'='ABC' | BINARY 'abc'='abc' |
+--------------------+--------------------+
|                  0 |                  1 |
+--------------------+--------------------+
```

Сравнивать можно не только целочисленные значения и строки, но также числа с плавающей точкой (листинг 15.16).

Листинг 15.16. Сравнение чисел с плавающей точкой

```
mysql> SELECT 0.01 = 0, 0.01 = .01;
+----------+------------+
| 0.01 = 0 | 0.01 = .01 |
+----------+------------+
|        0 |          1 |
+----------+------------+
```

Как видно из листинга 15.16, при написании чисел с плавающей точкой ведущий ноль можно не указывать.

Оператор <=>

Оператор эквивалентности <=> аналогичен обычному равенству, но возвращает он только два значения: 1 (истина) и 0 (ложь). Результата `NULL` он не возвращает. Пример, демонстрирующий работу данного оператора, приведен в листинге 15.17.

Листинг 15.17. Сравнение с использованием оператора эквивалентности <=>

```
mysql> SELECT 1<=>1, 1<=>2, NULL<=>1, NULL<=>NULL;
+-------+-------+----------+-------------+
| 1<=>1 | 1<=>2 | NULL<=>1 | NULL<=>NULL |
+-------+-------+----------+-------------+
|     1 |     0 |        0 |           1 |
+-------+-------+----------+-------------+
```

Как видно из листинга 15.17, сравнение `NULL` с `NULL` дает 1 (истина), в отличие от оператора равенства =, который бы дал в этом случае `NULL`.

Оператор <>

Оператор неравенства <> равен 1 (истина), если операнды не равны, и 0 (ложь) в противном случае. Листинг 15.18 демонстрирует работу данного оператора.

Замечание

Помимо записи <>, данный оператор может принимать форму !=.

Листинг 15.18. Использование оператора неравенства <>

```
mysql> SELECT 0.01<>0.01, 'ab'!='AB', BINARY 'ab'<>'AB', 'ab'!='cd';
+------------+-----------+------------------+------------+
| 0.01<>0.01 | 'ab'!='AB' | BINARY 'ab'<>'AB' | 'ab'!='cd' |
+------------+-----------+------------------+------------+
|          0 |          0 |                1 |          1 |
+------------+-----------+------------------+------------+
```

Данный оператор возвращает значения, противоположные оператору равенства =.

Оператор <

Оператор "меньше" < возвращает 1 (истина), если левый операнд меньше правого (листинг 15.19). В противном случае возвращается 0 (ложь).

Листинг 15.19. Использования оператора <

```
mysql> SELECT 3<10, 5<1, 'big number'<1;
+------+-----+---------------+
| 3<10 | 5<1 | 'big number'<1 |
+------+-----+---------------+
|    1 |   0 |              1 |
+------+-----+---------------+
```

Как видно из листинга 15.19, при сравнении числа со строкой строка автоматически приводится к числу, т. к. в приведенном примере извлечь числовое значение из строки нельзя, вместо нее подставляется значение 0, что приводит к 1 (истина).

Возвращаясь к учебной базе данных, выясним названия товарных позиций, количество которых на складе меньше 5 экземпляров (листинг 15.20).

Листинг 15.20. Выборка из таблицы `products` по цене

```
mysql> SELECT name, count FROM products WHERE count < 5;
+---------------------------+-------+
| name                      | count |
+---------------------------+-------+
| Celeron 2.0GHz            |     2 |
| Celeron 2.4GHz            |     4 |
| Celeron D 320 2.4GHz      |     1 |
```

```
| Intel Pentium 4 3.0GHz       |     1 |
| Gigabyte GA-8I848P-RS        |     4 |
| Gigabyte GA-8IG1000          |     2 |
| Asustek P4C800-E Delux       |     4 |
| ASUSTEK A9600XT/TD           |     2 |
| SAPPHIRE 256MB RADEON 9550   |     3 |
| Maxtor 6B200P0               |     4 |
| Seagate Barracuda ST3160023A |     3 |
+------------------------------+-------+
```

Оператор <=

Оператор "меньше или равно" <= возвращает 1 (истина), если левый операнд меньше правого или оба оператора равны (листинг 15.21), иначе возвращается 0 (ложь).

Листинг 15.21. Использование оператора <=

```
mysql> SELECT 3.4 <= '3.4', 10 <= 1;
+--------------+---------+
| 3.4 <= '3.4' | 10 <= 1 |
+--------------+---------+
|            1 |       0 |
+--------------+---------+
```

Оператор >

Оператор "больше" > возвращает 1 (истина), если левый операнд больше, чем правый (листинг 15.22), иначе возвращается 0 (ложь).

Листинг 15.22. Использование оператора >

```
mysql> SELECT 65 > 34, 65 > 65;
+---------+---------+
| 65 > 34 | 65 > 65 |
+---------+---------+
|       1 |       0 |
+---------+---------+
```

Возвращаясь к учебной базе данных, выясним названия товарных позиций, цена которых превышает 2000 рублей (листинг 15.23).

Листинг 15.23. Выборка из таблицы `products` по цене

```
mysql> SELECT name, price FROM products WHERE price > 2000;
+-----------------------------+---------+
| name                        | price   |
+-----------------------------+---------+
| Celeron 2.4GHz              | 2109.00 |
| Celeron D 325 2.53GHz       | 2747.00 |
| Intel Pentium 4 3.2GHz      | 7259.00 |
| Intel Pentium 4 3.0GHz      | 6147.00 |
| Intel Pentium 4 3.0GHz      | 5673.00 |
| Gigabyte GA-8IG1000         | 2420.00 |
| Gigabyte GA-8IPE1000G       | 2289.00 |
| Asustek P4C800-E Delux      | 5395.00 |
| Asustek P4P800-VM\L i865G   | 2518.00 |
| Epox EP-4PDA3I              | 2289.00 |
| ASUSTEK A9600XT/TD          | 5156.00 |
| SAPPHIRE 256MB RADEON 9550  | 2730.00 |
| GIGABYTE AGP GV-N59X128D    | 5886.00 |
| Maxtor 6Y120P0              | 2456.00 |
| Maxtor 6B200P0              | 3589.00 |
| Samsung SP0812C             | 2093.00 |
| Seagate Barracuda ST3160023A| 3139.00 |
| Seagate ST3120026A          | 2468.00 |
+-----------------------------+---------+
```

Оператор >=

Оператор "больше или равно" >= возвращает 1 (истина), если левый операнд больше правого или они равны, и 0 (ложь) в противном случае (листинг 15.24).

Листинг 15.24. Использование оператора >=

```
mysql> SELECT 65 >= 34, 65 >= 65;
+----------+----------+
| 65 >= 34 | 65 >= 65 |
+----------+----------+
|        1 |        1 |
+----------+----------+
```

Конструкция *IS NULL*

Конструкция `IS NULL` позволяет определить, равно ли проверяемое значение `NULL` или нет (листинг 15.25). Если значение равно `NULL`, оператор возвращает 1 (истина), в противном случае возвращается 0 (ложь).

Листинг 15.25. Использование конструкции `IS NULL`

```
mysql> SELECT NULL IS NULL, 0 IS NULL;
+--------------+-----------+
| NULL IS NULL | 0 IS NULL |
+--------------+-----------+
|            1 |         0 |
+--------------+-----------+
```

Рассмотрим запрос к учебной базе данных. В таблице `users` необязательные поля, такие как телефон, e-mail и URL пользователя могут принимать значение `NULL`, если нет достоверной информации о наличии или отсутствии этих данных о пользователе. Получим имена, фамилии и отчества пользователей, у которых не указан телефон (листинг 15.26).

Листинг 15.26. Запрос к таблице `users` с использованием конструкции `IS NULL`

```
mysql> SELECT surname, name, patronymic FROM users WHERE phone IS NULL;
+-----------+----------+------------+
| surname   | name     | patronymic |
+-----------+----------+------------+
| Кузнецов  | Максим   | Валерьевич |
| Нехорошев | Анатолий | Юрьевич    |
+-----------+----------+------------+
```

> ***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 15.27.

Листинг 15.27. Использование функции `ISNULL()`

```
mysql> SELECT ISNULL(NULL), ISNULL(0);
+--------------+-----------+
| ISNULL(NULL) | ISNULL(0) |
+--------------+-----------+
|            1 |         0 |
+--------------+-----------+
```

Конструкция *IS NOT NULL*

Конструкция `IS NOT NULL` является противопоставлением оператора `IS NULL`. Если значение равно `NULL`, оператор возвращает 0 (ложь), в противном случае возвращается 1 (истина) (листинг 15.28).

Листинг 15.28. Использование конструкции `IS NOT NULL`

```
mysql> SELECT NULL IS NOT NULL, 0 IS NOT NULL;
+------------------+---------------+
| NULL IS NOT NULL | 0 IS NOT NULL |
+------------------+---------------+
|                0 |             1 |
+------------------+---------------+
```

Выберем из таблицы `users` фамилии, имена и отчества всех пользователей, в данных которых имеется URL (листинг 15.29).

Листинг 15.29. Запрос к таблице `users` с использованием конструкции `IS NOT NULL`

```
mysql> SELECT surname, name, patronymic FROM users WHERE url IS NOT NULL;
+----------+--------+--------------+
| surname  | name   | patronymic   |
+----------+--------+--------------+
| Симдянов | Игорь  | Вячеславович |
| Кузнецов | Максим | Валерьевич   |
+----------+--------+--------------+
```

Конструкция *BETWEEN min AND max*

Конструкция `BETWEEN` возвращает 1 (истину), если проверяемое значение расположено между *min* и *max*, в противном случае возвращается 0 (ложь). В листинге 15.30

представлен запрос, извлекающий товарные позиции, цена которых располагается между 1000 и 2000 рублей.

Листинг 15.30. Использование конструкции `BETWEEN`

```
mysql> SELECT name, price FROM products
mysql> WHERE price BETWEEN 1000 AND 2000;
+------------------------------+---------+
| name                         | price   |
+------------------------------+---------+
| Celeron 1.8                  | 1595.00 |
| Celeron 2.0GHz               | 1969.00 |
| Celeron D 320 2.4GHz         | 1962.00 |
| Celeron D 315 2.26GHz        | 1880.00 |
| Gigabyte GA-8I848P-RS        | 1896.00 |
| ASUSTEK V9520X               | 1602.00 |
| DDR-400 256MB Kingston       | 1085.00 |
| DDR-400 256MB Hynix Original | 1179.00 |
| DDR-400 512MB Kingston       | 1932.00 |
| DDR-400 512MB PQI            | 1690.00 |
| DDR-400 512MB Hynix          | 1717.00 |
+------------------------------+---------+
```

Форма записи `price BERWEEN 1000 AND 2000` эквивалентна `1000 <= price AND price <= 2000`. Если типы сравниваемого значения и границ интервала сравнения не совпадают, то происходит приведение типов.

Конструкция *NOT BETWEEN min AND max*

Конструкция `NOT BETWEEN` возвращает 0 (ложь), если проверяемое значение расположено между *min* и *max*, в противном случае возвращается 1 (истина). Выясним, какие товарные позиции не вошли в ценовой интервал между 1000 и 2000 рублей, т. е. решим задачу, противоположную представленной в листинге 15.30 (листинг 15.31).

Листинг 15.31. Использование конструкции `NOT BETWEEN`

```
mysql> SELECT name, price FROM products
mysql> WHERE price NOT BETWEEN 1000 AND 2000;
+------------------------------+---------+
| name                         | price   |
+------------------------------+---------+
| Celeron 2.4GHz               | 2109.00 |
```

```
| Celeron D 325 2.53GHz         | 2747.00 |
| Intel Pentium 4 3.2GHz        | 7259.00 |
| Intel Pentium 4 3.0GHz        | 6147.00 |
| Intel Pentium 4 3.0GHz        | 5673.00 |
| Gigabyte GA-8IG1000           | 2420.00 |
| Gigabyte GA-8IPE1000G         | 2289.00 |
| Asustek P4C800-E Delux        | 5395.00 |
| Asustek P4P800-VM\L i865G     | 2518.00 |
| Epox EP-4PDA3I                | 2289.00 |
| ASUSTEK A9600XT/TD            | 5156.00 |
| SAPPHIRE 256MB RADEON 9550    | 2730.00 |
| GIGABYTE AGP GV-N59X128D      | 5886.00 |
| Maxtor 6Y120P0                | 2456.00 |
| Maxtor 6B200P0                | 3589.00 |
| Samsung SP0812C               | 2093.00 |
| Seagate Barracuda ST3160023A | 3139.00 |
| Seagate ST3120026A            | 2468.00 |
| DDR-400 256MB PQI             |  899.00 |
+-------------------------------+---------+
```

Функция *COALESCE()*

Функция `COALESCE()` возвращает первый элемент из списка, который не равен `NULL`. Применение данной функции демонстрируется в листинге 15.32.

> ***Замечание***
>
> *Списком* называется набор однородных значений, следующих через запятую и помещенных в круглые скобки. Такие списки можно получить в результате выполнения вложенных запросов, которые рассмотрены в *главе 24*.

Листинг 15.32. Использование функции `COALESCE()`

```
mysql> SELECT COALESCE(1 + NULL, 4 - NULL, 10, NULL, 2);
+-----------------------------------------+
| COALESCE(1 + NULL, 4 - NULL, 10, NULL, 2) |
+-----------------------------------------+
|                                      10 |
+-----------------------------------------+
```

Как видно из листинга 15.32, функция возвращает третий элемент списка, равный 10, все предыдущие элементы равны `NULL`, т. к. сложение и вычитание с `NULL` всегда дает `NULL`.

Функция *GREATEST()*

Функция `GREATEST()` возвращает максимальное значение из списка (листинг 15.33).

Листинг 15.33. Использование функции `GREATEST()`

```
mysql> SELECT GREATEST(4,3,-10);
+------------------+
| GREATEST(4,3,-10) |
+------------------+
|                 4 |
+------------------+
mysql> SELECT GREATEST(34.0,3.0,5.0,767.0);
+----------------------------+
| GREATEST(34.0,3.0,5.0,767.0) |
+----------------------------+
|                      767.0 |
+----------------------------+
mysql> SELECT GREATEST('B','A','C');
+----------------------+
| GREATEST('B','A','C') |
+----------------------+
| C                    |
+----------------------+
```

Функция *LEAST()*

Функция `LEAST()` является противоположной `GREATEST()` и возвращает минимальное значение из списка (листинг 15.34).

Листинг 15.34. Использование функции `LEAST()`

```
mysql> SELECT LEAST(-10,4,3);
+---------------+
| LEAST(-10,4,3) |
+---------------+
|            -10 |
+---------------+
```

Конструкция *IN*

Конструкция `IN` возвращает 1 (истина), если проверяемое значение входит в качестве элемента в список, в противном случае возвращается 0 (ложь). Использование конструкции `IN` демонстрируется в листинге 15.35.

Листинг 15.35. Использование конструкции `IN`

```
mysql> SELECT 2 IN (0,3,5,'wefwf');
+---------------------+
| 2 IN (0,3,5,'wefwf') |
+---------------------+
|                   0 |
+---------------------+
mysql> SELECT 'wefwf' IN (0,3,5,'wefwf');
+---------------------------+
| 'wefwf' IN (0,3,5,'wefwf') |
+---------------------------+
|                         1 |
+---------------------------+
```

Если проверяемое значение равно `NULL`, результат всегда равен `NULL` (листинг 15.36).

Листинг 15.36. `NULL` в качестве проверяемого значения

```
mysql> 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 15.37).

Листинг 15.37. `NULL` в качестве элемента списка

```
mysql> SELECT 2 IN (0,NULL,5,'wefwf');
+-------------------------+
| 2 IN (0,NULL,5,'wefwf') |
+-------------------------+
|                    NULL |
+-------------------------+
```

Важно отметить, что в предыдущих версиях MySQL в этом случае возвращался либо 0 (ложь), либо 1 (истина).

Конструкция *NOT IN*

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 15.38. Использование конструкции `NOT IN`

```
mysql> SELECT 2 NOT IN (0,3,5,'wefwf');
+--------------------------+
| 2 NOT IN (0,3,5,'wefwf') |
+--------------------------+
|                        1 |
+--------------------------+
```

Помимо представленного в листинге 15.38 синтаксиса конструкции `NOT IN`, существует альтернативная форма записи этого оператора, которая приведена в листинге 15.39.

Листинг 15.39. Альтернативная форма записи конструкции `NOT IN`

```
mysql> SELECT NOT 2 IN (0,3,5,'wefwf');
+--------------------------+
| NOT 2 IN (0,3,5,'wefwf') |
+--------------------------+
|                        1 |
+--------------------------+
```

Функция *INTERVAL()*

Функция `INTERVAL(N,N1,N2,N3,...)` 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 (листинг 15.40).

Листинг 15.40. Использование функции `INTERVAL()`

```
mysql> SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
+------------------------------------+
| INTERVAL(23, 1, 15, 17, 30, 44, 200) |
+------------------------------------+
|                                  3 |
+------------------------------------+
mysql> SELECT INTERVAL(10, 1, 10, 100, 1000);
```

```
+------------------------------+
| INTERVAL(10, 1, 10, 100, 1000) |
+------------------------------+
|                            2 |
+------------------------------+
mysql> SELECT INTERVAL(22, 23, 30, 44, 200);
+------------------------------+
| INTERVAL(22, 23, 30, 44, 200) |
+------------------------------+
|                            0 |
+------------------------------+
```

15.1.3. Логические операторы

В SQL логические операторы проверяют истинность или ложность выражений. Для обозначения истинного выражения используется любое ненулевое число, чаще 1, а для ложного — 0. Помимо указанных двух значений, логические операторы могут возвращать `NULL`.

> ***Замечание***
>
> Логические операторы называют также *булевыми* в честь математика Джорджа Буля, создателя булевой алгебры.

> ***Замечание***
>
> Логические операторы применяют к числовым данным. Если в качестве операнда выступает строка, то она преобразуется к числу согласно правилам, описанным ранее.

Оператор *NOT*

Это оператор логического отрицания. Если операнд, на который он действует, истинен (например, 1), оператор возвращает 0 (ложь), если операнд ложен (0), то возвращается 1 (истина).

> ***Замечание***
>
> Помимо `NOT` данный оператор может быть записан как `!`.

Листинг 15.41. Использование оператора `NOT`

```
mysql> SELECT NOT 10, NOT 0, NOT NULL, !(1 + 1), ! 1 + 1;
+--------+-------+----------+----------+---------+
| NOT 10 | NOT 0 | NOT NULL | !(1 + 1) | ! 1 + 1 |
+--------+-------+----------+----------+---------+
|      0 |     1 |     NULL |        0 |       1 |
+--------+-------+----------+----------+---------+
```

Как видно из листинга 15.41, применение оператора `NOT` к `NULL` возвращает `NULL`. Последний пример на самом деле сводится к `(!1) + 1 = 0 + 1`, т. е. сначала выполняется оператор отрицания `!`, а лишь затем сложения. В таком случае говорят, что приоритет оператора `!` выше, чем оператора `+`. Уровни приоритетов приводятся в *разд. 15.1.5*.

Оператор *OR*

Это оператор логического ИЛИ, который сравнивает два операнда и возвращает 1 (истина), если хотя бы один из операндов истинен (не 0 и не `NULL`), и 0 (ложь) в противном случае. Пример использования оператора `OR` приведен в листинге 15.42.

> ***Замечание***
>
> Помимо `OR` данный оператор может быть записан как `||`.

Листинг 15.42. Использование оператора `OR`

```
mysql> SELECT 1 OR 1, 1 OR 0, 0 OR 0, 0 || NULL, 1 || NULL;
+--------+--------+--------+-----------+-----------+
| 1 OR 1 | 1 OR 0 | 0 OR 0 | 0 || NULL | 1 || NULL |
+--------+--------+--------+-----------+-----------+
|      1 |      1 |      0 |      NULL |         1 |
+--------+--------+--------+-----------+-----------+
```

Оператор *AND*

Это оператор логического И, который сравнивает два операнда и возвращает 1 (истина), если оба операнда истинны (не 0 и не `NULL`), и 0 (ложь) в противном случае. Пример использования оператора `AND` приведен в листинге 15.43.

> ***Замечание***
>
> Помимо `AND` данный оператор может быть записан как `&&`.

Листинг 15.43. Использование оператора `AND`

```
mysql> SELECT 1 AND 1, 1 AND 0, 1 AND NULL, 0 && NULL;
+---------+---------+------------+-----------+
| 1 AND 1 | 1 AND 0 | 1 AND NULL | 0 && NULL |
+---------+---------+------------+-----------+
|       1 |       0 |       NULL |         0 |
+---------+---------+------------+-----------+
```

Оператор *XOR*

Это оператор исключающего логического ИЛИ. Выражение `a XOR b` эквивалентно `(a AND (NOT b)) OR ((NOT a) and b)`. Пример использования оператора `XOR` приведен в листинге 15.44.

Листинг 15.44. Использование оператора `XOR`

```
mysql> SELECT 1 XOR 1, 0 XOR 0, 1 XOR 0, 0 XOR 1, 1 XOR NULL, 0 XOR NULL;
+---------+---------+---------+---------+------------+------------+
| 1 XOR 1 | 0 XOR 0 | 1 XOR 0 | 0 XOR 1 | 1 XOR NULL | 0 XOR NULL |
+---------+---------+---------+---------+------------+------------+
|       0 |       0 |       1 |       1 |       NULL |       NULL |
+---------+---------+---------+---------+------------+------------+
```

15.1.4. Битовые операторы

Битовые операторы позволяют манипулировать отдельными битами в двоичной последовательности. Операции производятся над значениями типа `BIGINT` (64-разрядные целые). Если в качестве одного из операндов выступает `NULL`, результат также равен `NULL`. Использование битовых последовательностей позволяет значительно уменьшить объем таблиц при хранении наборов логических флагов, состояний и т. п.

Оператор *&*

Оператор `&` соответствует побитовому И (пересечение). При использовании битовых операторов следует помнить, что все операции выполняются над числами в двоичном представлении. Пусть имеются два числа 6 и 10, применение к ним оператора `&` дает 2 (листинг 15.45).

Листинг 15.45. Использование оператора `&`

```
mysql> SELECT 6 & 10;
+--------+
| 6 & 10 |
+--------+
|      2 |
+--------+
```

Этот, казалось бы, парадоксальный результат можно легко понять, если перевести цифры 6 и 10 в двоичное представление, в котором они равны 0110 и 1010 соответственно. Сложение разрядов при использовании оператора `&` происходит согласно пра-

вилам, представленным в табл. 15.1. Учитывая их, можно легко получить результат, который равен 0010 (в десятичной системе — 2):

```
0110 (6)
1010 (10)
0010 (2)
```

Таблица 15.1. *Правила сложения разрядов при использовании оператора* `&`

Операнд/результат	Значения			
Первый операнд	1	1	0	0
Второй операнд	1	0	1	0
Результат	1	0	0	0

В листинге 15.46 приведен более сложный пример: применение оператора `&` для чисел 113 и 45.

Листинг 15.46. Побитовое сложение `&`

```
mysql> SELECT 113 & 45;
+----------+
| 113 & 45 |
+----------+
|       33 |
+----------+
```

Для наглядности представим складываемые числа в двоичном виде:

```
1110001 (113)
0101101 (45)
0100001 (33)
```

Оператор |

Оператор `|` соответствует побитовому ИЛИ (объединение). В табл. 15.2 приведены правила сложения разрядов при использовании оператора `|`.

Таблица 15.2. *Правила сложения разрядов при использовании оператора* `|`

Операнд/результат	Значения			
Первый операнд	1	1	0	0
Второй операнд	1	0	1	0
Результат	1	1	1	0

Применяя операцию | к числам, рассмотренным в предыдущем разделе, получим следующий результат, приведенный в листинге 15.47.

Листинг 15.47. Использование оператора |

```
mysql> SELECT 6 | 10, 113 | 45;
+--------+----------+
| 6 | 10 | 113 | 45 |
+--------+----------+
|     14 |      125 |
+--------+----------+
```

Дадим двоичное представление чисел, чтобы действие оператора было более понятным:

```
0110 (6)
1010 (10)
1110 (14)
```

Для второй пары чисел:

```
1110001 (113)
0101101 (45)
1111101 (125)
```

Оператор ^

Оператор ^ соответствует побитовому исключающему ИЛИ (XOR). В табл. 15.3 приведены правила сложения разрядов при использовании оператора ^.

Таблица 15.3. *Правила сложения разрядов при использовании оператора ^*

Операнд/результат	Значения			
Первый операнд	1	1	0	0
Второй операнд	1	0	1	0
Результат	0	1	1	0

В листинге 15.48 приведен пример использования данного оператора применительно к числам 6, 10 и 113, 45.

Листинг 15.48. Использование оператора ^

```
mysql> SELECT 6 ^ 10, 113 ^ 45;
+--------+----------+
| 6 ^ 10 | 113 ^ 45 |
+--------+----------+
|     12 |       92 |
+--------+----------+
```

Двоичное представление первой пары чисел:

```
0110 (6)
1010 (10)
1100 (12)
```

Для второй пары чисел:

```
1110001 (113)
0101101 (45)
1011100 (92)
```

Оператор ~

Оператор ~ является побитовой инверсией и заменяет все 0 на 1 и наоборот. В листинге 15.49 демонстрируется использование данного оператора.

Листинг 15.49. Использование оператора ~

```
mysql> SELECT ~45, ~92;
+---------------------+---------------------+
| ~45                 | ~92                 |
+---------------------+---------------------+
| 18446744073709551570 | 18446744073709551523 |
+---------------------+---------------------+
```

Такие огромные числа, получившиеся в результате, являются следствием того, что все побитовые операции производятся над значениями типа `BIGINT` (64-разрядные целые). В двоичных числах ведущие нули обычно не указывают, но т. к. операция ~ производит их инвертирование — они заменяются единицами, приводя к очень большим величинам. Так, операция `~45` в двоичном представлении выглядит следующим образом:

```
0000000000000000000000000000000000000000000000000000000000101101 (45)
1111111111111111111111111111111111111111111111111111111111010010
```

Для `~92`:

```
00000000000000000000000000000000000000000000000000000001011100 (92)
11111111111111111111111111111111111111111111111111111110100011
```

Оператор <<

Оператор `<<` сдвигает влево все биты первого операнда на число позиций, указанных во втором операнде. Сдвиг на отрицательное значение приводит к нулевому значению.

Листинг 15.50. Использование оператора <<

```
mysql> SELECT 6 << 1, 6 << 2, 6 << 10, 6 << -1, 6 << 64;
+--------+--------+---------+---------+---------+
| 6 << 1 | 6 << 2 | 6 << 10 | 6 << -1 | 6 << 64 |
+--------+--------+---------+---------+---------+
|     12 |     24 |    6144 |       0 |       0 |
+--------+--------+---------+---------+---------+
```

В двоичном представлении эти операции выглядят следующим образом

- для `6 << 1`

  `0110 (6)`

  `1100 (12)`

- для `6 << 2`

  `00110 (6)`

  `11000 (24)`

- для `6 << 10`

  `0000000000110 (6)`

  `1100000000000 (6144)`

Как видно из листинга 15.50, нулевой результат является следствием не только сдвига на отрицательное число позиций, но и сдвига на 64 позиции, т. к. это приводит к тому, что все 1 уходят за границу 64-битного числа и все разряды принимают значение 0.

```
0000000000000000000000000000000000000000000000000000000000000000 (0)
```

Оператор >>

Данный оператор сдвигает вправо все биты первого операнда на число позиций, указанных во втором операнде. Сдвиг на отрицательное значение приводит к нулевому значению.

Листинг 15.51. Использование оператора >>

```
mysql> SELECT 6144 >> 10, 45 >> 2, 45 > 2;
+------------+---------+--------+
| 6144 >> 10 | 45 >> 2 | 45 > 2 |
+------------+---------+--------+
|          6 |      11 |      1 |
+------------+---------+--------+
```

В двоичном представлении эти операции выглядят следующим образом:

- для `6144 >> 10`

  `1100000000000 (6144)`
  `0000000000110 (6)`

- для `45 >> 2`

  `101101 (45)`
  `001011 (11)`

Как видно из листинга 15.51, при использовании операторов сдвига очень легко ошибиться, указав только один знак >. В этом случае, как было описано ранее, MySQL сравнит два числа и вернет либо 1 (истина), либо 0 (ложь). За такими опечатками следует следить очень внимательно, т. к. MySQL не генерирует предупреждений, в то же время результат будет отличаться от ожидаемого.

Функция *BIT_COUNT()*

Функция `BIT_COUNT(N)` возвращает количество единиц в числе `N`.

Пример использования функции `BIT_COUNT` представлен в листинге 15.52.

Листинг 15.52. Использование функции `BIT_COUNT()`

```
mysql> SELECT BIT_COUNT(4), BIT_COUNT(12), BIT_COUNT(478);
+--------------+---------------+----------------+
| BIT_COUNT(4) | BIT_COUNT(12) | BIT_COUNT(478) |
+--------------+---------------+----------------+
|            1 |             2 |              7 |
+--------------+---------------+----------------+
```

Для того чтобы лучше понять логику данного оператора, представим числа 4, 12 и 478 в двоичной форме:

```
000000100 (4)
000001100 (12)
111011110 (478)
```

15.1.5. Приоритет операторов

Рассмотренные операторы обладают различными приоритетами, что демонстрируется в листинге 15.53. Оператор отрицания ! обладает бóльшим приоритетом, чем оператор сложения +, поэтому если необходимо сначала сложить числа, а лишь затем применить к результату оператор !, следует добавлять круглые скобки.

Листинг 15.53. Приоритет операторов

```
mysql> SELECT ! (1 + 1), ! 1 + 1;
+-----------+---------+
| ! (1 + 1) | ! 1 + 1 |
+-----------+---------+
|         0 |       1 |
+-----------+---------+
```

Уровни приоритетов отображает список, представленный далее — сверху вниз уровень приоритета уменьшается. Записанные на одной строке операторы обладают одним уровнем приоритета и выполняются в выражении слева направо:

- BINARY, COLLATE;
- !, NOT;
- – (унарный минус), ~;
- ^;
- *, /, DIV, %, MOD;
- -, +;
- <<, >>;
- &;
- |;
- =, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN;
- BETWEEN, CASE, WHEN, THEN, ELSE;
- NOT;
- &&, AND;
- ||, OR, XOR;
- :=.

Часть операторов из списка не была рассмотрена в данной главе. Операторы LIKE, REGEXP рассматриваются в *главе 19*. Операторы CASE, WHEN, IF, THEN, ELSE обсуждаются в *главе 22*. Оператор := для работы с пользовательскими переменными описывается в *главе 23*.

Замечание

Уровень приоритетов отдельных операторов можно не запоминать, т. к. в спорных ситуациях всегда можно указать порядок вычислений при помощи круглых скобок.

В списке приоритетов операторов, представленном ранее, есть несколько особенностей. Унарный минус (применяемый для смены знака числа `-7`, `-(1+5)`) имеет приоритет на три порядка выше, чем бинарный (применяемый для вычитания `5-8`).

15.2. Математические функции

Все числовые функции возвращают `NULL`, если при их работе возникает ошибка, например, если аргументы выходят за границы допустимых значений или в качестве аргументов передается `NULL`.

MySQL, как и любая другая база данных, обладает большим числом встроенных функций, которые будут рассматриваться на протяжении следующих нескольких глав. Так же как и в любом другом языке программирования, функции характеризуются именем и аргументами, которые перечисляются через запятую за именем в круглых скобках. Если аргументы у функции отсутствуют, круглые скобки все равно следует указывать, например, `NOW()`. Результат функции подставляется в место вызова функции.

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]` конфигурационного файла `my.ini` директиву `sql-mode=IGNORE-SPACE`.

15.2.1. Функция *ABS()*

Функция `ABS(X)` возвращает абсолютное значение аргумента `X` (листинг 15.54).

Листинг 15.54. Использование функции `ABS()`

```
mysql> SELECT ABS(-4.3), ABS(4.3), ABS(NULL);
+-----------+----------+-----------+
| ABS(-4.3) | ABS(4.3) | ABS(NULL) |
+-----------+----------+-----------+
|       4.3 |      4.3 |      NULL |
+-----------+----------+-----------+
```

15.2.2. Функция *ACOS()*

Функция `ACOS(X)` возвращает арккосинус числа `X` или `NULL`, если значение `X` выходит из диапазона от –1 до 1 (листинг 15.55).

Листинг 15.55. Использование функции `ACOS()`

```
mysql> SELECT ACOS(1), ACOS(0), ACOS(1.001), ACOS(NULL);
+---------+-----------------+-------------+------------+
| ACOS(1) | ACOS(0)         | ACOS(1.001) | ACOS(NULL) |
+---------+-----------------+-------------+------------+
|       0 | 1.5707963267949 |        NULL |       NULL |
+---------+-----------------+-------------+------------+
```

15.2.3. Функция *ASIN()*

Функция `ASIN(X)` возвращает арксинус числа `X` или `NULL`, если значение `X` выходит из диапазона от –1 до 1 (листинг 15.56).

Листинг 15.56. Использование функции `ASIN()`

```
mysql> SELECT ASIN(1), ASIN(0), ASIN(1.001), ASIN(NULL);
+-----------------+---------+-------------+------------+
| ASIN(1)         | ASIN(0) | ASIN(1.001) | ASIN(NULL) |
+-----------------+---------+-------------+------------+
| 1.5707963267949 |       0 |        NULL |       NULL |
+-----------------+---------+-------------+------------+
```

15.2.4. Функция *ATAN()*

Функция `ATAN(X)` возвращает арктангенс числа `X`.

Листинг 15.57. Использование функции `ATAN(X)`

```
mysql> SELECT ATAN(0), ATAN(1), ATAN('единица'), ATAN(NULL);
+---------+-----------------+----------------+-------------+
| ATAN(0) | ATAN(1)         | ATAN('единица') | ATAN(NULL) |
+---------+-----------------+----------------+-------------+
|       0 | 0.78539816339745 |              0 |        NULL |
+---------+-----------------+----------------+-------------+
```

Как видно из листинга 15.57, передача в качестве аргумента строки приводит к ее автоматическому приведению к числу (в данном случае к 0).

Функция `ATAN()` может принимать также два аргумента `ATAN(X,Y)`, как это показано в листинге 15.58. В таком формате функция аналогична взятию арктангенса от частного `X` и `Y`, т. е. `ATAN(X/Y)`.

Листинг 15.58. Использование функции `ATAN(X,Y)`

```
mysql> SELECT ATAN(3,2), ATAN(3/2);
+------------------+------------------+
| ATAN(3,2)        | ATAN(3/2)        |
+------------------+------------------+
| 0.98279372324733 | 0.98279372324733 |
+------------------+------------------+
```

15.2.5. Функция *ATAN2()*

Функция `ATAN2(X,Y)` аналогична `ATAN(X,Y)`, однако для вычисления квадранта возвращаемого значения в ней учитываются знаки аргументов (листинг 15.59).

Листинг 15.59. Использование функции `ATAN2()`

```
mysql> SELECT ATAN2(1,1), ATAN2(1/-1), ATAN2(-1,-1);
+------------------+-------------------+------------------+
| ATAN2(1,1)       | ATAN2(1/-1)       | ATAN2(-1,-1)     |
+------------------+-------------------+------------------+
| 0.78539816339745 | -0.78539816339745 | -2.3561944901923 |
+------------------+-------------------+------------------+
```

15.2.6. Функция *CEILING()*

Функция `CEILING(X)` принимает дробное число `X` и возвращает наименьшее целое, не меньше, чем `X`.

Замечание

Для функции `CEILING()` существует более короткий синоним `CEIL()`.

Листинг 15.60. Использование функции CEILING()

```
mysql> 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 `CEILING()` не производит округления, как может показаться на первый взгляд. Она возвращает первое целое число, которое встречает справа от значения аргумента (рис. 15.1). Пример использования функции `CEILING()` приведен в листинге 15.60.

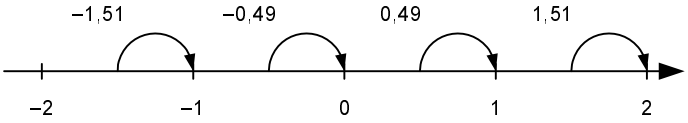


Рис. 15.1. Принцип действия функции `CEILING()`

Замечание

Результат, возвращаемый функцией `CEILING()`, имеет тип `BIGINT`.

15.2.7. Функция *COS()*

Функция `COS(X)` вычисляет косинус угла `X`, заданного в радианах (листинг 15.61).

Листинг 15.61. Использование функции COS()

```
mysql> SELECT COS(0), COS(1), COS(PI());
+--------+-----------------+-----------+
| COS(0) | COS(1)          | COS(PI()) |
+--------+-----------------+-----------+
|      1 | 0.54030230586814 |        -1 |
+--------+-----------------+-----------+
```

15.2.8. Функция *COT()*

Функция `COT(X)` вычисляет котангенс угла `X`, заданного в радианах (листинг 15.62).

Листинг 15.62. Использование функции `COT()`

```
mysql> SELECT COT(0), COT(1), COT(10);
+--------+-----------------+-----------------+
| COT(0) | COT(1)          | COT(10)         |
+--------+-----------------+-----------------+
|   NULL | 0.64209261593433 | 1.5423510453569 |
+--------+-----------------+-----------------+
```

15.2.9. Функция *CRC32()*

Функция `CRC32(str)` вычисляет значение кода циклической проверки избыточности строки *str* и возвращает 32-битное значение в диапазоне от 0 до $2^{32} - 1$. Если в качестве аргумента функции передано значение `NULL`, функция возвращает `NULL` (листинг 15.63).

> ***Замечание***
>
> Функция `CRC32()` введена в MySQL, начиная с версии 4.1.0.

Листинг 15.63. Использование функции `CRC32()`

```
mysql> SELECT CRC32('MySQL'), CRC32('0'), CRC32(0), CRC32(NULL);
+---------------+------------+------------+-------------+
| CRC32('MySQL') | CRC32('0') | CRC32(0)   | CRC32(NULL) |
+---------------+------------+------------+-------------+
|     3259397556 | 4108050209 | 4108050209 |        NULL |
+---------------+------------+------------+-------------+
```

15.2.10. Функция *DEGREES()*

Функция `DEGREES(X)` возвращает значение угла `X`, преобразованное из радиан в градусы (листинг 15.64).

Листинг 15.64. Использование функции DEGREES()

```
mysql> SELECT DEGREES(PI()), DEGREES(PI()/4), DEGREES(NULL);
+--------------+----------------+---------------+
| DEGREES(PI()) | DEGREES(PI()/4) | DEGREES(NULL) |
+---------------+-----------------+---------------+
|           180 |              45 |          NULL |
+---------------+-----------------+---------------+
```

15.2.11. Функция *EXP()*

Функция `EXP(X)` возвращает значение степени числа `X`: e^x, где e — основание натурального логарифма (листинг 15.65).

Листинг 15.65. Использование функции EXP()

```
mysql> SELECT EXP(1), EXP(2), EXP('10');
+---------------+----------------+----------------+
| EXP(1)        | EXP(2)         | EXP('10')      |
+---------------+----------------+----------------+
| 2.718281828459 | 7.3890560989307 | 22026.465794807 |
+---------------+----------------+----------------+
```

Как видно из листинга 15.65, строки автоматически преобразуются в числа (в данном случае 10).

15.2.12. Функция *FLOOR()*

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 15.66.

Листинг 15.66. Использование функции FLOOR()

```
mysql> 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 |
+-------------+-------------+--------------+--------------+
```

15.2.13. Функция *LOG()*

Функция `LOG(X)` возвращает натуральный логарифм (с основанием *e*) числа `X`.

> ***Замечание***
>
> Кроме `LOG(X)` допускается запись `LN(X)`.

Листинг 15.67. Использование функции `LOG(X)`

```
mysql> SELECT LOG(0), LOG(1), LN(2), LN(EXP(1));
+--------+--------+------------------+------------+
| LOG(0) | LOG(1) | LN(2)            | LN(EXP(1)) |
+--------+--------+------------------+------------+
|   NULL |      0 | 0.69314718055995 |          1 |
+--------+--------+------------------+------------+
```

Кроме формата с одним аргументом, показанного в листинге 15.67, функция допускает передачу ей двух аргументов — `LOG(B,X)`, в этом случае функция возвращает логарифм числа `X` по основанию `B` (листинг 15.68).

Листинг 15.68. Использование функции `LOG(B,X)`

```
mysql> SELECT LOG(10,100), LOG(2,256);
+-------------+------------+
| LOG(10,100) | LOG(2,256) |
+-------------+------------+
|           2 |          8 |
+-------------+------------+
```

15.2.14. Функция *LOG2()*

Функция `LOG2(X)`, пример которой показан в листинге 15.69, возвращает логарифм числа `X` по основанию 2 .

Листинг 15.69. Использование функции `LOG2()`

```
mysql> SELECT LOG2(255), LOG2(32768);
+-----------------+-------------+
| LOG2(255)       | LOG2(32768) |
+-----------------+-------------+
| 7.9943534368589 |          15 |
+-----------------+-------------+
```

15.2.15. Функция *LOG10()*

Функция `LOG10(X)` возвращает логарифм числа `X` по основанию 10 (листинг 15.70).

Листинг 15.70. Использование функции `LOG10()`

```
mysql> SELECT LOG10(0), LOG10(10), LOG10(100);
+----------+-----------+------------+
| LOG10(0) | LOG10(10) | LOG10(100) |
+----------+-----------+------------+
|     NULL |         1 |          2 |
+----------+-----------+------------+
```

15.2.16. Функция *MOD()*

Функция `MOD(M, N)` аналогична оператору `M % N` и возвращает остаток от деления `M` на `N`. Более подробно применение данной функции описано в *разд. 15.1.1*.

15.2.17. Функция *PI()*

Функция `PI()` не принимает аргументов и возвращает значение числа π (листинг 15.71).

Листинг 15.71. Использование функции `PI()`

```
mysql> SELECT PI()*2, PI(), PI()/2, PI()/4;
+----------+----------+------------+------------+
| PI()*2   | PI()     | PI()/2     | PI()/4     |
+----------+----------+------------+------------+
| 6.283185 | 3.141593 | 1.57079633 | 0.78539816 |
+----------+----------+------------+------------+
```

15.2.18. Функция *POW()*

Функция `POW(X,Y)` возвращает значение числа `X`, возведенного в степень `Y` — X^Y.

Замечание

Для функции `POW()` существует синоним `POWER()`.

Листинг 15.72. Использование функции POW()

```
mysql> SELECT POW(2,3), POW(2,-3), POWER(4,.5), POWER(16,.25);
+----------+-----------+-------------+---------------+
| POW(2,3) | POW(2,-3) | POWER(4,.5) | POWER(16,.25) |
+----------+-----------+-------------+---------------+
|        8 |     0.125 |           2 |             2 |
+----------+-----------+-------------+---------------+
```

Как видно из листинга 15.72, функция `POW()` допускает и дробные аргументы, что позволяет вычислять корень числа. Так в третьем столбце вычисляется квадратный корень, а в четвертом — корень четвертой степени.

15.2.19. Функция *RADIANS()*

Функция `RADIANS(X)` возвращает значение угла `X`, преобразованное из градусов в радианы (листинг 15.73).

Листинг 15.73. Использование функции RADIANS()

```
mysql> SELECT RADIANS(0), RADIANS(180), RADIANS(360);
+------------+-----------------+-----------------+
| RADIANS(0) | RADIANS(180)    | RADIANS(360)    |
+------------+-----------------+-----------------+
|          0 | 3.1415926535898 | 6.2831853071796 |
+------------+-----------------+-----------------+
```

15.2.20. Функция *RAND()*

Функция `RAND()` возвращает случайное значение 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 во все последующие вызовы — без параметров. Пример использования функции `RAND()` показан в листинге 15.74.

Листинг 15.74. Использование функции RAND()

```
mysql> 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 качестве примера выберем из таблицы `products` 10 случайных товарных позиций (листинг 15.75).

Листинг 15.75. Случайная выборка из таблицы с использованием функции `RAND()`

```
mysql> SELECT name FROM products ORDER BY RAND() LIMIT 10;
+-----------------------------+
| name                        |
+-----------------------------+
| Asustek P4P800-VM\L i865G   |
| ASUSTEK A9600XT/TD          |
| Gigabyte GA-8IG1000         |
| Celeron D 325 2.53GHz       |
| Asustek P4C800-E Delux      |
| DDR-400 512MB Kingston      |
| Celeron D 315 2.26GHz       |
| Gigabyte GA-8IPE1000G       |
| Seagate Barracuda ST3160023A |
| Intel Pentium 4 3.0GHz      |
+-----------------------------+
```

Часто требуется осуществить выборку одной случайной записи. Для этого достаточно указать после ключевого слова `LIMIT` цифру 1 (листинг 15.76).

Листинг 15.76. Ограничение выборки при помощи конструкции `LIMIT`

```
mysql> SELECT name FROM products ORDER BY RAND() LIMIT 1;
+-------------+
| name        |
+-------------+
| Celeron 1.8 |
+-------------+
```

15.2.21. Функция *ROUND()*

Функция `ROUND(X)` возвращает округленное до ближайшего целого значение числа `X`. Данная функция может принимать два аргумента — `ROUND(X,D)`, в этом случае округление производится до числа, имеющего `D` знаков после запятой (листинг 15.77).

Листинг 15.77. Использование функции `ROUND()`

```
mysql> SELECT ROUND(4.3), ROUND(4.7), ROUND(-3.56847,2), ROUND(1,3);
+------------+------------+-------------------+------------+
| ROUND(4.3) | ROUND(4.7) | ROUND(-3.56847,2) | ROUND(1,3) |
+------------+------------+-------------------+------------+
|          4 |          5 |             -3.57 |      1.000 |
+------------+------------+-------------------+------------+
```

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 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 15.78 показан пример применения функции `ROUND()`.

Листинг 15.78. Округление с дробной частью `.5`

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+------------+-------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+-------------+
| 3          |            2 |
+------------+-------------+
```

15.2.22. Функция *SIGN()*

Функция `SIGN(X)` позволяет определить знак числа `X` и возвращает –1, 0 или 1, если `X` отрицательно, равно 0 или положительно, соответственно (листинг 15.79).

Листинг 15.79. Использование функции `SIGN()`

```
mysql> SELECT SIGN(4.3), SIGN(0), SIGN(-4.3);
+-----------+---------+------------+
| SIGN(4.3) | SIGN(0) | SIGN(-4.3) |
+-----------+---------+------------+
|         1 |       0 |         -1 |
+-----------+---------+------------+
```

15.2.23. Функция *SIN()*

Функция `SIN(X)` вычисляет синус угла `X`, заданного в радианах (листинг 15.80).

Листинг 15.80. Использование функции `SIN()`

```
mysql> SELECT SIN(0), SIN(1), SIN(PI()/2);
+--------+----------------+------------+
| SIN(0) | SIN(1)         | SIN(PI()/2) |
+--------+----------------+------------+
|      0 | 0.8414709848079 |           1 |
+--------+----------------+------------+
```

15.2.24. Функция *SQPT()*

Функция `SQRT(X)` вычисляет квадратный корень числа `X` (листинг 15.81).

Листинг 15.81. Использование функции `SQRT()`

```
mysql> SELECT SQRT(4), SQRT(20), SQRT(-5), SQRT(NULL);
+---------+----------------+----------+------------+
| SQRT(4) | SQRT(20)       | SQRT(-5) | SQRT(NULL) |
+---------+----------------+----------+------------+
|       2 | 4.4721359549996 |     NULL |       NULL |
+---------+----------------+----------+------------+
```

Как видно из листинга 15.81, передача в качестве аргумента отрицательного значения приводит к значению `NULL`.

15.2.25. Функция *TAN()*

Функция `TAN(X)` вычисляет тангенс угла `X`, заданного в радианах (листинг 15.82).

Листинг 15.82. Использование функции `TAN()`

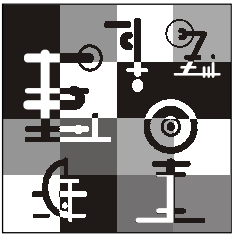
```
mysql> SELECT TAN(0), TAN(PI()/4), TAN(NULL);
+--------+-------------+-----------+
| TAN(0) | TAN(PI()/4) | TAN(NULL) |
+--------+-------------+-----------+
|      0 |           1 |      NULL |
+--------+-------------+-----------+
```

15.2.26. Функция *TRUNCATE()*

Функция `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 (листинг 15.83).

Листинг 15.83. Использование функции `TRUNCATE()`

```
mysql> 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 |
+------------------+------------------+------------------+
```



Глава 16

Функции даты и времени

Группа функций даты и времени предназначена для работы с данными временно́го типа: `DATETIME`, `DATE`, `TIMESTAMP`, `TIME` и `YEAR`, которые подробно рассматриваются в *главе 4*.

> ***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`.

Функции даты и времени могут принимать данные в различных форматах:

- в виде строки, содержащей полную дату и время суток '2005-12-27 08:30:00';
- строки, содержащей только дату '2005-12-27';
- в виде числа 20051227083000 или 20051227.

MySQL корректно распознает время во всех форматах и переводит его во внутреннее представление.

Ряд функций позволяет вернуть текущую дату или время суток. К ним относятся `NOW()`, `CURDATE()`, `CURTIME()`, `UTC_DATE()`, `UTC_TIME()`, `UTC_TIMESTAMP()`. 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.

16.1. Функция *ADDDATE()*

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. 16.1.

Рассмотрим запрос из листинга 16.1.

Листинг 16.1. Использование функции ADDDATE()

```
mysql> SELECT ADDDATE('2006-07-20',INTERVAL 10 DAY);
+--------------------------------------+
| ADDDATE('2006-07-20',INTERVAL 10 DAY) |
+--------------------------------------+
| 2006-07-30                           |
+--------------------------------------+
```

В листинге 16.1 приводится пример использования функции ADDDATE(). К дате 20 июля 2006 года добавляется 10 дней, в результате чего возвращается дата 30 июля 2006 года.

Таблица 16.1. *Типы временных интервалов функции ADDDATE()*

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

Таблица 16.1 *(окончание)*

Тип	Описание	Формат назначения
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

Функция ADDDATE() допускает и отрицательные значения (листинг 16.2). В этом случае происходит вычитание временного интервала.

Листинг 16.2. Работа с отрицательными значениями

```
mysql> SELECT ADDDATE('2006-07-20',INTERVAL -10 DAY);
+--------------------------------------+
| ADDDATE('2006-07-20',INTERVAL -10 DAY) |
+--------------------------------------+
| 2006-07-10                           |
+--------------------------------------+
```

Остановимся подробнее на табл. 16.1. Вместо единиц измерения DAY, которые использовались в листингах 16.1 и 16.2, можно указать любой параметр из первого столбца таблицы. 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 версии 5.0.0.

Работа с типом временного интервала MICROSECOND продемонстрирована в листинге 16.3.

Листинг 16.3. Использование типа MICROSECOND

```
mysql> SELECT ADDDATE('2006-07-20 00:00:00',INTERVAL 1 MICROSECOND);
+-----------------------------------------------------+
| ADDDATE('2006-07-20 00:00:00',INTERVAL 1 MICROSECOND) |
+-----------------------------------------------------+
| 2006-07-20 00:00:00.000001                          |
+-----------------------------------------------------+
```

```
mysql> SELECT ADDDATE('2006-07-20 00:00:00',
    -> INTERVAL 1000000 MICROSECOND);
+-----------------------------------------------------------+
| ADDDATE('2006-07-20 00:00:00',INTERVAL 1000000 MICROSECOND) |
+-----------------------------------------------------------+
| 2006-07-20 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 16.4 демонстрируется использование интервала `MINUTE_MICROSECOND`.

Листинг 16.4. Использование типа `MINUTE_MICROSECOND`

```
mysql> SELECT ADDDATE('2006-07-20',
    -> INTERVAL '59:10.20' MINUTE_MICROSECOND);
+-------------------------------------------------------------+
| ADDDATE('2006-07-20',INTERVAL '59:10.20' MINUTE_MICROSECOND) |
+-------------------------------------------------------------+
| 2006-07-20 00:59:10.200000                                  |
+-------------------------------------------------------------+
```

При использовании типа `WEEK` (неделя), показанного в листинге 16.5, и `QUARTER` (квартал), пример которого приведен в листинге 16.6, происходит увеличение даты на интервалы, кратные семи дням и трем месяцам, соответственно.

Листинг 16.5. Использование типа `WEEK`

```
mysql> SELECT ADDDATE('2006-07-20',INTERVAL 52 WEEK);
+---------------------------------------+
| ADDDATE('2006-07-20',INTERVAL 52 WEEK) |
+---------------------------------------+
| 2007-07-19                            |
+---------------------------------------+
mysql> SELECT ADDDATE('2006-07-20',INTERVAL 1 WEEK);
+--------------------------------------+
| ADDDATE('2006-07-20',INTERVAL 1 WEEK) |
+--------------------------------------+
| 2006-07-27                           |
+--------------------------------------+
```

Листинг 16.6. Использование типа `QUARTER`

```
mysql> SELECT ADDDATE('2006-07-20',INTERVAL 1 QUARTER);
+-----------------------------------------+
| ADDDATE('2006-07-20',INTERVAL 1 QUARTER) |
+-----------------------------------------+
| 2006-10-20                              |
+-----------------------------------------+
mysql> SELECT ADDDATE('2006-07-20',INTERVAL 3 QUARTER);
+-----------------------------------------+
| ADDDATE('2006-07-20',INTERVAL 3 QUARTER) |
+-----------------------------------------+
| 2007-04-20                              |
+-----------------------------------------+
```

Помимо функции `ADDDATE()`, для сложения даты с временным интервалом допустимо использование оператора `+` (листинг 16.7), который действует аналогично функции `ADDDATE()`.

Листинг 16.7. Использование оператора + с временными типами данных

```
mysql> SELECT '2006-07-20' + INTERVAL '10 12:59:59.14' DAY_MICROSECOND;
+-------------------------------------------------------+
| '2006-07-20' + INTERVAL '10 12:59:59.14' DAY_MICROSECOND |
+-------------------------------------------------------+
| 2006-07-30 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 16.8).

> ***Замечание***
>
> Наряду с оператором `+` можно использовать оператор `-` для вычитания временного интервала из даты.

Листинг 16.8. Выход за границы часа

```
mysql> SELECT '2006-07-20' + INTERVAL '10 12:100:59.14' DAY_MICROSECOND;
+-------------------------------------------------------+
| '2006-07-20' + INTERVAL '10 12:100:59.14' DAY_MICROSECOND |
+-------------------------------------------------------+
```

```
| 2006-07-30 13:40:59.140000                                |
+-----------------------------------------------------------+
mysql> SELECT '2006-07-20' + INTERVAL 30 MONTH;
+---------------------------------+
| '2006-07-20' + INTERVAL 30 MONTH |
+---------------------------------+
| 2009-01-20                      |
+---------------------------------+
```

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 16.9).

Замечание

Краткая форма функции `ADDDATE()` появилась в MySQL, начиная с версии 4.1.1.

Листинг 16.9. Использование краткой формы функции `ADDDATE()`

```
mysql> SELECT ADDDATE('2006-07-20',6), ADDDATE('2006-07-20',-6);
+-------------------------+-------------------------+
| ADDDATE('2006-07-20',6) | ADDDATE('2006-07-20',-6) |
+-------------------------+-------------------------+
| 2006-07-26              | 2006-07-14              |
+-------------------------+-------------------------+
```

Возвращаясь к учебной базе данных, смоделируем ситуацию перехода на летнее время и вычтем из времени оформления заказа в таблице `orders` один час (листинг 16.10).

Листинг 16.10. Перевод времени на один час

```
mysql> SELECT * FROM orders;
+----------+---------+---------------------+--------+------------+
| id_order | id_user | ordertime           | number | id_product |
+----------+---------+---------------------+--------+------------+
|        1 |       3 | 2005-01-04 10:39:38 |      1 |          8 |
|        2 |       6 | 2005-02-10 09:40:29 |      2 |         10 |
|        3 |       1 | 2005-02-18 13:41:05 |      4 |         20 |
|        4 |       3 | 2005-03-10 18:20:00 |      1 |         20 |
|        5 |       3 | 2005-03-17 19:15:36 |      1 |         20 |
+----------+---------+---------------------+--------+------------+
```

```
mysql> UPDATE orders SET ordertime = ordertime - INTERVAL 1 HOUR;
mysql> SELECT * FROM orders;
+----------+---------+---------------------+--------+------------+
| id_order | id_user | ordertime           | number | id_product |
+----------+---------+---------------------+--------+------------+
|        1 |       3 | 2005-01-04 09:39:38 |      1 |          8 |
|        2 |       6 | 2005-02-10 08:40:29 |      2 |         10 |
|        3 |       1 | 2005-02-18 12:41:05 |      4 |         20 |
|        4 |       3 | 2005-03-10 17:20:00 |      1 |         20 |
|        5 |       3 | 2005-03-17 18:15:36 |      1 |         20 |
+----------+---------+---------------------+--------+------------+
```

Замечание

На нулевых значениях временного типа '0000-00-00 00:00:00' вычитание не отражается.

16.2. Функция *ADDTIME()*

Функция `ADDTIME(`*expr1*`, `*expr2*`)` возвращает результат сложения двух временных значений *expr1* и *expr2*.

Замечание

Функция `ADDTIME()` появилась в MySQL в версии 4.1.1.

Листинг 16.11. Использование функции `ADDTIME()`

```
mysql> SELECT ADDTIME('1997-12-31 23:59:59.999999','1 1:1:1.000002');
+-----------------------------------------------------+
| ADDTIME('1997-12-31 23:59:59.999999','1 1:1:1.000002') |
+-----------------------------------------------------+
| 1998-01-02 01:01:01.000001                          |
+-----------------------------------------------------+
mysql> SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');
+-------------------------------------------+
| ADDTIME('01:00:00.999999', '02:00:00.999998') |
+-------------------------------------------+
| 03:00:01.999997                           |
+-------------------------------------------+
```

Как видно из листинга 16.11, дата и время суток в обоих аргументах может задаваться произвольно, результат автоматически преобразуется к максимальному формату.

16.3. Функция *CONVERT_TZ()*

Функция `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`. Пример использования функции приводится в листинге 16.12.

> ***Замечание***
>
> Функция `CONVERT_TZ()` появилась в MySQL в версии 4.1.3.

Листинг 16.12. Использование функции `CONVERT_TZ()`

```
mysql> SELECT CONVERT_TZ('2006-07-01 12:00:00','+00:00','+10:00');
+-----------------------------------------------------+
| CONVERT_TZ('2006-07-01 12:00:00','+00:00','+10:00') |
+-----------------------------------------------------+
| 2006-07-01 22:00:00                                 |
+-----------------------------------------------------+
```

16.4. Функция *CURDATE()*

Функция `CURDATE()`, приведенная в листинге 16.13, возвращает текущую дату в формате `YYYY-MM-DD` или `YYYYDDMM` в зависимости от того, вызывается функция в текстовом или числовом контексте.

Листинг 16.13. Использование функции `CURDATE()`

```
mysql> SELECT CURDATE(), CURDATE() + 0, CURRENT_DATE(), CURRENT_DATE;
+------------+--------------+----------------+--------------+
| CURDATE()  | CURDATE() + 0 | CURRENT_DATE() | CURRENT_DATE |
+------------+--------------+----------------+--------------+
| 2006-07-17 |      20060717 | 2006-07-17     | 2006-07-17   |
+------------+--------------+----------------+--------------+
```

> ***Замечание***
>
> Функция `CURDATE()` имеет два синонима: `CURRENT_DATE()` и `CURRENT_DATE`. То есть вариант `CURRENT_DATE` допускает написание без круглых скобок.

В листинге 16.14 демонстрируется вставка новой записи в таблицу `orders` учебной базы данных `shop`, в которой имеется `DATETIME`-поле `ordertime`. Подстановка вместо значения этого поля в SQL-операторе `INSERT` функции `CURDATE()` приводит к автоматическому назначению данному полю текущей даты.

Листинг 16.14. Вставка новой записи в таблицу `orders`

```
mysql> INSERT INTO orders VALUES (NULL, 3, CURDATE(), 1, 20);
mysql> SELECT * FROM orders WHERE id_order = 6;
+----------+---------+---------------------+--------+------------+
| id_order | id_user | ordertime           | number | id_product |
+----------+---------+---------------------+--------+------------+
|        6 |       3 | 2005-05-27 00:00:00 |      1 |         20 |
+----------+---------+---------------------+--------+------------+
```

Как видно из листинга 16.14, часам, минутам и секундам автоматически присваиваются нулевые значения.

16.5. Функция *CURTIME()*

Функция `CURTIME()` возвращает текущее время суток в формате `hh-mm-ss` или `hhmmss` в зависимости от того, вызывается функция в текстовом или числовом контексте (листинг 16.15).

Листинг 16.15. Использование функции `CURTIME()`

```
mysql> SELECT CURTIME(), CURTIME() + 0, CURRENT_TIME(), CURRENT_TIME;
+-----------+---------------+----------------+--------------+
| CURTIME() | CURTIME() + 0 | CURRENT_TIME() | CURRENT_TIME |
+-----------+---------------+----------------+--------------+
| 00:26:08  |          2608 | 00:26:08       | 00:26:08     |
+-----------+---------------+----------------+--------------+
```

Замечание

Функции `CURTIME()` соответствуют два синонима: `CURRENT_TIME()` и `CURRENT_TIME`. То есть вариант `CURRENT_TIME` допускает написание без круглых скобок.

16.6. Функция *DATE()*

Функция `DATE()` извлекает из значения даты и времени суток в формате `DATETIME` дату, отсекая часы, минуты и секунды (листинг 16.16).

Листинг 16.16. Использование функции `DATE()`

```
mysql> SELECT DATE('2006-07-17 00:26:08');
+----------------------------+
| DATE('2006-07-17 00:26:08') |
+----------------------------+
| 2006-07-17                 |
+----------------------------+
```

Замечание

Функция `DATE()` появилась в MySQL в версии 4.1.1.

16.7. Функция *DATEDIFF()*

Функция `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.

Пример запроса приведен в листинге 16.17.

Листинг 16.17. Использование функции `DATEDIFF()`

```
mysql> SELECT DATEDIFF('2006-12-31 23:59:59','2006-12-30');
+-----------------------------------------+
| DATEDIFF('2006-12-31 23:59:59','2006-12-30') |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
mysql> SELECT DATEDIFF('2006-11-30 23:59:59','2006-12-31');
+-----------------------------------------+
| DATEDIFF('2006-11-30 23:59:59','2006-12-31') |
+-----------------------------------------+
|                                     -31 |
+-----------------------------------------+
```

Как видно из листинга 16.17, если первый аргумент *`begin`* меньше, чем второй *`end`*, результат получается отрицательным.

Замечание

Функция `DATEDIFF()` появилась в MySQL, начиная с версии 4.1.1.

16.8. Функция *DATE_FORMAT()*

Функция `DATE_FORMAT(`*`date`*`, `*`format`*`)` форматирует время *`date`* в соответствии со строкой *`format`*. В строке *`format`* могут использоваться определители, представленные в табл. 16.2.

Таблица 16.2. *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.)
`%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 входят в табл. 16.2, выводятся без изменений. Пример использования функции `DATE_FORMAT()` представлен в листинге 16.18.

Листинг 16.18. Использование функции `DATE_FORMAT()`

```
mysql> SELECT DATE_FORMAT('2006-01-01 01:59:59','На дворе %Y год');
+-------------------------------------------------+
| DATE_FORMAT('2006-01-01 01:59:59','На дворе %Y год') |
+-------------------------------------------------+
| На дворе 2006 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 16.19).

Листинг 16.19. Преобразование даты из формата `YYYY-MM-DD` в `DD.MM.YYYY`

```
mysql> SELECT NOW(), DATE_FORMAT(NOW(),'%d.%m.%Y');
+---------------------+------------------------------+
| NOW()               | DATE_FORMAT(NOW(),'%d.%m.%Y') |
+---------------------+------------------------------+
| 2006-07-18 15:42:03 | 18.07.2006                   |
+---------------------+------------------------------+
```

Замечание

Определитель `%f` для микросекунд появился в MySQL, начиная с версии 4.1.0.

При помощи функции `DATE_FORMAT()` время и дату из одного формата можно преобразовывать в любой другой формат. Если входной аргумент *`date`* содержит лишнюю информацию, она будет отброшена; если информации недостаточно, недостающие элементы будут дополнены с присвоением им нулевых значений (листинг 16.20).

Листинг 16.20. Преобразование формата при помощи функции `DATE_FORMAT()`

```
mysql> SELECT NOW(), DATE_FORMAT(NOW(),'%T');
+---------------------+-----------------------+
| NOW()               | DATE_FORMAT(NOW(),'%T') |
+---------------------+-----------------------+
| 2006-07-18 15:51:16 | 15:51:16              |
+---------------------+-----------------------+
```

```
mysql> SELECT NOW(), DATE_FORMAT(NOW(),'%T');
+---------------------+------------------------+
| NOW()               | DATE_FORMAT(NOW(),'%T') |
+---------------------+------------------------+
| 2006-07-18 15:51:16 | 15:51:16               |
+---------------------+------------------------+
mysql> SELECT DATE_FORMAT('2006-07-18', '%Y-%m-%d %T');
+-----------------------------------------+
| DATE_FORMAT('2006-07-18', '%Y-%m-%d %T') |
+-----------------------------------------+
| 2006-07-18 00:00:00                     |
+-----------------------------------------+
```

16.9. Функция *DAY()*

Функция `DAY(date)` является синонимом функции `DAYOFMONTH(date)`, принимает в качестве аргумента дату *date* и возвращает порядковый номер дня в месяце (от 1 до 31). Пример представлен в листинге 16.21.

Листинг 16.21. Использование функции `DAY()`

```
mysql> SELECT DAY('2006-07-03');
+-------------------+
| DAY('2006-07-03') |
+-------------------+
|                 3 |
+-------------------+
```

Как видно из листинга 16.21, результат возвращается без ведущего нуля.

> ***Замечание***
>
> Функция `DAY()` введена в MySQL, начиная с версии 4.1.1. В более ранних реализациях MySQL следует использовать `DAYOFMONTH()`.

16.10. Функция *DAYNAME()*

Функция `DAYNAME(date)` принимает в качестве аргумента дату *date* и возвращает день недели в виде полного английского названия (листинг 16.22).

Листинг 16.22. Использование функции `DAYNAME()`

```
mysql> SELECT DAYNAME('2006-07-30');
+----------------------+
| DAYNAME('2006-07-30') |
+----------------------+
| Sunday               |
+----------------------+
```

Обычно перевод английских названий производят уже в прикладной программе. Соответствие английских и русских названий дней недели приведено в табл. 16.3.

***Таблица 16.3.** Соответствие английских и русских названий дней недели*

Номер	Английское название	Сокращенный вариант	Русское название
0	Sunday	Sun	Воскресенье
1	Monday	Mon	Понедельник
2	Tuesday	Tue	Вторник
3	Wednesday	Wed	Среда
4	Thursday	Thu	Четверг
5	Friday	Fri	Пятница
6	Saturday	Sat	Суббота

Следует помнить, что в западных странах неделя начинается с воскресенья (Sunday), а не с понедельника (Monday), как в Российской Федерации, поэтому нумерация дней недели в MySQL начинается именно с воскресенья.

16.11. Функция *DAYOFMONTH()*

Функция `DAYOFMONTH(date)`, показанная в листинге 16.23, принимает в качестве аргумента дату `date` и возвращает порядковый номер дня в месяце (от 1 до 31).

Листинг 16.23. Использование функции `DAYOFMONTH()`

```
mysql> SELECT DAYOFMONTH('2006-07-08'), DAYOFMONTH('2006-07-20');
+------------------------+-------------------------+
| DAYOFMONTH('2006-07-08') | DAYOFMONTH('2006-07-20') |
+------------------------+-------------------------+
|                      8 |                      20 |
+------------------------+-------------------------+
```

16.12. Функция *DAYOFWEEK()*

Функция `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.

Пример приведен в листинге 16.24.

Листинг 16.24. Использование функции DAYOFWEEK()

```
mysql> SELECT DAYOFWEEK('2006-07-18'), DAYNAME('2006-07-18');
+-------------------------+-----------------------+
| DAYOFWEEK('2006-07-18') | DAYNAME('2006-07-18') |
+-------------------------+-----------------------+
|                       3 | Tuesday               |
+-------------------------+-----------------------+
```

В листинге 16.24 результат соответствует вторнику.

16.13. Функция *DAYOFYEAR()*

Функция `DAYOFYEAR(date)` принимает в качестве аргумента дату *date* и возвращает порядковый номер дня в году (от 1 до 366). Пример использования функции `DAYOFYEAR()` показан в листинге 16.25.

Листинг 16.25. Использование функции DAYOFYEAR()

```
mysql> SELECT DAYOFYEAR('2006-02-23'), DAYOFYEAR('2006-12-31');
+-------------------------+-------------------------+
| DAYOFYEAR('2006-02-23') | DAYOFYEAR('2006-12-31') |
+-------------------------+-------------------------+
|                      54 |                     365 |
+-------------------------+-------------------------+
```

16.14. Функция *EXTRACT()*

Функция `EXTRACT(type FROM datetime)` получает дату и время суток *datetime* и возвращает часть, определяемую параметром *type*, который может принимать значения из первого столбца табл. 16.1 (листинг 16.26).

Листинг 16.26. Использование функции `EXTRACT()`

```
mysql> SELECT EXTRACT(YEAR FROM '2006-12-31 14:30:10');
+-----------------------------------------+
| EXTRACT(YEAR FROM '2006-12-31 14:30:10') |
+-----------------------------------------+
|                                    2006 |
+-----------------------------------------+
mysql> SELECT EXTRACT(MINUTE FROM '2006-12-31 14:30:10');
+-------------------------------------------+
| EXTRACT(MINUTE FROM '2006-12-31 14:30:10') |
+-------------------------------------------+
|                                         30 |
+-------------------------------------------+
mysql> SELECT EXTRACT(DAY FROM '2006-12-31 14:30:10');
+---------------------------------------+
| EXTRACT(DAY FROM '2006-12-31 14:30:10') |
+---------------------------------------+
|                                    31 |
+---------------------------------------+
mysql> SELECT EXTRACT(HOUR_SECOND FROM '2006-12-31 14:30:10');
+-----------------------------------------------+
| EXTRACT(HOUR_SECOND FROM '2006-12-31 14:30:10') |
+-----------------------------------------------+
|                                        143010 |
+-----------------------------------------------+
```

Помимо полного формата времени `YYYY-MM-DD hh:mm:ss`, допускается использование укороченной формы, включающей только дату `YYYY-MM-DD` (листинг 16.27).

Листинг 16.27. Использование укороченного формата `YYYY-MM-DD`

```
mysql> SELECT EXTRACT(DAY FROM '2006-12-31');
+-------------------------------+
| EXTRACT(DAY FROM '2006-12-31') |
+-------------------------------+
|                            31 |
+-------------------------------+
```

Извлечение при помощи функции `EXTRACT()` микросекунд, секунд, минут и часов из укороченного варианта времени, включающего только дату `YYYY-MM-DD`, приводит к непредсказуемому результату.

16.15. Функция *FROM_DAYS()*

Функция `FROM_DAYS(N)` (листинг 16.28) принимает количество дней *N*, прошедших с нулевого года, и возвращает дату в формате `YYYY-MM-DD`.

Листинг 16.28. Использование функции `FROM_DAYS()`

```
mysql> SELECT FROM_DAYS(0), FROM_DAYS(732419);
+--------------+-------------------+
| FROM_DAYS(0) | FROM_DAYS(732419) |
+--------------+-------------------+
| 0000-00-00   | 2005-04-18        |
+--------------+-------------------+
```

Обычно функция `FROM_DAYS()` используется совместно с функцией `TO_DAYS(date)`, которая решает обратную задачу — преобразует дату `date` из формата `YYYY-MM-DD` в число дней, прошедших с нулевого года (листинг 16.29).

Листинг 16.29. Совместное использование функций `FROM_DAYS()` и `TO_DAYS()`

```
mysql> SELECT TO_DAYS('2010-10-08');
+-----------------------+
| TO_DAYS('2010-10-08') |
+-----------------------+
|                734418 |
+-----------------------+
mysql> SELECT FROM_DAYS(734418 + 10);
+------------------------+
| FROM_DAYS(734418 + 10) |
+------------------------+
| 2010-10-18             |
+------------------------+
```

16.16. Функция *FROM_UNIXTIME()*

Функция `FROM_UNIXTIME(unix_timestamp)` 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 16.30. Использование функции FROM_UNIXTIME()

```
mysql> 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 16.30, 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.

Обычно функцию `FROM_UNIXTIME()` используют совместно с функцией `UNIX_TIMESTAMP(`*`date`*`)`, которая решает обратную задачу, принимая дату и время суток *`date`* и возвращая число секунд, прошедшее с полуночи 1 января 1970 года (листинг 16.31).

Листинг 16.31. Совместное использование функций UNIX_TIMESTAMP() и FROM_UNIXTIME()

```
mysql> SELECT UNIX_TIMESTAMP('2010-10-08 00:00:00');
+---------------------------------------+
| UNIX_TIMESTAMP('2010-10-08 00:00:00') |
+---------------------------------------+
|                            1286481600 |
+---------------------------------------+
mysql> SELECT FROM_UNIXTIME(1286481600 + 3600 * 24 * 30);
+-----------------------------------------+
| FROM_UNIXTIME(1286481600 + 3600 * 24 * 30) |
+-----------------------------------------+
| 2010-11-06 23:00:00                      |
+-----------------------------------------+
```

Функция `FROM_UNIXTIME(`*`unix_timestamp`*`, `*`format`*`)` может принимать второй параметр *`format`*, который представляет собой строку, содержащую определители из табл. 16.2. Использование определителей позволяет отформатировать строку, например, так, как это представлено в листинге 16.32.

Листинг 16.32. Форматирование результата в функции FROM_UNIX()

```
mysql> SELECT
    -> FROM_UNIXTIME(1286481600,'В %Y году контракт заканчивается');
+-------------------------------------------------------------+
| FROM_UNIXTIME(1286481600,'В %Y году контракт заканчивается') |
+-------------------------------------------------------------+
| В 2010 году контракт заканчивается                          |
+-------------------------------------------------------------+
```

16.17. Функция *GET_FORMAT()*

Данная функция возвращает строку форматирования, соответствующую одному из пяти стандартов времени. Полученную строку можно затем использовать в функциях `DATE_FORMAT()` и `STR_TO_DATE()`. Функция имеет следующий синтаксис:

```
GET_FORMAT (DATE|TIME|DATETIME, 'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL')
```

В качестве первого параметра выступает одно из трех ключевых слов:

- `DATE` — получить формат для даты;
- `TIME` — получить формат для времени;
- `DATETIME` — получить формат для даты и времени.

Второй параметр задает название стандарта, согласно которому следует отформатировать время, и может принимать одну из строк:

- `'EUR'` — европейский стандарт IBM;
- `'USA'` — американский стандарт IBM;
- `'JIS'` — японский индустриальный стандарт;
- `'ISO'` — стандарт ISO (Международная организация стандартов);
- `'INTERNAL'` — интернациональный стандарт.

Пример использования функции `GET_FORMAT()` приведен в листинге 16.33.

> ***Замечание***
>
> Функция `GET_FORMAT()` введена в MySQL, начиная с версии 4.1.1.

Листинг 16.33. Использование функции GET_FORMAT()

```
mysql> SELECT GET_FORMAT(DATE,'EUR');
+-----------------------+
| GET_FORMAT(DATE,'EUR') |
+-----------------------+
| %d.%m.%Y              |
+-----------------------+
```

Представление времени во всех пяти стандартах приведено в табл. 16.4.

Таблица 16.4. *Представление времени в различных стандартах*

Вызов функции	Результат
GET_FORMAT(DATE,'USA')	%m.%d.%Y
GET_FORMAT(DATE,'JIS')	%Y-%m-%d
GET_FORMAT(DATE,'ISO')	%Y-%m-%d
GET_FORMAT(DATE,'EUR')	%d.%m.%Y
GET_FORMAT(DATE,'INTERNAL')	%Y%m%d
GET_FORMAT(DATETIME,'USA')	%Y-%m-%d-%H.%i.%s
GET_FORMAT(DATETIME,'JIS')	%Y-%m-%d %H:%i:%s
GET_FORMAT(DATETIME,'ISO')	%Y-%m-%d %H:%i:%s
GET_FORMAT(DATETIME,'EUR')	%Y-%m-%d-%H.%i.%s
GET_FORMAT(DATETIME,'INTERNAL')	%Y%m%d%H%i%s
GET_FORMAT(TIME,'USA')	%h:%i:%s %p
GET_FORMAT(TIME,'JIS')	%H:%i:%s
GET_FORMAT(TIME,'ISO')	%H:%i:%s
GET_FORMAT(TIME,'EUR')	%H.%i.%S
GET_FORMAT(TIME,'INTERNAL')	%H%i%s

Пример использования функции `GET_FORMAT()` совместно с `DATE_FORMAT()` приведен в листинге 16.34.

Листинг 16.34. Совместное использование `GET_FORMAT()` и `DATE_FORMAT()`

```
mysql> SELECT DATE_FORMAT('2006-07-03',GET_FORMAT(DATE,'EUR'));
+-----------------------------------------------+
| DATE_FORMAT('2006-07-03',GET_FORMAT(DATE,'EUR')) |
+-----------------------------------------------+
| 03.07.2006                                    |
+-----------------------------------------------+
```

16.18. Функция *HOUR()*

Функция `HOUR(`*`datetime`*`)` возвращает значение часа (от 0 до 23) для времени *`datetime`* (листинг 16.35).

Листинг 16.35. Использование функции HOUR()

```
mysql> SELECT HOUR('12:10:45'), HOUR('2006-08-20 12:10:45');
+-----------------+---------------------------+
| HOUR('12:10:45') | HOUR('2006-08-20 12:10:45') |
+-----------------+---------------------------+
|               12 |                        12 |
+-----------------+---------------------------+
```

Одной из особенностей функции HOUR() является тот факт, что если время суток выходит за 24-часовую границу, функция вернет большее число (листинг 16.36).

Листинг 16.36. Выход за границу 24-часового дня

```
mysql> SELECT HOUR('248:10:45');
+-------------------+
| HOUR('248:10:45') |
+-------------------+
|               248 |
+-------------------+
```

16.19. Функция *LAST_DAY()*

Функция 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 (листинг 16.37).

Листинг 16.37. Использование функции LAST_DAY()

```
mysql> SELECT LAST_DAY('2003-02-05'), LAST_DAY('2004-02-05');
+-----------------------+-----------------------+
| LAST_DAY('2006-02-05') | LAST_DAY('2004-02-05') |
+-----------------------+-----------------------+
| 2006-02-28             | 2004-02-29             |
+-----------------------+-----------------------+
mysql> SELECT LAST_DAY('2006-01-01 01:01:01');
+--------------------------------+
| LAST_DAY('2006-01-01 01:01:01') |
+--------------------------------+
```

```
| 2006-01-31                    |
+-------------------------------+
mysql> SELECT LAST_DAY('2006-07-32');
+-----------------------+
| LAST_DAY('2006-07-32') |
+-----------------------+
| NULL                  |
+-----------------------+
```

Замечание

Функция `LAST_DAY()` введена в MySQL, начиная с версии 4.1.1.

16.20. Функция *MAKEDATE()*

Функция `MAKEDATE(year, dayofyear)` принимает в качестве параметров год *year*, номер дня в году *dayofyear* и возвращает дату в формате `YYYY-MM-DD` (листинг 16.38).

Листинг 16.38. Использование функции `MAKEDATE()`

```
mysql> SELECT MAKEDATE(2006,31), MAKEDATE(2006,32);
+------------------+------------------+
| MAKEDATE(2006,31) | MAKEDATE(2006,32) |
+------------------+------------------+
| 2006-01-31        | 2006-02-01        |
+------------------+------------------+
mysql> SELECT MAKEDATE(2006,365), MAKEDATE(2004,365);
+-------------------+-------------------+
| MAKEDATE(2006,365) | MAKEDATE(2004,365) |
+-------------------+-------------------+
| 2006-12-31         | 2004-12-30         |
+-------------------+-------------------+
mysql> SELECT MAKEDATE(2006,0), MAKEDATE(2001,400);
+-----------------+-------------------+
| MAKEDATE(2006,0) | MAKEDATE(2001,400) |
+-----------------+-------------------+
| NULL             | 2002-02-04         |
+-----------------+-------------------+
```

Параметр *dayofyear* должен быть строго больше нуля, в противном случае возвращается NULL.

Замечание

Функция MAKEDATE() введена в MySQL, начиная с версии 4.1.1.

16.21. Функция *MAKETIME()*

Функция 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 (листинг 16.39).

Листинг 16.39. Использование функции MAKETIME()

```
mysql> SELECT MAKETIME(12,15,30), MAKETIME(12,60,0), MAKETIME(120,15,30);
+------------------+------------------+--------------------+
| MAKETIME(12,15,30) | MAKETIME(12,60,0) | MAKETIME(120,15,30) |
+------------------+------------------+--------------------+
| 12:15:30           | NULL             | 120:15:30          |
+------------------+------------------+--------------------+
```

Выход параметра *hour* за границу 24-часового диапазона допускается, в то время как выход за границу параметров *minute* и *second* приводит к NULL.

Замечание

Функция MAKETIME() введена в MySQL, начиная с версии 4.1.1.

16.22. Функция *MICROSECOND()*

Функция MICROSECOND(*datetime*) извлекает из временного значения *datetime* микросекунды (листинг 16.40).

Листинг 16.40. Использование функции MICROSECOND()

```
mysql> SELECT MICROSECOND('12:00:00.123456');
+-------------------------------+
| MICROSECOND('12:00:00.123456') |
+-------------------------------+
|                         123456 |
+-------------------------------+
```

```
mysql> SELECT MICROSECOND('2006-07-31 23:59:59.000010');
+-----------------------------------------+
| MICROSECOND('2006-07-31 23:59:59.000010') |
+-----------------------------------------+
|                                      10 |
+-----------------------------------------+
```

Замечание

Функция `MICROSECOND()` введена в MySQL, начиная с версии 4.1.1.

16.23. Функция *MINUTE()*

Функция `MINUTE(datetime)` возвращает значение минут (от 0 до 59) для времени суток *datetime* (листинг 16.41).

Листинг 16.41. Использование функции `MINUTE()`

```
mysql> SELECT MINUTE('2006-07-03 10:05:03'), MINUTE('11:59:59');
+-----------------------------+-------------------+
| MINUTE('2006-07-03 10:05:03') | MINUTE('11:59:59') |
+-----------------------------+-------------------+
|                             5 |                59 |
+-----------------------------+-------------------+
```

16.24. Функция *MONTH()*

Функция `MONTH(datetime)` возвращает числовое значение месяца года (от 1 до 12) для даты *datetime* (листинг 16.42).

Листинг 16.42. Использование функции `MONTH()`

```
mysql> SELECT MONTH('2006-02-03'), MONTH('2006-08-12 10:00:00');
+--------------------+-----------------------------+
| MONTH('2006-02-03') | MONTH('2006-08-12 10:00:00') |
+--------------------+-----------------------------+
|                  2 |                           8 |
+--------------------+-----------------------------+
```

16.25. Функция *MONTHNAME()*

Функция `MONTHNAME(datetime)` возвращает строку с названием месяца для даты *datetime* (листинг 16.43).

Листинг 16.43. Использование функции MONTHNAME()

```
mysql> SELECT MONTHNAME('2006-02-05'), MONTHNAME('2006-08-12 10:10:00');
+------------------------+---------------------------------+
| MONTHNAME('2006-02-05') | MONTHNAME('2006-08-12 10:10:00') |
+------------------------+---------------------------------+
| February                | August                          |
+------------------------+---------------------------------+
```

Обычно перевод английских названий производят уже в прикладной программе. Соответствие английских и русских названий месяцев приведено в табл. 16.5.

Таблица 16.5. *Соответствие английских и русских названий месяцев*

Номер	Английское название	Сокращенный вариант	Русское название
1	January	Jan	Январь
2	February	Feb	Февраль
3	March	Mar	Март
4	April	Apr	Апрель
5	May	May	Май
6	June	Jun	Июнь
7	Jule	Jul	Июль
8	August	Aug	Август
9	September	Sep	Сентябрь
10	October	Oct	Октябрь
11	November	Nov	Ноябрь
12	December	Dec	Декабрь

16.26. Функция *NOW()*

Функция `NOW()` 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 (листинг 16.44).

Замечание

Функция `NOW()` имеет пять синонимов: `LOCALTIME()`, `LOCALTIME`, `LOCALTIMESTAMP()`, `LOCALTIMESTAMP`, `SYSDATE()`.

Листинг 16.44. Использование функции NOW()

```
mysql> SELECT NOW()+0, NOW(), DATE_FORMAT(NOW(),'%Y');
+---------------+---------------------+-----------------------+
| NOW()+0       | NOW()               | DATE_FORMAT(NOW(),'%Y') |
+---------------+---------------------+-----------------------+
| 20060719184838 | 2006-07-19 18:48:38 | 2006                  |
+---------------+---------------------+-----------------------+
```

16.27. Функция *PERIOD_ADD()*

Функция 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 (листинг 16.45).

Листинг 16.45. Использование функции PERIOD_ADD()

```
mysql> SELECT PERIOD_ADD(200610,2), PERIOD_ADD(0610,6),
    -> PERIOD_ADD(0610,-6);
+---------------------+-------------------+---------------------+
| PERIOD_ADD(200610,2) | PERIOD_ADD(0610,6) | PERIOD_ADD(0610,-6) |
+---------------------+-------------------+---------------------+
|              200612 |            200704 |              200604 |
+---------------------+-------------------+---------------------+
```

Результат представляет собой число в формате YYYYMM.

16.28. Функция *PERIOD_DIFF()*

Функция PERIOD_DIFF(*period1*, *period2*) вычисляет разницу в месяцах между двумя датами *period1* и *period2*, которые представлены в числовом формате YYYYMM или YYMM (листинг 16.46). Передача в качестве аргументов даты в любом другом формате приводит к непредсказуемому результату.

Листинг 16.46. Использование функции PERIOD_DIFF()

```
mysql> SELECT PERIOD_DIFF(200607, 200507), PERIOD_DIFF(200611, 0702);
+---------------------------+--------------------------+
| PERIOD_DIFF(200607, 200507) | PERIOD_DIFF(200611, 0702) |
+---------------------------+--------------------------+
|                        12 |                       -3 |
+---------------------------+--------------------------+
```

Результат представляет собой целое число: положительное, если первый аргумент больше второго, и отрицательное, если первый аргумент меньше второго.

16.29. Функция *QUARTER()*

Функция `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` (листинг 16.47).

Листинг 16.47. Использование функции `QUARTER()`

```
mysql> SELECT QUARTER('2006-01-01'), QUARTER('2006-12-01 23:00:00');
+----------------------+------------------------------+
| QUARTER('2006-01-01') | QUARTER('2006-12-01 23:00:00') |
+----------------------+------------------------------+
|                     1 |                            4 |
+----------------------+------------------------------+
```

16.30. Функция *SECOND()*

Функция `SECOND(`*`time`*`)` возвращает число секунд для времени суток *time*, которое задается либо в виде строки `'hh:mm:ss'`, либо числа `hhmmss` (листинг 16.48). Передача в качестве аргумента даты в любом другом формате приводит к непредсказуемому результату.

Листинг 16.48. Использование функции `SECOND()`

```
mysql> SELECT SECOND(120012), SECOND('10:10:10');
+---------------+-------------------+
| SECOND(120012) | SECOND('10:10:10') |
+---------------+-------------------+
|             12 |                 10 |
+---------------+-------------------+
```

16.31. Функция *SEC_TO_TIME()*

Функция `SEC_TO_TIME(`*`seconds`*`)` принимает число секунд *seconds*, прошедшее от начала суток и возвращает время в формате `hh:mm:ss` или `hhmmss` в зависимости от того, вызывается функция в строковом или числовом контексте (листинг 16.49).

Листинг 16.49. Использование функции SEC_TO_TIME()

```
mysql> SELECT SEC_TO_TIME(1), SEC_TO_TIME(36584), SEC_TO_TIME(100584);
+---------------+-------------------+--------------------+
| SEC_TO_TIME(1) | SEC_TO_TIME(36584) | SEC_TO_TIME(100584) |
+---------------+-------------------+--------------------+
| 00:00:01       | 10:09:44           | 27:56:24            |
+---------------+-------------------+--------------------+
```

Замечание

Обратную задачу, т. е. преобразование времени в формате `hh:mm:ss` в число секунд, прошедших от начала суток, выполняет функция `TIME_TO_SEC()`.

16.32. Функция *STR_TO_DATE()*

Функция `STR_TO_DATE(`*`str`*`, `*`format`*`)` является противоположной функцией `DATE_FORMAT`. Данная функция принимает дату и время суток в виде строки *`str`*, соответствующей формату, задаваемому параметром *`format`*, которая содержит определители из табл. 16.2. В результате функция возвращает время в формате MySQL `YYYY-MM-DD hh:mm:ss` (листинг 16.50).

Замечание

Функция `STR_TO_DATE()` появилась в MySQL, начиная с версии 4.1.1.

Листинг 16.50. Использование функции STR_TO_DATE()

```
mysql> SELECT STR_TO_DATE('03.07.2006 09.20','%d.%m.%Y %H.%i');
+-----------------------------------------------+
| STR_TO_DATE('03.07.2006 09.20','%d.%m.%Y %H.%i') |
+-----------------------------------------------+
| 2006-07-03 09:20:00                           |
+-----------------------------------------------+
mysql> SELECT STR_TO_DATE('10arp', '%carp');
+-----------------------------+
| STR_TO_DATE('10arp', '%carp') |
+-----------------------------+
| 0000-10-00                  |
+-----------------------------+
mysql> SELECT STR_TO_DATE('2006-15-10 00:00:00','%Y-%m-%d %H:%i:%s');
```

```
+-----------------------------------------------------+
| STR_TO_DATE('2006-15-10 00:00:00','%Y-%m-%d %H:%i:%s') |
+-----------------------------------------------------+
| NULL                                                |
+-----------------------------------------------------+
```

Значение `NULL` в последнем примере вызвано передачей функции некорректного значения месяца (`15`).

16.33. Функция *SUBDATE()*

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 16.51).

> ***Замечание***
>
> Подробное описание параметров функции приведено в *разд. 16.1*.

> ***Замечание***
>
> Функция `SUBDATE()` имеет синоним `DATE_SUB()`.

Листинг 16.51. Использование функции `SUBDATE()`

```
mysql> SELECT ADDDATE('06-07-10', INTERVAL -5 DAY);
+-------------------------------------+
| ADDDATE('06-07-10', INTERVAL -5 DAY) |
+-------------------------------------+
| 2006-07-05                          |
+-------------------------------------+
mysql> SELECT SUBDATE('06-07-10', INTERVAL 5 DAY);
+-------------------------------------+
| SUBDATE('06-07-10', INTERVAL 5 DAY) |
+-------------------------------------+
| 2006-07-05                          |
+-------------------------------------+
```

16.34. Функция *SUBTIME()*

Функция `SUBTIME(`*`datetime`*`, `*`time`*`)` вычитает из временной величины *`datetime`* время суток *`time`* и возвращает результат (листинг 16.52). Величина *`datetime`* может быть задана либо в полном формате `YYYY-MM-DD hh:mm:ss`, либо в кратком формате `hh:mm:ss`, величину *`time`* следует задавать только в кратком формате `hh:mm:ss`.

Версия

Функция `SUBTIME()` появилась в MySQL, начиная с версии 4.1.1.

Листинг 16.52. Использование функции `SUBTIME()`

```
mysql> SELECT SUBTIME('2006-12-31 23:59:59.999999','1:1:1.000002');
+----------------------------------------------------+
| SUBTIME('2006-12-31 23:59:59.999999','1:1:1.000002') |
+----------------------------------------------------+
| 2006-12-31 22:58:58.999997                         |
+----------------------------------------------------+
mysql> SELECT SUBTIME('01:00:00', '02:00:00');
+---------------------------------+
| SUBTIME('01:00:00', '02:00:00') |
+---------------------------------+
| -01:00:00                       |
+---------------------------------+
```

16.35. Функция *TIME()*

Функция `TIME(`*`datetime`*`)` извлекает время суток в формате `hh:mm:ss` из даты и времени *`datetime`* (листинг 16.53).

Версия

Функция `TIME()` появилась в MySQL, начиная с версии 4.1.1.

Листинг 16.53. Использование функции `TIME()`

```
mysql> SELECT TIME('2006-07-31 01:02:03');
+----------------------------+
| TIME('2006-07-31 01:02:03') |
+----------------------------+
| 01:02:03                   |
+----------------------------+
mysql> SELECT TIME('2006-07-31 01:02:03.000123');
+-----------------------------------+
| TIME('2006-07-31 01:02:03.000123') |
+-----------------------------------+
| 01:02:03.000123                   |
+-----------------------------------+
```

16.36. Функция *TIMEDIFF()*

Функция `TIMEDIFF(expr, expr2)` возвращает разницу между временными значениями, заданными параметрами *expr* и *expr2* (листинг 16.54).

Версия

Функция `TIMEDIFF()` появилась в MySQL, начиная с версии 4.1.1.

Листинг 16.54. Использование функции `TIMEDIFF()`

```
mysql> SELECT TIMEDIFF('2006:01:01 00:00:00','2006:01:01 00:00:00.000001');
+-----------------------------------------------------------+
| TIMEDIFF('2006:01:01 00:00:00','2006:01:01 00:00:00.000001') |
+-----------------------------------------------------------+
| -00:00:00.000001                                          |
+-----------------------------------------------------------+
mysql> SELECT TIMEDIFF('2006-12-31 23:59:59','2006-12-30 01:01:01');
+-----------------------------------------------------+
| TIMEDIFF('2006-12-31 23:59:59','2006-12-30 01:01:01') |
+-----------------------------------------------------+
| 46:58:58                                            |
+-----------------------------------------------------+
```

16.37. Функция *TIMESTAMP()*

Функция `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` (листинг 16.55).

Замечание

Функция `TIMESTAMP()` появилась в MySQL, начиная с версии 4.1.1.

Листинг 16.55. Использование функции `TIMESTAMP()`

```
mysql> SELECT TIMESTAMP('2006-12-31'), TIMESTAMP('2006-12-31 15:00:10');
+----------------------+---------------------------------+
| TIMESTAMP('2006-12-31') | TIMESTAMP('2006-12-31 15:00:10') |
+----------------------+---------------------------------+
| 2006-12-31 00:00:00     | 2006-12-31 15:00:10             |
+----------------------+---------------------------------+
```

Как видно из листинга 16.55, для даты и времени суток в полном формате `YYYY-MM-DD hh:mm:ss` функция возвращает значение параметра *expr* без изменений.

Данная функция может принимать второй необязательный аргумент *time*, который представляет собой время в формате `hh:mm:ss`. Если второй аргумент присутствует, то функция осуществляет сложение аргументов (листинг 16.56).

Листинг 16.56. Использование второго параметра в функции `TIMESTAMP()`

```
mysql> SELECT TIMESTAMP('2006-12-31','01:00:10');
+-----------------------------------+
| TIMESTAMP('2006-12-31','01:00:10') |
+-----------------------------------+
| 2006-12-31 01:00:10               |
+-----------------------------------+
```

16.38. Функция *TIMESTAMPADD()*

Функция `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*, принимающего значения из табл. 16.6.

Таблица 16.6. *Значения параметра* `interval`

Значение	Единицы измерения
FRAC_SECOND	Микросекунды
SECOND	Секунды
MINUTE	Минуты
HOUR	Часы
DAY	Дни
WEEK	Недели
MONTH	Месяцы
QUARTER	Кварталы
YEAR	Года

Замечание

Для каждого из приведенных в списке значений существует синоним, отличающийся от них префиксом `SQL_TSI_`. Например, для `DAY` вполне допустимо использование синонима `SQL_TSI_DAY`.

Замечание

Функция `TIMESTAMPADD()` появилась в MySQL, начиная с версии 5.0.0.

Функция возвращает результат в полном формате `YYYY-MM-DD hh:mm:ss` (листинг 16.57).

Листинг 16.57. Использование функции `TIMESTAMPADD()`

```
mysql> SELECT TIMESTAMPADD(FRAC_SECOND,1,'2006-01-02');
+----------------------------------------+
| TIMESTAMPADD(FRAC_SECOND,1,'2006-01-02') |
+----------------------------------------+
| 2006-01-02 00:00:00.000001               |
+----------------------------------------+
```

16.39. Функция *TIMESTAMPDIFF()*

Функция `TIMESTAMPDIFF(`*`interval`*`, `*`datetime_expr1`*`, `*`datetime_expr2`*`)` возвращает разницу между двумя датами *`datetime_expr1`* или *`datetime_expr2`*, заданными в кратком `YYYY-MM-DD` или полном `YYYY-MM-DD hh:mm:ss` форматах (листинг 16.58). Единицы измерения, в которых функция возвращает результат, задаются параметром *`interval`*, принимающим значения из табл. 16.6.

Версия

Функция `TIMESTAMPADD()` появилась в MySQL, начиная с версии 5.0.0.

Листинг 16.58. Использование функции `TIMESTAMPDIFF()`

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2006-02-01','2006-05-01');
+-----------------------------------------------+
| TIMESTAMPDIFF(MONTH,'2006-02-01','2006-05-01') |
+-----------------------------------------------+
|                                             3 |
+-----------------------------------------------+
mysql> SELECT TIMESTAMPDIFF(YEAR,'2006-05-01','2005-01-01');
+----------------------------------------------+
| TIMESTAMPDIFF(YEAR,'2006-05-01','2005-01-01') |
+----------------------------------------------+
|                                           -1 |
+----------------------------------------------+
```

Как видно из листинга 16.58, функция возвращает положительную величину, если *datetime_expr2* больше *datetime_expr1*, и отрицательную в противном случае.

16.40. Функция *TIME_FORMAT()*

Функция TIME_FORMAT(*time*, *format*) форматирует время *time*, которое задается в виде hh:mm:ss, согласно строке форматирования *fomat*, содержащей определители из табл. 16.2. Пример представлен в листинге 16.59.

> ***Замечание***
>
> Строка формата *format* функции TIME_FORMAT() аналогична используемой в синтаксисе функции DATE_FORMAT(), но в TIME_FORMAT() указываются только определители, относящиеся к часам, минутам, секундам и микросекундам. Использование остальных определителей приводит к результату, равному NULL или 0.

Листинг 16.59. Использование функции TIME_FORMAT()

```
mysql> SELECT TIME_FORMAT('100:45:10', '%H.%i.%s');
+--------------------------------------+
| TIME_FORMAT('100:45:10', '%H.%i.%s') |
+--------------------------------------+
| 100.45.10                            |
+--------------------------------------+
```

Как видно из листинга 16.59, для часов допустимо использование величин, превышающих 24-часовой интервал, для минут и секунд величины больше 59 не допустимы.

16.41. Функция *TIME_TO_SEC()*

Функция TIME_TO_SEC(*time*) принимает в качестве аргумента время суток *time* hh:mm:ss и возвращает число секунд, прошедшее с начала суток. Данная функция часто используется совместно с функцией SEC_TO_TIME(), которая решает обратную задачу (листинг 16.60).

Листинг 16.60. Использование функции TIME_TO_SEC()

```
mysql> SELECT TIME_TO_SEC('22:23:00');
+-------------------------+
| TIME_TO_SEC('22:23:00') |
+-------------------------+
|                   80580 |
+-------------------------+
```

```
mysql> SELECT SEC_TO_TIME(80580 + 3600);
+--------------------------+
| SEC_TO_TIME(80580 + 3600) |
+--------------------------+
| 23:23:00                 |
+--------------------------+
```

16.42. Функция *TO_DAYS()*

Функция `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 (листинг 16.61).

Листинг 16.61. Использование функции `TO_DAYS()`

```
mysql> SELECT TO_DAYS('1582-01-01'), TO_DAYS('2005-10-10');
+----------------------+----------------------+
| TO_DAYS('1582-01-01') | TO_DAYS('2005-10-10') |
+----------------------+----------------------+
|               577814 |               732594 |
+----------------------+----------------------+
```

Использование функции `TO_DAYS()` обычно происходит совместно с функцией `FROM_DAYS()`, которая решает обратную задачу — преобразует число дней, прошедших с нулевого года в дату `YYYY-MM-DD` (листинг 16.62).

Листинг 16.62. Совместное использование функций `TO_DAYS()` и `FROM_DAYS()`

```
mysql> SELECT TO_DAYS('2005-10-10');
+----------------------+
| TO_DAYS('2005-10-10') |
+----------------------+
|               732594 |
+----------------------+
mysql> SELECT FROM_DAYS(732594 + 31);
+------------------------+
| FROM_DAYS(732594 + 31) |
+------------------------+
| 2005-11-10             |
+------------------------+
```

16.43. Функция *UNIX_TIMESTAMP()*

Функция `UNIX_TIMESTAMP()` возвращает число секунд, прошедших с полуночи 1 января 1970 года (листинг 16.63). 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.

Листинг 16.63. Использование функции `UNIX_TIMESTAMP()`

```
mysql> SELECT UNIX_TIMESTAMP(),UNIX_TIMESTAMP('1997-10-04 22:23:00');
+------------------+---------------------------------------+
| UNIX_TIMESTAMP() | UNIX_TIMESTAMP('1997-10-04 22:23:00') |
+------------------+---------------------------------------+
|       1113999060 |                             875989380 |
+------------------+---------------------------------------+
```

Замечание

Если функции `UNIX_TIMESTAMP()` через параметр *`datetime`* передается величина с датой до 1 января 1970 года или после 1 января 2038 года, возвращается 0.

16.44. Функция *UTC_DATE()*

Функция `UTC_DATE()` возвращает текущую дату в виде строки `YYYY-MM-DD` или числа `YYYYMMDD` в зависимости от того, в каком контексте вызывается функция (листинг 16.64). В отличие от функций `NOW()` или `CURDATE()`, возвращающих локальную дату, в `UTC_DATE()` время исчисляется по Гринвичу.

Замечание

Функция `UTC_DATE()` появилась в MySQL, начиная с версии 4.1.1.

Листинг 16.64. Использование функции `UTC_DATE()`

```
mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
+------------+---------------+
| UTC_DATE() | UTC_DATE() + 0 |
+------------+---------------+
| 2006-07-20 |       20060720 |
+------------+---------------+
```

Замечание

Для функции `UTC_DATE()` имеется синоним `UTC_DATE`, т. е. возможно использование функции без круглых скобок.

16.45. Функция *UTC_TIME()*

Функция `UTC_TIME()` возвращает текущее время суток в виде строки `hh:mm:ss` или числа `hhmmss` в зависимости от того, в каком контексте вызывается функция (листинг 16.65). В отличие от функций `NOW()` и `CURTIME()`, возвращающих локальную дату, в `UTC_DATE()` время исчисляется по Гринвичу.

Замечание

Функция `UTC_TIME()` появилась в MySQL, начиная с версии 4.1.1.

Листинг 16.65. Использование функции `UTC_TIME()`

```
mysql> SELECT UTC_TIME(), UTC_TIME + 0, NOW();
+------------+-------------+---------------------+
| UTC_TIME() | UTC_TIME + 0 | NOW()               |
+------------+-------------+---------------------+
| 12:40:29   |       124029 | 2006-07-20 16:40:29 |
+------------+-------------+---------------------+
```

Замечание

Для функции `UTC_TIME()` имеется синоним `UTC_TIME`, т. е. возможно использование функции без круглых скобок.

16.46. Функция *UTC_TIMESTAMP()*

Функция `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 (листинг 16.66). В отличие от функции `NOW()`, возвращающей локальную дату, в `UTC_DATE()` время исчисляется по Гринвичу.

Замечание

Функция `UTC_TIMESTAMP()` появилась в MySQL, начиная с версии 4.1.1.

Листинг 16.66. Использование функции `UTC_TIMESTAMP()`

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP + 0, NOW();
+---------------------+--------------------+---------------------+
| UTC_TIMESTAMP()     | UTC_TIMESTAMP + 0 | NOW()               |
+---------------------+--------------------+---------------------+
| 2006-07-20 13:00:27 |    20060720130027 | 2006-07-20 17:00:27 |
+---------------------+--------------------+---------------------+
```

Замечание

Для функции `UTC_TIMESTAMP()` имеется синоним `UTC_TIMESTAMP`, т. е. возможно использование функции без круглых скобок.

16.47. Функция *WEEK()*

Функция `WEEK(date)` возвращает номер недели (от 0 до 53) в году для даты *date* (листинг 16.67). Предполагается, что неделя начинается с воскресенья.

Листинг 16.67. Использование функции `WEEK()`

```
mysql> SELECT WEEK('2006-01-31'), WEEK('2006-12-31');
+--------------------+--------------------+
| WEEK('2006-01-31') | WEEK('2006-12-31') |
+--------------------+--------------------+
|                  5 |                 53 |
+--------------------+--------------------+
```

Функция также допускает наличие второго необязательного параметра `WEEK(date, mode)` (листинг 16.68), который представляет собой число из первого столбца табл. 16.7 и определяет порядок вычисления номера недели.

Таблица 16.7. *Значение второго параметра функции `WEEK()`*

Значение *mode*	Первый день недели	Диапазон	Неделя с номером 1 — это первая неделя
0	Воскресенье	0—53	С воскресеньем в этом году
1	Понедельник	0—53	С более чем 3 днями в этом году
2	Воскресенье	1—53	С воскресеньем в этом году
3	Понедельник	1—53	С более чем 3 днями в этом году
4	Воскресенье	0—53	С более чем 3 днями в этом году
5	Понедельник	0—53	С понедельником в этом году
6	Воскресенье	1—53	С более чем 3 днями в этом году
7	Понедельник	1—53	С понедельником в этом году

По умолчанию, когда второй аргумент *mode* не указывается, ему присваивается значение 4.

Замечание

До версии MySQL 4.0.17 вычисления по умолчанию проводились в режиме 3.

Листинг 16.68. Использование второго параметра в функции `WEEK()`

```
mysql> SELECT WEEK('2005-04-24',1), WEEK('2005-04-24',0),
    -> DAYNAME('2005-04-24');
+--------------------+--------------------+----------------------+
| WEEK('2005-04-24',1) | WEEK('2005-04-24',0) | DAYNAME('2005-04-24') |
+--------------------+--------------------+----------------------+
|                 16 |                 17 | Sunday                |
+--------------------+--------------------+----------------------+
```

16.48. Функция *WEEKDAY()*

Функция `WEEKDAY(date)` принимает дату *date* в полном `YYYY-MM-DD hh:mm:ss` или кратком `YYYY-MM-DD` форматах (листинг 16.69) и возвращает номер дня недели (0 для понедельника, 1 для вторника, ..., 6 для воскресенья).

> ***Замечание***
>
> Функция `WEEKDAY()` появилась в MySQL, начиная с версии 4.1.1.

Листинг 16.69. Использование функции `WEEKDAY()`

```
mysql> SELECT WEEKDAY('2006-07-20 22:23:00'),
    -> DAYNAME('2006-07-20 22:23:00');
+------------------------------+------------------------------+
| WEEKDAY('2006-07-20 22:23:00') | DAYNAME('2006-07-20 22:23:00') |
+------------------------------+------------------------------+
|                            3 | Thursday                     |
+------------------------------+------------------------------+
mysql> SELECT WEEKDAY('2006-11-04'), DAYNAME('2006-11-04');
+----------------------+----------------------+
| WEEKDAY('2006-11-04') | DAYNAME('2006-11-04') |
+----------------------+----------------------+
|                    5 | Saturday             |
+----------------------+----------------------+
```

16.49. Функция *WEEKOFYEAR()*

Функция `WEEKOFYEAR(datetime)` возвращает порядковый номер недели в году (от 1 до 53) для даты *datetime* (листинг 16.70). Порядок вычисления номера соответствует режиму 3 функции `WEEK(date,3)` из табл. 16.7.

Листинг 16.70. Использование функции `WEEKOFYEAR()`

```
mysql> SELECT WEEKOFYEAR('1998-02-20');
+-------------------------+
| WEEKOFYEAR('1998-02-20') |
+-------------------------+
|                       8 |
+-------------------------+
```

16.50. Функция *YEAR()*

Функция `YEAR(`*`datetime`*`)` возвращает год (от 1000 до 9999) для даты *`datetime`* (листинг 16.71).

Листинг 16.71. Использование функции `YEAR()`

```
mysql> SELECT YEAR('06-07-03'), YEAR('2006-07-03'),
    -> YEAR('06-07-03 10:23:15');
+-----------------+-------------------+--------------------------+
| YEAR('06-07-03') | YEAR('2006-07-03') | YEAR('06-07-03 10:23:15') |
+-----------------+-------------------+--------------------------+
|             2006 |              2006 |                     2006 |
+-----------------+-------------------+--------------------------+
```

16.51. Функция *YEARWEEK()*

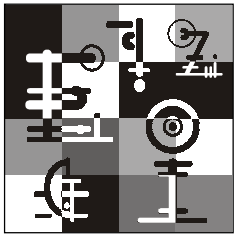
Функция `YEARWEEK(`*`date`*`)` возвращает число в формате `YYYYWW`, представляющее год и номер недели (от 0 до 53) в году и соответствующее дате *`date`* (листинг 16.72). Предполагается, что неделя начинается с воскресенья.

Листинг 16.72. Использование функции `YEARWEEK()`

```
mysql> SELECT YEARWEEK('2006-01-01');
+-----------------------+
| YEARWEEK('2006-01-01') |
+-----------------------+
|                200601 |
+-----------------------+
```

```
mysql> SELECT YEARWEEK('2006-01-01',0), YEARWEEK('2006-01-01',1);
+-------------------------+-------------------------+
| YEARWEEK('2006-01-01',0) | YEARWEEK('2006-01-01',1) |
+-------------------------+-------------------------+
|                  200601 |                  200552 |
+-------------------------+-------------------------+
```

Как видно из листинга 16.72, функция может принимать дополнительный аргумент, задающий день, с которого начинается неделя. Передача в качестве такого аргумента значения 0 задает в качестве первого дня воскресенье, 1 — понедельник и т. д. Функция `YAERWEEK()` никогда не возвращает для недели значение 0, если же такое значение выпадает, то она возвращает предыдущий год и число 52 для недели.



Глава 17

Строковые функции

Данная группа функций позволяет осуществлять различные преобразования строк. Все строковые функции условно можно поделить на функции общего назначения, которые будут рассмотрены в данной главе, и поисковые функции, описываемые в *главах 19* и *20*.

> ***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]` конфигурационного файла my.ini директиву `sql-mode=IGNORE-SPACE`.

При использовании строковых функций следует помнить, что позиция в строке начинается с 1, а не с 0, как, например, в C-подобных языках программирования.

17.1. Функция *ASCII()*

Функция `ASCII(str)` возвращает значение ASCII-кода первого символа строки *`str`* (листинг 17.1). Для пустой строки возвращается значение 0 или `NULL`, если в качестве *`str`* передано значение `NULL`.

Листинг 17.1. Использование функции `ASCII()`

```
mysql> SELECT ASCII('2'), ASCII(2), ASCII(0), ASCII(NULL);
+------------+----------+----------+-------------+
| ASCII('2') | ASCII(2) | ASCII(0) | ASCII(NULL) |
+------------+----------+----------+-------------+
|         50 |       50 |       48 |        NULL |
+------------+----------+----------+-------------+
```

Возвращаемые функцией символы лежат в диапазоне от 0 до 256.

17.2. Функция *BIN()*

Функция `BIN(N)` принимает десятичное число *N* и преобразует его двоичное представление (листинг 17.2).

> ***Замечание***
>
> Вызов функции `BIN(N)` эквивалентен вызову функции `CONV(N,10,2)`.

Листинг 17.2. Использование функции `BIN()`

```
mysql> SELECT BIN(12), BIN(95);
+---------+---------+
| BIN(12) | BIN(95) |
+---------+---------+
| 1100    | 1011111 |
+---------+---------+
```

17.3. Функция *BIT_LENGTH()*

Функция `BIT_LENGTH(str)` принимает строку *str* и возвращает ее длину в битах (листинг 17.3).

Листинг 17.3. Использование функции `BIT_LENGTH()`

```
mysql> SELECT BIT_LENGTH('text'), BIT_LENGTH(1);
+--------------------+---------------+
| BIT_LENGTH('text') | BIT_LENGTH(1) |
+--------------------+---------------+
|                 32 |             8 |
+--------------------+---------------+
```

Обычно каждый символ представлен 8 битами (1 байт), но в ряде кодировок под символ отводится более одного байта. Узнать максимальное число битов, отводимое под символ, можно при помощи SQL-запроса `SHOW CHARACTER SET` (см. листинг 12.4).

17.4. Функция *CHAR()*

Функция `CHAR(N1,N2,...)` принимает последовательность из ASCII-кодов и возвращает строку, созданную путем объединения соответствующих им символов (листинг 17.4).

Листинг 17.4. Использование функции CHAR()

```
mysql> SELECT CHAR(77,121,83,81,76);
+----------------------+
| CHAR(77,121,83,81,76) |
+----------------------+
| MySQL                |
+----------------------+
```

Замечание

Аргументы со значениями NULL игнорируются.

Начиная с версии MySQL 5.0.15, если код, передаваемый в качестве одного из аргументов функции CHAR() больше 255, строка автоматически конвертируется в многобайтовую строку. Пример использования кодов, больших 255, приводится в листинге 17.5.

Листинг 17.5. Формирование многобайтовых строк

```
mysql> SELECT HEX(CHAR(1,0)), HEX(CHAR(256));
+----------------+----------------+
| HEX(CHAR(1,0)) | HEX(CHAR(256)) |
+----------------+----------------+
| 0100           | 0100           |
+----------------+----------------+
mysql> SELECT HEX(CHAR(1,0,0)), HEX(CHAR(256*256));
+------------------+--------------------+
| HEX(CHAR(1,0,0)) | HEX(CHAR(256*256)) |
+------------------+--------------------+
| 010000           | 010000             |
+------------------+--------------------+
```

Однако для представления полученной строки не используется какая-либо многобайтовая кодировка вроде UTF8 или UCS2, и строка считается по умолчанию обычной бинарной строкой. Если необходимо привести результат к какой-либо кодировке, ее требуется указать явно при помощи конструкции USING (листинг 17.6).

Листинг 17.6. Приведение результата к кодировке UTF8

```
mysql> SELECT CHARSET(CHAR(0x65)), CHARSET(CHAR(0x65 USING utf8));
+--------------------+------------------------------+
| CHARSET(CHAR(0x65)) | CHARSET(CHAR(0x65 USING utf8)) |
+--------------------+------------------------------+
| binary             | utf8                         |
+--------------------+------------------------------+
```

17.5. Функция *CHAR_LENGTH()*

Функция `CHAR_LENGTH(str)` возвращает количество символов в строке *str*. В отличие от функции `LENGTH()` расчет ведется не по байтам, а по символам, поэтому функция корректно работает с многобайтовыми кодировками (листинг 17.7).

Замечание

Для функции `CHAR_LENGTH()` имеется синоним `CHARACTER_LENGTH()`.

Замечание

Функция `LENGTH()` рассматривается в *разд. 17.21*.

Листинг 17.7. Использование функции `CHAR_LENGTH()`

```
mysql> SELECT CHAR_LENGTH(CONVERT('MySQL' USING ucs2));
+-----------------------------------------+
| CHAR_LENGTH(CONVERT('MySQL' USING ucs2)) |
+-----------------------------------------+
|                                       5 |
+-----------------------------------------+
mysql> SELECT LENGTH(CONVERT('MySQL' USING ucs2));
+-----------------------------------+
| LENGTH(CONVERT('MySQL' USING ucs2)) |
+-----------------------------------+
|                                10 |
+-----------------------------------+
```

Для демонстрации работы функции выбрана кодировка UCS2, которая отводит ровно 2 байта под любой символ, в отличие от UTF8, которая отводит 1 байт под символы английского алфавита, 2 байта под символы европейских алфавитов и 3 байта под символы азиатских алфавитов.

Возвращаясь к учебной базе данных `shop`, подсчитаем число символов в полях `name` и `description` таблицы `products` (листинг 17.8).

Листинг 17.8. Подсчет числа символов в текстовом поле

```
mysql> SELECT name, CHAR_LENGTH(name) AS lname,
    -> CHAR_LENGTH(description) AS ldescription FROM products;
+-----------------------------+-------+--------------+
| name                        | lname | ldescription |
+-----------------------------+-------+--------------+
| Celeron 1.8                 |    11 |           59 |
```

```
| Celeron 2.0GHz                |    14 |           55 |
| Celeron 2.4GHz                |    14 |           55 |
| Celeron D 320 2.4GHz          |    20 |           60 |
| Celeron D 325 2.53GHz         |    21 |           62 |
| Celeron D 315 2.26GHz         |    21 |           62 |
| Intel Pentium 4 3.2GHz        |    22 |           78 |
| Intel Pentium 4 3.0GHz        |    22 |           80 |
| Intel Pentium 4 3.0GHz        |    22 |           77 |
| Gigabyte GA-8I848P-RS         |    21 |          117 |
| Gigabyte GA-8IG1000           |    19 |          124 |
| Gigabyte GA-8IPE1000G         |    21 |          139 |
| Asustek P4C800-E Delux        |    22 |          145 |
| Asustek P4P800-VM\L i865G     |    25 |          136 |
| Epox EP-4PDA3I                |    14 |          116 |
| ASUSTEK A9600XT/TD            |    18 |          107 |
| ASUSTEK V9520X                |    14 |           95 |
| SAPPHIRE 256MB RADEON 9550    |    26 |           56 |
| GIGABYTE AGP GV-N59X128D      |    24 |           49 |
| Maxtor 6Y120P0                |    14 |           56 |
| Maxtor 6B200P0                |    14 |           56 |
| Samsung SP0812C               |    15 |          133 |
| Seagate Barracuda ST3160023A  |    28 |           70 |
| Seagate ST3120026A            |    18 |           60 |
| DDR-400 256MB Kingston        |    22 |           42 |
| DDR-400 256MB Hynix Original  |    29 |           48 |
| DDR-400 256MB PQI             |    17 |           37 |
| DDR-400 512MB Kingston        |    22 |           42 |
| DDR-400 512MB PQI             |    17 |           37 |
| DDR-400 512MB Hynix           |    19 |           39 |
+-------------------------------+-------+--------------+
```

17.6. Функция *CHARSET()*

Функция `CHARSET(str)` возвращает имя кодировки, в которой представлена строка, передаваемая функции в качестве единственного аргумента (листинг 17.9).

Замечание

Функция `CHARSET()` введена в СУБД MySQL, начиная с версии 4.1.0.

Листинг 17.9. Использование функции `CHARSET()`

```
mysql> SELECT CHARSET('Программирование'),
    -> CHARSET(CONVERT('MySQL' USING utf8));
+----------------------------+-------------------------------------+
| CHARSET('Программирование') | CHARSET(CONVERT('MySQL' USING utf8)) |
+----------------------------+-------------------------------------+
| cp1251                     | utf8                                |
+----------------------------+-------------------------------------+
```

17.7. Функция *COLLATION()*

Функция `COLLATION(str)` возвращает порядок сортировки, установленный для кодировки аргумента *str* (листинг 17.10).

Замечание

Функция `COLLATION()` введена в MySQL, начиная с версии 4.1.0.

Листинг 17.10. Использование функции `COLLATION()`

```
mysql> SELECT COLLATION('Программирование'),
    -> COLLATION(CONVERT('MySQL' USING utf8));
+------------------------------+--------------------------------------+
| COLLATION('Программирование') | COLLATION(CONVERT('MySQL' USING utf8))|
+------------------------------+--------------------------------------+
| cp1251_general_ci            | utf8_general_ci                      |
+------------------------------+--------------------------------------+
```

17.8. Функция *COMPRESS()*

Функция `COMPRESS(string_to_compress)` сжимает строку *string_to_compress*. Пример представлен в листинге 17.11.

Замечание

Данная функция работает только в том случае, если MySQL скомпилирована совместно с библиотекой zlib, в противном случае всегда возвращается значение `NULL`.

Замечание

Функция `COMPRESS()` введена в MySQL, начиная с версии 4.1.1.

Листинг 17.11. Использование функции COMPRESS()

```
mysql> 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.

Следует отметить, что эффект сжатия заметен только для длинных текстовых строк с большим числом повторяющихся элементов.

Рассмотрим пример сжатия данных таблицы products в учебной базе данных shop (листинг 17.12).

Листинг 17.12. Сжатие описания товарных позиций в таблице products

```
mysql> select CHAR_LENGTH(description) AS ldescription,
    -> CHAR_LENGTH(COMPRESS(description)) AS compress from products;
+--------------+----------+
| ldescription | compress |
+--------------+----------+
|           59 |       70 |
|           55 |       65 |
|           55 |       66 |
|           60 |       72 |
|           62 |       74 |
|           62 |       74 |
|           78 |       90 |
|           80 |       92 |
|           77 |       89 |
|          117 |      123 |
```

```
|           124 |       134 |
|           139 |       149 |
|           145 |       154 |
|           136 |       143 |
|           116 |       127 |
|           107 |       115 |
|            95 |       107 |
|            56 |        69 |
|            49 |        62 |
|            56 |        68 |
|            56 |        66 |
|           133 |       136 |
|            70 |        82 |
|            60 |        71 |
|            42 |        57 |
|            48 |        63 |
|            37 |        52 |
|            42 |        57 |
|            37 |        52 |
|            39 |        54 |
+--------------+-----------+
```

Как видно из листинга 17.12, 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.

17.9. Функция *CONCAT()*

Функция `CONCAT(`*`str1`*`, `*`str2`*`, ...)` возвращает строку, созданную путем объединения всех аргументов, количество которых не ограничено (листинг 17.13). Если хотя бы один из аргументов равен `NULL`, возвращается значение `NULL`.

Листинг 17.13. Использование функции `CONCAT()`

```
mysql> SELECT CONCAT('ab'), CONCAT('ab','cd'), CONCAT('ab',NULL);
+--------------+-------------------+-------------------+
| CONCAT('ab') | CONCAT('ab','cd') | CONCAT('ab',NULL) |
+--------------+-------------------+-------------------+
| ab           | abcd              | NULL              |
+--------------+-------------------+-------------------+
```

```
mysql> SELECT CONCAT('Hello,',' ','world','!');
+---------------------------------+
| CONCAT('Hello,',' ','world','!') |
+---------------------------------+
| Hello, world!                   |
+---------------------------------+
```

Замечание

Числовые аргументы конвертируются в строки.

Если аргументы функции имеют различные кодировки, результирующая строка будет иметь кодировку первого аргумента.

Помимо функции `CONCAT()` возможно объединение строк при задании их одной за другой, как это показано в листинге 17.14.

Листинг 17.14. Альтернативный способ объединения строк

```
mysql> SELECT 'ab' 'cd' 'ef', 'ab' 'cd' = 'abcd';
+--------+--------------------+
| ab     | 'ab' 'cd' = 'abcd' |
+--------+--------------------+
| abcdef |                  1 |
+--------+--------------------+
```

Как видно из листинга 17.14, сравнение следующих друг за другом строк `'ab'` и `'cd'` с конечной строкой `'abcd'` приводит к значению 1 (истина).

17.10. Функция *CONCAT_WS()*

Функция `CONCAT_WS(`*separator*`, `*str1*`, `*str2*`, ...)`, как и предыдущая, объединяет аргументы *str1*, *str2* и т. д., помещая между ними разделитель *separator* (листинг 17.15).

В отличие от функции `CONCAT()` данная функция игнорирует параметры, равные `NULL`, за исключением *separator*, равенство которого `NULL` обращает результат в `NULL`.

Листинг 17.15. Использование функции `CONCAT_WS()`

```
mysql> SELECT CONCAT_WS(NULL,'ab','bc'), CONCAT_WS(',','ab',NULL,'bc');
+--------------------------+-----------------------------+
| CONCAT_WS(NULL,'ab','bc') | CONCAT_WS(',','ab',NULL,'bc') |
+--------------------------+-----------------------------+
| NULL                     | ab,bc                       |
+--------------------------+-----------------------------+
```

Если в качестве параметров *str1*, *str2* и т. д. передаются пустые строки, они также игнорируются.

17.11. Функция *CONV()*

Функция 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.

> ***Замечание***
>
> Функция CONV() возвращает NULL, если хотя бы один из аргументов равен NULL.

Листинг 17.16. Использование функции CONV()

```
mysql> SELECT CONV('E',16,2), CONV('E',16,10), CONV('E',16,8);
+---------------+----------------+---------------+
| CONV('E',16,2) | CONV('E',16,10) | CONV('E',16,8) |
+---------------+----------------+---------------+
| 1110           | 14             | 16            |
+---------------+----------------+---------------+
```

В листинге 17.16 шестнадцатеричное число E (14) переводится в двоичную, десятичную и восьмеричную системы счисления.

17.12. Функция *ELT()*

Функция 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 (листинг 17.17).

Листинг 17.17. Использование функции ELT()

```
mysql> SELECT ELT(3,'Hello','this','is','a','base');
+-------------------------------------+
| ELT(3,'Hello','this','is','a','base') |
+-------------------------------------+
| is                                  |
+-------------------------------------+
```

17.13. Функция *EXPORT_SET()*

Функция `EXPORT_SET(`*`bits`*`, `*`on`*`, `*`off`*`[, `*`separator`*`[, `*`number_of_bits`*`]])` возвращает число *bits* в двоичном представлении, 1 в котором заменяется на *on*, а 0 — на *off* (листинг 17.18). 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*).

Пример запроса приведен в листинге 17.18.

Листинг 17.18. Использование функции `EXPORT_SET()`

```
mysql> SELECT EXPORT_SET(7,'+','-','');
+-------------------------------------------------------------------+
| EXPORT_SET(7,'+','-','')                                          |
+-------------------------------------------------------------------+
| +++-------------------------------------------------------------- |
+-------------------------------------------------------------------+
mysql> SELECT EXPORT_SET(7,'+','-','',5);
+---------------------------+
| EXPORT_SET(7,'+','-','',5) |
+---------------------------+
| +++--                     |
+---------------------------+
```

Следует отметить, что в отличие от обычно двоичного числа символы располагаются слева направо, а не справа налево. Так в листинге 17.19 число 4, в двоичном представлении равное 100, представляется как 001.

Листинг 17.19. Порядок следования символов

```
mysql> SELECT EXPORT_SET(4,'1','0','');
+-------------------------------------------------------------------+
| EXPORT_SET(4,'1','0','')                                          |
+-------------------------------------------------------------------+
| 0010000000000000000000000000000000000000000000000000000000000000 |
+-------------------------------------------------------------------+
mysql> SELECT EXPORT_SET(4,'1','0','',3);
```

```
+---------------------------+
| EXPORT_SET(4,'1','0','',3) |
+---------------------------+
| 001                       |
+---------------------------+
```

Замечание

Функция `EXPORT_SET()` возвращает значение `NULL`, если один из аргументов равен `NULL`.

17.14. Функция *FIELD()*

Функция `FIELD(str, str1, str2, ...)` (листинг 17.20) находит строку *str* в списке строк *str1*, *str2*, ... и возвращает номер строки из этого списка (нумерация начинается с 1).

Листинг 17.20. Использование функции `FIELD()`

```
mysql> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', NULL, 'foo');
+-----------------------------------------------+
| FIELD('ej', 'Hej', 'ej', 'Heja', NULL, 'foo') |
+-----------------------------------------------+
|                                             2 |
+-----------------------------------------------+
mysql> SELECT FIELD(NULL, 'Hej', 'ej', 'Heja', NULL, 'foo');
+-----------------------------------------------+
| FIELD(NULL, 'Hej', 'ej', 'Heja', NULL, 'foo') |
+-----------------------------------------------+
|                                             0 |
+-----------------------------------------------+
```

Наличие в списке строк со значением `NULL` игнорируется, если же искомая строка *str* равна `NULL`, возвращается 0.

17.15. Функция *FIND_IN_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 17.21).

Замечание

Функция FIND_IN_SET() идеально подходит для определения, входит ли значение в строковый набор поля SET.

Листинг 17.21. Использование функции FIND_IN_SET()

```
mysql> SELECT FIND_IN_SET('show','hide,lock,show');
+-------------------------------------+
| FIND_IN_SET('show','hide,lock,show') |
+-------------------------------------+
|                                   3 |
+-------------------------------------+
mysql> SELECT FIND_IN_SET('active','hide,lock,show');
+--------------------------------------+
| FIND_IN_SET('active','hide,lock,show') |
+--------------------------------------+
|                                    0 |
+--------------------------------------+
```

Если один из аргументов функции FIND_IN_SET() равен NULL, функция возвращает NULL.

17.16. Функция *FORMAT()*

Функция FORMAT(*X*, *D*) преобразует число *X* в строку с *D* знаками после запятой (листинг 17.22). Каждые три разряда разделяются запятой.

Листинг 17.22. Использование функции FORMAT()

```
mysql> SELECT FORMAT(12345.12345,4), FORMAT(12345.1,4),
    -> FORMAT(12345.1,0);
+----------------------+------------------+------------------+
| FORMAT(12345.12345,4) | FORMAT(12345.1,4) | FORMAT(12345.1,0) |
+----------------------+------------------+------------------+
| 12,345.1234           | 12,345.1000       | 12,345            |
+----------------------+------------------+------------------+
```

Замечание

Если аргумент *D* имеет нулевое значение, возвращаемый результат не включает десятичную точку и дробную часть.

17.17. Функция *HEX()*

Функция `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 (листинг 17.23).

Листинг 17.23. Использование функции `HEX()` с числовым параметром

```
mysql> SELECT HEX(255), HEX(5678);
+----------+-----------+
| HEX(255) | HEX(5678) |
+----------+-----------+
| FF       | 162E      |
+----------+-----------+
```

Когда в качестве аргумента выступает строка, то функция переводит в шестнадцатеричное представление каждый символ строки и объединяет результат (листинг 17.24).

Листинг 17.24. Использование функции `HEX()` со строковым параметром

```
mysql> SELECT HEX('MySQL'), HEX('M'), HEX('y'), HEX('S'), HEX('Q'),
    -> HEX('L');
+--------------+----------+----------+----------+----------+----------+
| HEX('MySQL') | HEX('M') | HEX('y') | HEX('S') | HEX('Q') | HEX('L') |
+--------------+----------+----------+----------+----------+----------+
| 4D7953514C   | 4D       | 79       | 53       | 51       | 4C       |
+--------------+----------+----------+----------+----------+----------+
mysql> SELECT HEX('255'), HEX('5678');
+------------+-------------+
| HEX('255') | HEX('5678') |
+------------+-------------+
| 323535     | 35363738    |
+------------+-------------+
```

Замечание

Для преобразования чисел с использованием других систем счисления можно воспользоваться функцией `CONV()`.

17.18. Функция *INSERT()*

Функция `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 (листинг 17.25).

Листинг 17.25. Использование функции `INSERT()`

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
+----------------------------------+
| INSERT('Quadratic', 3, 4, 'What') |
+----------------------------------+
| QuWhattic                        |
+----------------------------------+
mysql> SELECT INSERT('Quadratic', -1, 4, 'What');
+-----------------------------------+
| INSERT('Quadratic', -1, 4, 'What') |
+-----------------------------------+
| Quadratic                         |
+-----------------------------------+
mysql> SELECT INSERT('Quadratic', 3, 100, 'What');
+------------------------------------+
| INSERT('Quadratic', 3, 100, 'What') |
+------------------------------------+
| QuWhat                             |
+------------------------------------+
```

17.19. Функция *INSTR()*

Функция `INSTR(`*str*`, `*substr*`)` возвращает позицию первого вхождения подстроки *substr* в строку *str* (листинг 17.26).

> ***Замечание***
>
> Функция `INSTR()` эквивалентна функции `LOCATE()` с двумя параметрами, за исключением того факта, что в `LOCATE(`*substr*`, `*str*`)` порядок параметров обратный.

Листинг 17.26. Использование функции `INSTR()`

```
mysql> SELECT INSTR('foobarbar', 'bar'), INSTR('xbar', 'foobar');
+--------------------------+------------------------+
| INSTR('foobarbar', 'bar') | INSTR('xbar', 'foobar') |
+--------------------------+------------------------+
|                         4 |                      0 |
+--------------------------+------------------------+
```

17.20. Функция *LEFT()*

Функция `LEFT(str, len)` возвращает `len` крайних левых символов строки `str` (листинг 17.27). Если аргумент `len` превышает число символов в строке `str`, она возвращается без изменений. Если `len` равно `NULL` или меньше 0, возвращается пустая строка.

Листинг 17.27. Использование функции `LEFT()`

```
mysql> SELECT LEFT('программирование', 5), LEFT('программирование', -1);
+----------------------------+-----------------------------+
| LEFT('программирование', 5) | LEFT('программирование', -1) |
+----------------------------+-----------------------------+
| прогр                      |                             |
+----------------------------+-----------------------------+
```

17.21. Функция *LENGTH()*

Функция `LENGTH(str)` (листинг 17.28) возвращает длину строки `str`. Данная функция может работать некорректно с многобайтными кодировками, т. к. фактически возвращается число байтов в строке.

> ***Замечание***
>
> Функция `CHAR_LENGTH()` рассматривается в *разд. 17.5*.

Листинг 17.28. Использование функции `LENGTH()`

```
mysql> SELECT LENGTH('MySQL');
+-----------------+
| LENGTH('MySQL') |
+-----------------+
|               5 |
+-----------------+
```

17.22. Функция *LOAD_FILE()*

Функция `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`. Создадим файл hello следующего содержания:

```
This is a text from file "hello"
```

Файл расположим в каталоге C:\mysql\bin\, тогда получить доступ к нему можно при помощи SQL-запроса, представленного в листинге 17.29.

Листинг 17.29. Использование функции LOAD_FILE()

```
mysql> SELECT LOAD_FILE("C:/mysql/bin/hello"), LOAD_FILE("hello");
+-----------------------------------+--------------------+
| LOAD_FILE("C:/mysql/bin/hello")   | LOAD_FILE("hello") |
+-----------------------------------+--------------------+
| This is a text from file "hello" | NULL               |
+-----------------------------------+--------------------+
```

Обычно данную функцию используют для заполнения BLOB-полей в операторах INSERT и UPDATE (листинг 17.30).

Листинг 17.30. Использование функции LOAD_FILE() при заполнении BLOB-полей

```
UPDATE tbl_name
SET blob_column = LOAD_FILE('C:/tmp/picture')
WHERE id=1;
```

17.23. Функция *LOCATE()*

Функция LOCATE(*substr*, *str*[, *pos*]) возвращает позицию первого вхождения подстроки *substr* в строку *str* (листинг 17.31). Если ни одного вхождения 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.

Листинг 17.31. Использование функции LOCATE()

```
mysql> SELECT LOCATE('bar','foobarbar'), LOCATE('xbar', 'foobar');
+---------------------------+---------------------------+
| LOCATE('bar','foobarbar') | LOCATE('xbar', 'foobar') |
+---------------------------+---------------------------+
|                         4 |                         0 |
+---------------------------+---------------------------+
mysql> SELECT LOCATE('bar','foobarbar',5);
+-----------------------------+
| LOCATE('bar','foobarbar',5) |
+-----------------------------+
|                           7 |
+-----------------------------+
```

17.24. Функция *LOWER()*

Функция `LOWER(str)` возвращает строку *str*, в которой все символы записаны в нижнем регистре (листинг 17.32).

Замечание

Для функции `LOWER()` имеется синоним — `LCASE()`.

Листинг 17.32. Использование функции `LOWER()`

```
mysql> SELECT LOWER('Hello, World!');
+-----------------------+
| LOWER('Hello, World!') |
+-----------------------+
| hello, world!          |
+-----------------------+
```

Замечание

Функция `LOWER()` корректно работает с многобайтными кодировками.

17.25. Функция *LPAD()*

Функция `LPAD(str, len, padstr)` (листинг 17.33) возвращает строку *str*, дополненную слева строкой *padstr* до длины *len*.

Листинг 17.33. Использование функции `LPAD()`

```
mysql> SELECT LPAD('111',10,'123456789'), LPAD('111',10,'57');
+---------------------------+---------------------+
| LPAD('111',10,'123456789') | LPAD('111',10,'57') |
+---------------------------+---------------------+
| 1234567111                | 5757575111          |
+---------------------------+---------------------+
```

Если строка *str* содержит более *len* символов, она усекается до длины *len* (листинг 17.34).

Листинг 17.34. Усечение строки при помощи `LPAD()`

```
mysql> SELECT LPAD('Hello, world!',10,'1');
+----------------------------+
| LPAD('Hello, world!',10,'1') |
+----------------------------+
| Hello, wor                 |
+----------------------------+
```

17.26. Функция *LTRIM()*

Функция `LTRIM(str)` возвращает строку *str*, в которой удалены все начальные пробелы (листинг 17.35).

Листинг 17.35. Использование функции `LTRIM()`

```
mysql> SELECT '  111  ' AS ST, LTRIM('  111  ') AS LT;
+---------+-------+
| ST      | LT    |
+---------+-------+
|   111   | 111   |
+---------+-------+
```

Замечание

Функция корректно работает с многобайтными кодировками.

17.27. Функция *MAKE_SET()*

Функция `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.

Пример запроса представлен в листинге 17.36.

Листинг 17.36. Использование функции `MAKE_SET()`

```
mysql> SELECT MAKE_SET(8,'a','b','c','d','e'),
    -> MAKE_SET(7,'a','b','c','d');
+--------------------------------+----------------------------+
| MAKE_SET(8,'a','b','c','d','e') | MAKE_SET(7,'a','b','c','d') |
+--------------------------------+----------------------------+
| d                              | a,b,c                      |
+--------------------------------+----------------------------+
```

Результат в листинге 17.36 легко понять, если вспомнить, что цифра 8 в двоичном представлении записывается как 1000, а 7 — как 0111.

17.28. Функция *MID()*

Функция `MID(str, pos[, len])` возвращает подстроку строки *str*, которая начинается с позиции *pos* и имеет длину *len* символов (листинг 17.37). Если третий параметр *len* не указывается, то подстрока возвращается, начиная с позиции *pos* и до конца строки *str*.

Замечание

Для функции `MID()` имеется синоним — `SUBSTRING()`.

Листинг 17.37. Использование функции `MID()`

```
mysql> SELECT MID('Hello, world! This is a database!',8,5);
+--------------------------------------------+
| MID('Hello, world! This is a database!',8,5) |
+--------------------------------------------+
| world                                      |
+--------------------------------------------+
mysql> SELECT MID('Hello, world! This is a database!',8);
+-----------------------------------------+
| MID('Hello, world! This is a database!',8) |
+-----------------------------------------+
| world! This is a database!              |
+-----------------------------------------+
```

Рассмотрим более сложный пример, связанный с учебной базой данных `shop`. Пусть стоит задача вывести первые буквы фамилий покупателей из таблицы `users`, причем в результирующий список каждая буква должна входить только один раз.

Листинг 17.38. Вывод первых букв фамилий покупателей из таблицы `users`

```
mysql> SELECT id_user, surname FROM users ORDER BY id_user;
+---------+-----------+
| id_user | surname   |
+---------+-----------+
|       1 | Иванов    |
|       2 | Лосев     |
|       3 | Симдянов  |
|       4 | Кузнецов  |
|       5 | Нехорошев |
|       6 | Корнеев   |
+---------+-----------+
mysql> SELECT MID(surname,1,1) AS first_char FROM users
    -> GROUP BY first_char ORDER BY first_char;
+------------+
| first_char |
+------------+
```

```
| И           |
| К           |
| Л           |
| Н           |
| С           |
+-------------+
```

Как видно из листинга 17.38, в таблице имеются две фамилии на букву К: Кузнецов и Корнеев, однако благодаря конструкции `GROUP BY` в результирующий список попадает только одна буква К. Для удобства ссылки на новый столбец при помощи ключевого слова `AS` результат функции переименовывается в столбец `first_char`.

17.29. Функция *OCT()*

Функция `OCT(N)` возвращает число *N* в восьмеричной системе счисления (листинг 17.39).

Листинг 17.39. Использование функции `OCT()`

```
mysql> SELECT OCT('255'), OCT(255), OCT(5678);
+-------------+----------+-----------+
| OCT('255') | OCT(255) | OCT(5678) |
+-------------+----------+-----------+
| 377         | 377      | 13056     |
+-------------+----------+-----------+
```

В отличие от функции `HEX()`, функция `OCT()` не делает различий между строковыми и числовыми типами параметра *N*.

> ***Замечание***
>
> Для преобразования чисел с использованием других систем счисления можно воспользоваться функцией `CONV()`, которая рассматривается в *разд. 17.11*.

17.30. Функция *ORD()*

Функция `ORD(str)` (листинг 17.40) возвращает ASCII-код крайнего левого символа строки *str*. Аналогична функции `ASCII()`, но в отличие от последней корректно работает с многобайтными кодировками.

Листинг 17.40. Использование функции ORD()

```
mysql> SELECT ORD('MySQL'),ORD('База');
+--------------+-------------+
| ORD('MySQL') | ORD('База') |
+--------------+-------------+
|           77 |         129 |
+--------------+-------------+
```

17.31. Функция *POSITION()*

Функция `POSITION(substr IN str)` полностью аналогична функции `LOCATE(substr, str)`.

17.32. Функция *QUOTE()*

Функция `QUOTE(str)` экранирует строку *str* с тем, чтобы получить корректное значение для SQL-выражения (листинг 17.41). Строка заключается в одинарные кавычки, и каждое вхождение одинарной кавычки (`'`), обратного слеша (`\`), значения ASCII NUL и символа от нажатия комбинации клавиш <Ctrl>+<Z> экранируются обратным слешем.

> ***Замечание***
>
> Если аргумент *str* равен `NULL`, то тогда результатом будет слово `NULL` без окружающих кавычек.

Листинг 17.41. Использование функции QUOTE()

```
mysql> SELECT QUOTE("Don't!");
+-----------------+
| QUOTE("Don't!") |
+-----------------+
| 'Don\'t!'       |
+-----------------+
```

17.33. Функция *REPEAT()*

Функция `REPEAT(str,count)` (листинг 17.42) возвращает строку, полученную из *count* повторений строки *str*. Если аргумент *count* имеет отрицательное значение или 0, возвращается пустая строка.

Листинг 17.42. Использование функции REPEAT()

```
mysql> SELECT REPEAT('MySQL ',3);
+--------------------+
| REPEAT('MySQL ',3) |
+--------------------+
| MySQL MySQL MySQL  |
+--------------------+
```

17.34. Функция *REPLACE()*

Функция `REPLACE(str, from_str, to_str)` (листинг 17.43) возвращает строку *str*, в которой все подстроки *from_str* заменены на *to_str*.

Листинг 17.43. Использование функции REPLACE()

```
mysql> SELECT REPLACE('MS SQL - лучшая база данных', 'MS SQL', 'MySQL');
+-----------------------------------------------------------+
| REPLACE('MS SQL - лучшая база данных', 'MS SQL', 'MySQL') |
+-----------------------------------------------------------+
| MySQL - лучшая база данных                                |
+-----------------------------------------------------------+
```

Замечание

Функция `REPLACE()` корректно работает с многобайтными кодировками.

Часто результат, возвращаемый одной функцией, может быть использован в качестве аргумента второй функции. В случае функции `REPLACE()` часто прибегают к каскадному выполнению этой функции, т. е. возвращаемый функцией результат передается в качестве параметра *str* следующей функции `REPLACE()`. В листинге 17.44 в одном запросе к таблице `products` учебной базы данных `shop` подстрока "Pentium 4" заменяется на "P IV", а "Celeron" на "Cel".

Листинг 17.44. Каскадное использование функции REPLACE()

```
mysql> SELECT id_product, REPLACE(name,'Pentium 4', 'P IV')
    -> FROM products ORDER BY id_product LIMIT 10;
+------------+---------------------------------+
| id_product | REPLACE(name,'Pentium 4', 'P IV') |
+------------+---------------------------------+
|          1 | Celeron 1.8                     |
|          2 | Celeron 2.0GHz                  |
```

```
|          3 | Celeron 2.4GHz                    |
|          4 | Celeron D 320 2.4GHz              |
|          5 | Celeron D 325 2.53GHz             |
|          6 | Celeron D 315 2.26GHz             |
|          7 | Intel P IV 3.2GHz                 |
|          8 | Intel P IV 3.0GHz                 |
|          9 | Intel P IV 3.0GHz                 |
|         10 | Gigabyte GA-8I848P-RS             |
+------------+-----------------------------------+
mysql> SELECT id_product,
    -> REPLACE(REPLACE(name,'Pentium 4','P IV'),'Celeron','Cel') AS res
    -> FROM products ORDER BY id_product LIMIT 10;
+------------+----------------------+
| id_product | res                  |
+------------+----------------------+
|          1 | Cel 1.8              |
|          2 | Cel 2.0GHz           |
|          3 | Cel 2.4GHz           |
|          4 | Cel D 320 2.4GHz     |
|          5 | Cel D 325 2.53GHz    |
|          6 | Cel D 315 2.26GHz    |
|          7 | Intel P IV 3.2GHz    |
|          8 | Intel P IV 3.0GHz    |
|          9 | Intel P IV 3.0GHz    |
|         10 | Gigabyte GA-8I848P-RS |
+------------+----------------------+
```

Как видно из листинга 17.44, обе замены удалось провести в рамках одного запроса. Важно помнить, что изменения касаются только результата и не затрагивают данных самой таблицы.

Функция `REPLACE()` может так же использоваться для удаления подстрок, с этой целью достаточно в качестве третьего параметра *`to_str`* передать пустую строку. Так, в листинге 17.45 удаляются все пробельные символы из результата запроса.

Листинг 17.45. Удаление пробельных символов

```
mysql> SELECT id_product, REPLACE(name,' ','') FROM products
    -> ORDER BY id_product LIMIT 10;
+------------+--------------------+
| id_product | REPLACE(name,' ','') |
+------------+--------------------+
|          1 | Celeron1.8         |
```

```
|           2 | Celeron2.0GHz        |
|           3 | Celeron2.4GHz        |
|           4 | CeleronD3202.4GHz    |
|           5 | CeleronD3252.53GHz   |
|           6 | CeleronD3152.26GHz   |
|           7 | IntelPentium43.2GHz  |
|           8 | IntelPentium43.0GHz  |
|           9 | IntelPentium43.0GHz  |
|          10 | GigabyteGA-8I848P-RS |
+-------------+----------------------+
```

17.35. Функция *REVERSE()*

Функция `REVERSE(str)` возвращает строку *str*, записанную в обратном порядке (листинг 17.46).

Листинг 17.46. Использование функции `REVERSE()`

```
mysql> SELECT REVERSE("Hello, I'm MySQL");
+----------------------------+
| REVERSE("Hello, I'm MySQL") |
+----------------------------+
| LQSyM m'I ,olleH           |
+----------------------------+
```

Замечание

Функция `REVERSE()` корректно работает с многобайтными кодировками.

17.36. Функция *RIGHT()*

Функция `RIGHT(str, len)` (листинг 17.47) 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`.

Листинг 17.47. Использование функции `RIGHT()`

```
mysql> SELECT RIGHT('MySQL',NULL),RIGHT('MySQL',0), RIGHT('MySQL',2);
+--------------------+------------------+------------------+
| RIGHT('MySQL',NULL) | RIGHT('MySQL',0) | RIGHT('MySQL',2) |
+--------------------+------------------+------------------+
|                    |                  | QL               |
+--------------------+------------------+------------------+
```

Замечание

Функция `RIGHT()` корректно работает с многобайтными кодировками.

17.37. Функция *RPAD()*

Функция `RPAD(str, len, padstr)` (листинг 17.48) возвращает строку *str*, дополненную справа строкой *padstr* до длины *len*.

Листинг 17.48. Использование функции `RPAD()`

```
mysql> SELECT RPAD('111',10,'23456789'), RPAD('111',10,'57');
+--------------------------+--------------------+
| RPAD('111',10,'23456789') | RPAD('111',10,'57') |
+--------------------------+--------------------+
| 1112345678                | 1115757575          |
+--------------------------+--------------------+
```

Если строка *str* содержит более *len* символов, она усекается до *len* (листинг 17.49).

Листинг 17.49. Отбрасывание лишних символов

```
mysql> SELECT RPAD('Hello, world!',10,'1');
+----------------------------+
| RPAD('Hello, world!',10,'1') |
+----------------------------+
| Hello, wor                  |
+----------------------------+
```

17.38. Функция *RTRIM()*

Функция `RTRIM(str)` возвращает строку *str*, в которой удалены все конечные пробелы (листинг 17.50).

Листинг 17.50. Использование функции `RTRIM()`

```
mysql> SELECT '  111  ' AS ST,RTRIM('  111  ') AS RT;
+---------+-------+
| ST      | RT    |
+---------+-------+
|   111   |   111 |
+---------+-------+
```

Замечание

Функция `RTRIM()` корректно работает с многобайтными кодировками.

17.39. Функция *SOUNDEX()*

Функция `SOUNDEX(str)` (листинг 17.51) возвращает произношение строки *str* по алгоритму, описанному Д. Кнутом. Все не алфавитные символы в строке игнорируются, а национальные символы рассматриваются как гласные.

Листинг 17.51. Использование функции `SOUNDEX()`

```
mysql> SELECT SOUNDEX('Hello'), SOUNDEX('Howl'), SOUNDEX('Cow'),
    -> SOUNDEX('Cowl');
+-----------------+----------------+---------------+----------------+
| SOUNDEX('Hello') | SOUNDEX('Howl') | SOUNDEX('Cow') | SOUNDEX('Cowl') |
+-----------------+----------------+---------------+----------------+
| H400             | H400           | C000          | C400           |
+-----------------+----------------+---------------+----------------+
```

17.40. Функция *SPACE()*

Функция `SPACE(N)` возвращает строку, состоящую из *N* пробелов, или пустую строку, если *N* имеет отрицательное значение (листинг 17.52).

Замечание

Если *N* принимает значение `NULL`, то функция также возвращает `NULL`.

Листинг 17.52. Использование функции `SPACE()`

```
mysql> SELECT SPACE(NULL), SPACE(0), SPACE(20);
+-------------+----------+----------------------+
| SPACE(NULL) | SPACE(0) | SPACE(20)            |
+-------------+----------+----------------------+
| NULL        |          |                      |
+-------------+----------+----------------------+
```

17.41. Функция *SUBSTRING()*

Функция `SUBSTRING(str, pos[, len])` возвращает подстроку строки *str*, начинающуюся с позиции *pos* и имеющую длину в *len* символов (листинг 17.53). Если третий параметр *len* не указывается, то подстрока возвращается, начиная с позиции *pos* и до конца строки *str*.

Замечание

Для функции `SUBSTRING()` имеется синоним — `MID()`.

Листинг 17.53. Использование функции `SUBSTRING()`

```
mysql> SELECT SUBSTRING('Hello, world! This is a database!',8,5);
+-------------------------------------------------+
| SUBSTRING('Hello, world! This is a database!',8,5) |
+-------------------------------------------------+
| world                                           |
+-------------------------------------------------+
mysql> SELECT SUBSTRING('Hello, world! This is a database!',8);
+-------------------------------------------------+
| SUBSTRING('Hello, world! This is a database!',7) |
+-------------------------------------------------+
| world! This is a database!                      |
+-------------------------------------------------+
```

Помимо указанного выше синтаксиса, в функции `SUBSTRING` можно использовать следующий синтаксис: `SUBSTRING(`*`str`* `FROM` *`pos`* `[FOR` *`len`*`])`, как показано в листинге 17.54.

Листинг 17.54. Альтернативный синтаксис функции `SUBSTRING()`

```
mysql> SELECT SUBSTRING('Hello, world! This is a database!' FROM 8 FOR
5);
+-----------------------------------------------------------+
| SUBSTRING('Hello, world! This is a database!' FROM 8 FOR 5) |
+-----------------------------------------------------------+
| world                                                     |
+-----------------------------------------------------------+
mysql> SELECT SUBSTRING('Hello, world! This is a database!' FROM 8);
+-------------------------------------------------+
| SUBSTRING('Hello, world! This is a database!' FROM 8) |
+-------------------------------------------------+
| world! This is a database!                      |
+-------------------------------------------------+
```

Замечание

Функция `SUBSTRING()` корректно работает с многобайтными кодировками.

17.42. Функция *SUBSTRING_INDEX()*

Функция `SUBSTRING_INDEX(str, delim, N)` возвращает подстроку строки `str` (листинг 17.55). Если параметр *N* имеет положительное значение, то функция `SUBSTRING_INDEX()` находит *N*-ое вхождение (отсчет слева) подстроки *delim* в строке *str* и возвращает всю часть строки, расположенную слева от подстроки *delim*. Если *N* имеет отрицательное значение, то находится *N*-ое вхождение (отсчет справа) подстроки *delim* в строке *str* и возвращается часть строки, расположенная справа от подстроки *delim*.

Листинг 17.55. Использование функции `SUBSTRING_INDEX()`

```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
+-----------------------------------------+
| SUBSTRING_INDEX('www.mysql.com', '.', 2) |
+-----------------------------------------+
| www.mysql                                |
+-----------------------------------------+
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
+------------------------------------------+
| SUBSTRING_INDEX('www.mysql.com', '.', -2) |
+------------------------------------------+
| mysql.com                                 |
+------------------------------------------+
```

Замечание

Функция `SUBSTRING_INDEX()` корректно работает с многобайтными кодировками.

17.43. Функция *TRIM()*

Функция `TRIM([[BOTH|LEADING|TRAILING] [remstr] FROM] str)` удаляет из строки *str* расположенные в начале (в конце) символы, указанные в строке *remstr* (листинг 17.56). 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 слово не задано, то по умолчанию устанавливается `BOTH`. Если строка *remstr* не задана, в качестве удаляемых символов выступают пробелы.

Листинг 17.56. Использование функции `TRIM()`

```
mysql> SELECT '  111  ' AS ST, TRIM('  111  ') AS BT;
+---------+-----+
| ST      | BT  |
+---------+-----+
```

```
|   111   | 111 |
+---------+-----+
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
+-----------------------------------+
| TRIM(LEADING 'x' FROM 'xxxbarxxx') |
+-----------------------------------+
| barxxx                            |
+-----------------------------------+
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
+-------------------------------+
| TRIM(BOTH 'x' FROM 'xxxbarxxx') |
+-------------------------------+
| bar                           |
+-------------------------------+
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz');
+-----------------------------------+
| TRIM(TRAILING 'xyz' FROM 'barxxyz') |
+-----------------------------------+
| barx                              |
+-----------------------------------+
```

Замечание

Функция `TRIM()` корректно работает с многобайтными кодировками.

17.44. Функция *UNCOMPRESS()*

Функция `UNCOMPRESS(`*`string_to_uncompress`*`)` (листинг 17.57) разархивирует строку *`string_to_uncompress`*, сжатую при помощи функции `COMPRESS()`. Если в качестве аргумента функции передается обычная, не сжатая строка, возвращается `NULL`.

Замечание

Данная функция работает только в том случае, если пакет MySQL скомпилирован с библиотекой zlib, в противном случае возвращается всегда `NULL`.

Замечание

Функция `UNCOMPRESS()` введена в MySQL, начиная с версии 4.1.1.

Листинг 17.57. Использование функции UNCOMPRESS()

```
mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
+----------------------------------+
| UNCOMPRESS(COMPRESS('any string')) |
+----------------------------------+
| any string                       |
+----------------------------------+
mysql> SELECT UNCOMPRESS('any string');
+--------------------------+
| UNCOMPRESS('any string') |
+--------------------------+
| NULL                     |
+--------------------------+
```

17.45. Функция *UNCOMPRESSED_LENGTH()*

Функция UNCOMPRESSED_LENGTH(*compressed_string*) возвращает длину строки *compressed_string* сжатой функцией COMPRESS() до применения данной функции (листинг 17.58). То есть при помощи функции UNCOMPRESSED_LENGTH() можно выяснить, какой будет длина строки после разархивирования ее функцией UNCOMPRESS().

> ***Замечание***
>
> Функция UNCOMPRESS() введена в MySQL, начиная с версии 4.1.1.

Листинг 17.58. Использование функции UNCOMPRESSED_LENGTH()

```
mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',1000)));
+-----------------------------------------------+
| UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',1000))) |
+-----------------------------------------------+
|                                          1000 |
+-----------------------------------------------+
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
+-----------------------------------+
| LENGTH(COMPRESS(REPEAT('a',1000))) |
+-----------------------------------+
|                                21 |
+-----------------------------------+
```

17.46. Функция *UNHEX()*

Функция `UNHEX(str)` является обратной функции `HEX()` и интерпретирует каждую пару символов строки *str* как шестнадцатеричный код, который необходимо преобразовать в символ (листинг 17.59).

> ***Замечание***
>
> Функция `UNCOMPRESS()` введена в MySQL, начиная с версии 4.1.2.

Листинг 17.59. Использование функции `UNHEX()`

```
mysql> SELECT UNHEX('4D7953514C'), 0x4D7953514C;
+--------------------+--------------+
| UNHEX('4D7953514C') | 0x4D7953514C |
+--------------------+--------------+
| MySQL               | MySQL        |
+--------------------+--------------+
mysql> SELECT UNHEX(HEX('string')), UNHEX(HEX('1234'));
+----------------------+--------------------+
| UNHEX(HEX('string')) | UNHEX(HEX('1234')) |
+----------------------+--------------------+
| string               | 1234               |
+----------------------+--------------------+
```

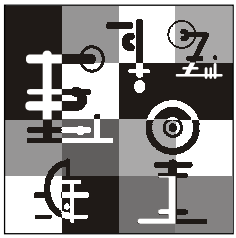
17.47. Функция *UPPER()*

Функция `UPPER(str)` переводит все символы строки *str* в верхний регистр (листинг 17.60). Если строка *str* имеет значение `NULL`, возвращается `NULL`.

> ***Замечание***
>
> Функция `UPPER()` имеет синоним — `UCASE()`.

Листинг 17.60. Использование функции `UPPER()`

```
mysql> SELECT UPPER('Hello, world!');
+-----------------------+
| UPPER('Hello, world!') |
+-----------------------+
| HELLO, WORLD!          |
+-----------------------+
```



Глава 18

Безопасность и MySQL

В данной главе рассматриваются многочисленные функции шифрования данных, которые входят в состав СУБД MySQL. Чаще всего шифрованию подвергаются пароли. В случае кражи базы данных злоумышленник получит доступ только к зашифрованным записям, на расшифровку которых потребуется время, за которое пароли перестанут быть актуальными.

Различают *необратимое шифрование*, когда подвергшиеся шифрованию данные невозможно расшифровать, и так называемое *симметричное шифрование*, когда данные шифруются с использованием секретного ключа, обладая которым можно произвести дешифровку данных. Как это на первый взгляд ни удивительно, но чаще прибегают именно к необратимому шифрованию данных. Введенный пользователем пароль также подвергается необратимому шифрованию, и между собой сравниваются зашифрованные строки — если они совпадают, пароль верен. Благодаря этой схеме никто, кроме пользователя, даже администратор базы данных, не имеет доступа к настоящему паролю.

18.1. Функции *AES_ENCRYPT()* и *AES_DECRYPT()*

Функции `AES_ENCRYPT()` и `AES_DECRYPT()` позволяют симметрично зашифровать и расшифровать данные, используя 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. В MySQL выбран 128-битовый ключ, поскольку он работает намного быстрее и обеспечивает вполне достаточную защиту. Функции имеют следующий синтаксис:

`AES_ENCRYPT(`*`str`*`, `*`key_str`*`)`

`AES_DECRYPT(`*`crypt_str`*`, `*`key_str`*`)`

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".

> ***Замечание***
>
> Функции `AES_ENCRYPT()` и `AES_DECRYPT()` были добавлены в версию 4.0.2 и могут рассматриваться как наиболее криптографически защищенные шифрующие функции, доступные в настоящее время в СУБД MySQL.

В листинге 18.1 приводится пример использования функций `AES_ENCRYPT()` и `AES_DECRYPT()`.

Листинг 18.1. Использование функций `AES_ENCRYPT()` и `AES_DECRYPT()`

```
mysql> SELECT AES_ENCRYPT('MySQL','секретный ключ');
+--------------------------------------+
| AES_ENCRYPT('MySQL','секретный ключ') |
+--------------------------------------+
| ...ЗАьZЭj9к•-еП™Mћ                    |
+--------------------------------------+
mysql> SELECT AES_DECRYPT('...ЗАьZЭj9к•-еП™Mћ','секретный ключ');
+-----------------------------------------------+
| AES_DECRYPT('...ЗАьZЭj9к•-еП™Mћ','секретный ключ') |
+-----------------------------------------------+
| MySQL                                         |
+-----------------------------------------------+
```

В качестве секретного ключа следует использовать фразу, которую сложно подобрать при помощи перебора или словаря. Желательно, чтобы фраза была длинной, запоминающейся, и в то же время состояла из несуществующих в словарях слов, например "бызэ даннух".

Рассмотрим шифрование поля `email` в таблице `users` учебной базы данных `shop`. Шифрование данного поля позволит уберечь пользователей от спамеров в случае, если база данных попадет им в руки. Так как поля таблицы уже созданы и существуют, воспользуемся для шифрования данных этих полей оператором `UPDATE` (листинг 18.2).

Листинг 18.2. Шифрование e-mail

```
mysql> SELECT email FROM users;
+----------------------+
| email                |
+----------------------+
| ivanov@email.ru      |
| losev@email.ru       |
| simdyanov@softtime.ru |
| kuznetsov@softtime.ru |
| NULL                 |
| korneev@domen.ru     |
+----------------------+
mysql> UPDATE users SET email = AES_ENCRYPT(email,'бызэ даннух');
mysql> SELECT email FROM users;
+----------------------------------+
| email                            |
+----------------------------------+
| □xтЫl—lЬ}‘.r®™Ц                  |
| №.LЙъўЕ□ѓ¦ФUIЪ"н                 |
| ђIgПуLA □2W□TI,,jeE:гУ@ддS
пСгц· |
| =-"рО26ецЩґу□jeE:гУ@ддS
пСгц· |
| NULL                             |
V1' |ђчўЧ_ПLЌЕЩ•БМ...ъtb©л"пж¬□я
+----------------------------------+
```

Как видно из листинга 18.2, прямой просмотр поля `email` приводит к выводу "мусора". Для того чтобы узнать настоящие адреса `email`, необходимо воспользоваться функцией дешифровки `AES_DECRYPT()` и знать секретный ключ (листинг 18.3).

Листинг 18.3. Дешифрация данных

```
mysql> SELECT AES_DECRYPT(email,'бызэ даннух') FROM users;
+----------------------------------+
| AES_DECRYPT(email,'бызэ даннух') |
+----------------------------------+
| ivanov@email.ru                  |
```

```
| losev@email.ru                  |
| simdyanov@softtime.ru           |
| kuznetsov@softtime.ru           |
| NULL                            |
| korneev@domen.ru                |
+---------------------------------+
```

18.2. Функции *ENCODE()* и *DECODE()*

Функции `ENCODE()` и `DECODE()` имеют синтаксис, аналогичный рассмотренным ранее функциям `AES_ENCRYPT()` и `AES_DECRYPT()`.

`ENCODE(`*str*`, `*pass_str*`)`

`DECODE(`*crypt_str*`, `*pass_str*`)`

Функция `ENCODE()` шифрует строку *str*, используя аргумент *pass_str* как секретный ключ. Для расшифровки результата следует использовать функцию `DECODE()`, которая принимает в качестве параметра *crypt_str* зашифрованную строку и возвращает исходную. Результирующая строка, которую возвращает функция `ENCODE()`, представляет собой двоичную строку той же длины, что и *str*. Для хранения результата в столбце следует использовать столбец типа `BLOB` — это не позволит повредить строку кодировкой. Пример использования функций `ENCODE()` и `DECODE()` приведен в листинге 18.4.

Листинг 18.4. Использование функций `ENCODE()` и `DECODE()`

```
mysql> SELECT DECODE(ENCODE('MySQL','бызэ даннух'),'бызэ даннух');
+-------------------------------------------------+
| DECODE(ENCODE('MySQL','бызэ даннух'),'бызэ даннух') |
+-------------------------------------------------+
| MySQL                                           |
+-------------------------------------------------+
```

18.3. Функции *DES_ENCRYPT()* и *DES_DECRYPT()*

Функции `DES_ENCRYPT()` и `DES_DECRYPT()` (листинг 18.5) осуществляют симметричное шифрование и дешифрование данных при помощи алгоритма 3DES (Triple-DES) и имеют следующий синтаксис:

`DES_ENCRYPT(`*string_to_encrypt*`[, (`*key_number* | *key_string*`) ])`

`DES_DECRYPT(`*string_to_decrypt*`[, `*key_string*`])`

Функция DES_ENCRYPT() в качестве первого параметра принимает строку *string_to_encrypt*, которая подвергается шифрованию. 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 указать при старте сервера MySQL в параметре --des-key-file. Важно отметить, что повторная передача номера при использовании функции DES_DECRYPT() 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.

> ***Замечание***
>
> Если в MySQL не включена поддержка протокола SSL, функции DES_ENCRYPT() и DES_DECRYPT() возвращают NULL.

Листинг 18.5. Использование функций DES_ENCRYPT() и DES_DECRYPT()

```
mysql> SELECT DES_DECRYPT(DES_ENCRYPT('MySQL'));
+----------------------------------+
| DES_DECRYPT(DES_ENCRYPT('MySQL')) |
+----------------------------------+
| MySQL                            |
+----------------------------------+
```

18.4. Функция *ENCRYPT()*

Функция ENCRYPT() подвергает данные необратимому шифрованию, используя вызов системной функции crypt() UNIX, и имеет следующий синтаксис:

```
ENCRYPT(str[, salt])
```

Строка *str* подвергается необратимому шифрованию, т. е. функции, расшифровывающей зашифрованные данные, уже нет. Если второй необязательный параметр *salt* не указывается, то результат каждый раз получается новым (листинг 18.6). Если в качестве второго параметра передается определенная строка, то при повторном вызове функции с таким же параметром результат будет воспроизводиться, если параметр не указан — результат каждый раз будет разным (листинг 18.7).

Листинг 18.6. Использование функции ENCRYPT()

```
mysql> SELECT ENCRYPT('MySQL'), ENCRYPT('MySQL');
+------------------+------------------+
| ENCRYPT('MySQL') | ENCRYPT('MySQL') |
+------------------+------------------+
| qnh1AW/pwGY2A    | snshG/AgkhKoU    |
+------------------+------------------+
```

Замечание

Если системный вызов `crypt()` отсутствует в системе, например, при использовании Windows в качестве операционной системы, функция `ENCRYPT()` всегда возвращает `NULL`.

Листинг 18.7. Использование функции ENCRYPT() совместно со вторым параметром

```
mysql> SELECT ENCRYPT('MySQL','password'), ENCRYPT('MySQL','password');
+----------------------------+----------------------------+
| ENCRYPT('MySQL','password') | ENCRYPT('MySQL','password') |
+----------------------------+----------------------------+
| pa6Fh2dhjP10c               | pa6Fh2dhjP10c               |
+----------------------------+----------------------------+
```

В листинге 18.7 так же, как и в листинге 18.6, необратимому шифрованию подвергается строка `'MySQL'`, только на этот раз в качестве параметра *`salt`* передается секретный ключ `'password'`. Как видно из листинга, это приводит к тому, что результат, возвращаемый функцией `ENCRYPT()`, воспроизводится при повторном вызове функции.

18.5. Функция *MD5()*

Функция `MD5()` (листинг 18.8) 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 18.8. Использование функции `MD5()`

```
mysql> SELECT MD5('MySQL'), MD5('MySQL');
+----------------------------------+----------------------------------+
| MD5('MySQL')                     | MD5('MySQL')                     |
+----------------------------------+----------------------------------+
| 62a004b95946bb97541afa471dcca73a | 62a004b95946bb97541afa471dcca73a |
+----------------------------------+----------------------------------+
mysql> SELECT MD5('MySQL1'), MD5('MySQL');
+----------------------------------+----------------------------------+
| MD5('MySQL1')                    | MD5('MySQL')                     |
+----------------------------------+----------------------------------+
| fc3dd4ddcb132de1f9552818344e1b09 | 62a004b95946bb97541afa471dcca73a |
+----------------------------------+----------------------------------+
```

Замечание

Алгоритм MD5 также часто применяется для создания уникального хеш-кода объемных файлов, которые передаются по сети. Загрузив файл, всегда можно проверить его целостность, вычислив код по алгоритму MD5 и сравнив полученный результат с хеш-кодом, предоставляемым распространителем. Это позволяет отследить повреждения файла, вызванные передачей через сеть, а также предотвращает фальсификацию дистрибутива.

18.6. Функция *PASSWORD()*

Функция `PASSWORD()` (листинг 18.9) подвергает необратимому шифрованию данные *`str`* и имеет следующий синтаксис:

```
PASSWORD(str)
```

Именно эта функция используется для шифрования паролей MySQL в столбце `password` в таблице привилегий `user` системной базы данных `mysql`.

Листинг 18.9. Использование функции `PASSWORD()`

```
mysql> SELECT PASSWORD('MySQL');
+-------------------------------------------+
| PASSWORD('MySQL')                         |
+-------------------------------------------+
| *1799AB5202FE2E9958365F9B3ECBBF53657254C7 |
+-------------------------------------------+
```

До версии MySQL 4.1 использовался менее защищенный алгоритм шифрования, результат работы которого представлен в листинге 18.10.

Листинг 18.10. Результат работы функции `PASSWORD()` в старых версиях MySQL

```
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 18.11).

Листинг 18.11. Использование функции `OLD_PASSWORD()`

```
mysql> 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()`.

18.7. Функция *SHA1()*

Функция `SHA1()` (листинг 18.12) вычисляет 160-битовую контрольную сумму по алгоритму SHA1 (Secure Hash Algorithm) для строки *`str`* и имеет следующий синтаксис:

`SHA1(`*`str`*`)`

Возвращаемая величина представляет собой 40-разрядное шестнадцатеричное число или `NULL` (в том случае, если входной аргумент был равен `NULL`).

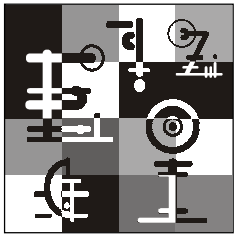
Листинг 18.12. Использование функции `SHA1()`

```
mysql> SELECT SHA('MySQL');
+------------------------------------------+
| SHA('MySQL')                             |
+------------------------------------------+
| deaa0c393a6613972aaccbf1fecfdad67aa21e88 |
+------------------------------------------+
```

Замечание

Для функции `SHA1()` имеется синоним `SHA()`.

Так же, как и в случае алгоритма MD5, результаты являются уникальными для каждой строки.



Глава 19

Поиск и регулярные выражения

Одной из сильных сторон баз данных по сравнению с регулярными и плоскими файлами, рассматриваемыми в качестве примитивных баз данных, являются широкие возможности поиска. СУБД MySQL предоставляет многочисленные инструменты для осуществления поиска, среди которых оператор `LIKE`, выполняющий простейшие операции поиска, оператор `RLIKE`, предоставляющий возможности поиска по регулярным выражениям, и механизм полнотекстового поиска, который подробно рассматривается в *главе 20*. В данной главе обсуждается поиск с использованием операторов `LIKE` и `RLIKE`.

19.1. Оператор *LIKE*

Оператор `LIKE` предназначен для сравнения строк с использованием простейших регулярных выражений и имеет следующий синтаксис:

expr LIKE *pat*

Оператор часто применяется в конструкции `WHERE` и возвращает 1 (истину), если выражение *expr* соответствует выражению *pat*, и 0 (ложь) в противном случае.

Пример запроса представлен в листинге 19.1.

Листинг 19.1. Использование оператора `LIKE`

```
mysql> SELECT 'MySQL' LIKE 'MySQL', 'MySQL' LIKE 'mysql';
+---------------------+---------------------+
| 'MySQL' LIKE 'MySQL' | 'MySQL' LIKE 'mysql' |
+---------------------+---------------------+
|                   1 |                   1 |
+---------------------+---------------------+
mysql> 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 19.1 операции аналогичны оператору =. Главное преимущество оператора `LIKE` перед оператором равенства заключается в использовании спецсимволов, приведенных в табл. 19.1.

Таблица 19.1. *Спецсимволы, используемые в операторе* `LIKE`

Символ	Описание
`%`	Соответствует любому количеству символов, даже их отсутствию
`_`	Соответствует ровно одному символу

При помощи спецсимволов, представленных в табл. 19.1, можно задать различные шаблоны соответствия (листинг 19.2).

Листинг 19.2. Использование спецсимвола `%`

```
mysql> SELECT 'Программист' LIKE 'Программ%';
+--------------------------------+
| 'Программист' LIKE 'Программ%' |
+--------------------------------+
|                              1 |
+--------------------------------+
mysql> SELECT 'Программа' LIKE 'Программ%';
+-----------------------------+
| 'Программа' LIKE 'Программ%' |
+-----------------------------+
|                           1 |
+-----------------------------+
mysql> SELECT 'Программ' LIKE 'Программ%';
```

```
+----------------------------+
| 'Программ' LIKE 'Программ%' |
+----------------------------+
|                          1 |
+----------------------------+
```

Как видно из листинга 19.2, спецсимвол `%` заменяет собой любую последовательность символов, в том числе и пустую строку, поэтому шаблон `Программ%` удовлетворяет словам "Программист", "Программа" и "Программ" и будет удовлетворять любому слову, начинающемуся с выражения `Программ`.

Спецсимвол `%` может быть размещен в любом месте шаблона, как в начале, так и в середине слова (листинг 19.3).

Листинг 19.3. Использование спецсимвола % в начале и середине шаблона

```
mysql> SELECT 'Программирование' LIKE 'П%е','Печенье' LIKE 'П%е';
+-----------------------------+---------------------+
| 'Программирование' LIKE 'П%е' | 'Печенье' LIKE 'П%е' |
+-----------------------------+---------------------+
|                           1 |                   1 |
+-----------------------------+---------------------+
mysql> SELECT 'Программирование' LIKE '%ние','Кодирование' LIKE '%ние';
+-------------------------------+--------------------------+
| 'Программирование' LIKE '%ние' | 'Кодирование' LIKE '%ние' |
+-------------------------------+--------------------------+
|                             1 |                        1 |
+-------------------------------+--------------------------+
```

Другой спецсимвол _ соответствует одному произвольному символу. Так, шаблон из трех знаков подчеркивания ___ будет соответствовать любому слову, состоящему из трех символов: код, рот, абв (листинг 19.4).

Листинг 19.4. Использование спецсимвола _

```
mysql> SELECT 'код' LIKE '___','рот' LIKE '___', 'абв' LIKE '___';
+-----------------+-----------------+-----------------+
| 'код' LIKE '___' | 'рот' LIKE '___' | 'абв' LIKE '___' |
+-----------------+-----------------+-----------------+
|               1 |               1 |               1 |
+-----------------+-----------------+-----------------+
```

Ограничений на позицию спецсимвола _, так же как и в случае %, нет: можно располагать его в любой части шаблона. Пример использования символа _ приведен в листингах 19.4 и 19.5.

Листинг 19.5. Использование спецсимвола _ в разных частях шаблона

```
mysql> 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 слеша, как это продемонстрировано в листинге 19.6.

Листинг 19.6. Поиск символов % и _

```
mysql> SELECT '15 %' LIKE '15 \%', 'my_sql' LIKE 'my\_sql';
+--------------------+-------------------------+
| '15 %' LIKE '15 \%' | 'my_sql' LIKE 'my\_sql' |
+--------------------+-------------------------+
|                  1 |                       1 |
+--------------------+-------------------------+
mysql> SELECT '15' LIKE '15 \%', 'my sql' LIKE 'my\_sql';
+------------------+------------------------+
| '15' LIKE '15 \%' | 'my sql' LIKE 'my\_sql' |
+------------------+------------------------+
|                0 |                      0 |
+------------------+------------------------+
```

В заключение данного раздела рассмотрим несколько примеров с обработки учебной базы данных shop. Первый запрос, представленный в листинге 19.7, извлекает из таблицы products записи, в которых имеется слово "Celeron".

Листинг 19.7. Извлечение из таблицы products записей, относящихся к Celeron

```
mysql> SELECT name FROM products WHERE name LIKE 'Celeron%';
+----------------------+
| name                 |
+----------------------+
```

```
| Celeron 1.8           |
| Celeron 2.0GHz        |
| Celeron 2.4GHz        |
| Celeron D 320 2.4GHz  |
| Celeron D 325 2.53GHz |
| Celeron D 315 2.26GHz |
+-----------------------+
```

Следующий запрос, который представлен в листинге 19.8, извлекает из таблицы `products` записи, соответствующие товарным позициям, цена которых находится между 1000 и 2000 рублей.

> ***Замечание***
>
> Для оперирования цифрами лучше использовать математические операторы, рассмотренные в *главе 15*, т. к. в этом случае действия с числами выполняются быстрее. Кроме того, СУБД не будет тратить время на преобразование типа.

Листинг 19.8. Извлечение из таблицы `products` записей

```
mysql> SELECT name, price FROM products
    -> WHERE price LIKE '1___.00'
    -> ORDER BY price;
+------------------------------+---------+
| name                         | price   |
+------------------------------+---------+
| DDR-400 256MB Kingston       | 1085.00 |
| DDR-400 256MB Hynix Original | 1179.00 |
| Celeron 1.8                  | 1595.00 |
| ASUSTEK V9520X               | 1602.00 |
| DDR-400 512MB PQI            | 1690.00 |
| DDR-400 512MB Hynix          | 1717.00 |
| Celeron D 315 2.26GHz        | 1880.00 |
| Gigabyte GA-8I848P-RS        | 1896.00 |
| DDR-400 512MB Kingston       | 1932.00 |
| Celeron D 320 2.4GHz         | 1962.00 |
| Celeron 2.0GHz               | 1969.00 |
+------------------------------+---------+
```

19.2. Оператор *NOT LIKE*

Оператор `NOT LIKE`, пример использования которого показан в листинге 19.9, 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. То есть с его помощью можно извлечь записи, которые не удовлетворяют указанному условию.

> ***Замечание***
>
> Оператор `NOT LIKE` можно представить в виде `NOT (`*expr* `LIKE` *pat*`)`.

Возвращаясь к учебной базе данных `shop`, извлечем из таблицы `products` записи, не вошедшие в список, который вернул запрос из листинга 19.8.

Листинг 19.9. Использование оператора `NOT LIKE`

```
mysql> SELECT name, price FROM products
    -> WHERE price NOT LIKE '1___.00'
    -> ORDER BY price;
+-----------------------------+---------+
| name                        | price   |
+-----------------------------+---------+
| DDR-400 256MB PQI           |  899.00 |
| Samsung SP0812C             | 2093.00 |
| Celeron 2.4GHz              | 2109.00 |
| Gigabyte GA-8IPE1000G       | 2289.00 |
| Epox EP-4PDA3I              | 2289.00 |
| Gigabyte GA-8IG1000         | 2420.00 |
| Maxtor 6Y120P0              | 2456.00 |
| Seagate ST3120026A          | 2468.00 |
| Asustek P4P800-VM\L i865G   | 2518.00 |
| SAPPHIRE 256MB RADEON 9550  | 2730.00 |
| Celeron D 325 2.53GHz       | 2747.00 |
| Seagate Barracuda ST3160023A| 3139.00 |
| Maxtor 6B200P0              | 3589.00 |
| ASUSTEK A9600XT/TD          | 5156.00 |
| Asustek P4C800-E Delux      | 5395.00 |
| Intel Pentium 4 3.0GHz      | 5673.00 |
| GIGABYTE AGP GV-N59X128D    | 5886.00 |
| Intel Pentium 4 3.0GHz      | 6147.00 |
| Intel Pentium 4 3.2GHz      | 7259.00 |
+-----------------------------+---------+
```

19.3. Оператор *SOUND LIKE*

Оператор `SOUND LIKE` аналогичен `SOUNDEX(`*`expr1`*`) = SOUNDEX(`*`expr2`*`)` и имеет следующий синтаксис:

`expr1` `SOUND LIKE` *`expr2`*

Оператор позволяет сравнивать слова *`expr1`* и *`expr2`* по звучанию и возвращает 1, если слова звучат одинаково, и 0 в противном случае.

> ***Замечание***
>
> Оператор `SOUND LIKE` появился в СУБД MySQL, начиная с версии 4.1.

19.4. Оператор *RLIKE (REGEXP)*

Оператор `RLIKE` (или его синоним `REGEXP`) позволяет производить поиск в соответствии с регулярными выражениями, которые предоставляют 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`.

Регулярные выражения — это специализированный язык поиска и осуществления манипуляций подстроками в тексте. Для работы с регулярными выражениями в СУБД MySQL предназначены конструкции `RLIKE` и `REGEXP`, которые являются синонимами и 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***
>
> 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.

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 такую подстроку ("грам"). Например, этому регулярному выражению будут соответствовать строки "программирование", "грамм", "грампластинка" и т. п.

Пример использования оператора `RLIKE` приведен в листинге 19.10.

Листинг 19.10. Использование оператора RLIKE

```
mysql> SELECT 'грамм' RLIKE 'грам', 'грампластинка' RLIKE 'грам';
+--------------------+----------------------------+
| 'грамм' RLIKE 'грам' | 'грампластинка' RLIKE 'грам' |
+--------------------+----------------------------+
|                  1 |                          1 |
+--------------------+----------------------------+
mysql> SELECT 'программирование' RLIKE 'грам';
+-------------------------------+
| 'программирование' RLIKE 'грам' |
+-------------------------------+
|                             1 |
+-------------------------------+
```

Как видно из листинга 19.10, 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 19.11).

Листинг 19.11. Использование спецсимвола ^

```
mysql> SELECT 'программирование' RLIKE '^грам',
    -> 'грампластинка' RLIKE '^грам';
+-------------------------------+----------------------------+
| 'программирование' RLIKE '^грам' | 'грампластинка' RLIKE '^грам' |
+-------------------------------+----------------------------+
|                             0 |                          1 |
+-------------------------------+----------------------------+
```

Спецсимвол `$` помогает привязать регулярное выражение к концу строки (листинг 19.12), т. е. применение символов ^ и `$` 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 19.12. Использование спецсимвола $

```
mysql> SELECT 'грампластинка' RLIKE '^грампластинка$';
+---------------------------------------+
| 'грампластинка' RLIKE '^грампластинка$' |
+---------------------------------------+
|                                     1 |
+---------------------------------------+
mysql> 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`, которые рассматриваются в *разд. "Конструкция IS NULL"* и *"Конструкция IS NOT NULL" главы 15*.

Возвращаясь к учебной базе данных `shop`, извлечем из таблицы `products` записи, поля `name` у которых заканчиваются подстрокой "HGz" (листинг 19.13).

Листинг 19.13. Извлечение всех записей, содержащих подстроку "HGz"

```
mysql> SELECT name FROM products WHERE name RLIKE 'GHz$';
+-----------------------+
| name                  |
+-----------------------+
| Celeron 2.0GHz        |
| Celeron 2.4GHz        |
| Celeron D 320 2.4GHz  |
| Celeron D 325 2.53GHz |
| Celeron D 315 2.26GHz |
| Intel Pentium 4 3.2GHz |
| Intel Pentium 4 3.0GHz |
| Intel Pentium 4 3.0GHz |
+-----------------------+
```

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 19.14).

Листинг 19.14. Использование последовательностей [[:<:]] и [[:>:]]

```
mysql> SELECT 'a word a' REGEXP '[[:<:]]word[[:>:]]';
+--------------------------------------+
| 'a word a' REGEXP '[[:<:]]word[[:>:]]' |
+--------------------------------------+
|                                    1 |
+--------------------------------------+
mysql> SELECT 'a xword a' REGEXP '[[:<:]]word[[:>:]]';
+---------------------------------------+
| 'a xword a' REGEXP '[[:<:]]word[[:>:]]' |
+---------------------------------------+
|                                     0 |
+---------------------------------------+
```

Символ вертикальной черты | (листинг 19.15) применяется в регулярном выражении для задания альтернативных масок:

```
'abc|абв'
```

Этому регулярному выражению соответствует любая строка, содержащая подстроки "abc" или "абв".

Листинг 19.15. Использование спецсимвола |

```
mysql> SELECT 'abc' RLIKE 'abc|абв', 'абв' RLIKE 'abc|абв';
+----------------------+----------------------+
| 'abc' RLIKE 'abc|абв' | 'абв' RLIKE 'abc|абв' |
+----------------------+----------------------+
|                    1 |                    1 |
+----------------------+----------------------+
```

Спецсимвол | 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 слеша \\ — в этом случае они теряют свое специальное значение и рассматриваются как обычные символы.

Возвращаясь к нашей учебной базе данных `shop`, можно привести следующий пример. Пусть необходимо извлечь из таблицы `products` записи, поля `name` которых содержат в своем составе подстроки "Pentium" или "Celeron". В этом случае запрос может выглядеть так, как это представлено в листинге 19.16.

Листинг 19.16. Извлечение всех записей, содержащих подстроки "Pentium" и "Celeron"

```
mysql> SELECT name FROM products WHERE name RLIKE 'Pentium|Celeron';
+------------------------+
| name                   |
+------------------------+
| Celeron 1.8            |
| Celeron 2.0GHz         |
| Celeron 2.4GHz         |
| Celeron D 320 2.4GHz   |
| Celeron D 325 2.53GHz  |
| Celeron D 315 2.26GHz  |
| Intel Pentium 4 3.2GHz |
| Intel Pentium 4 3.0GHz |
| Intel Pentium 4 3.0GHz |
+------------------------+
```

Выражение `'Intel Pentium|Celeron'` обеспечивает поиск строк, в состав которых входит подстрока `'Intel Pentium'` или `'Celeron'`. Для того чтобы производился поиск строк `'Intel Pentium'` или `'Intel Celeron'`, необходимо прибегнуть к круглым скобкам:

```
'Intel (Pentium|Celeron)'
```

Для задания класса символов используются квадратные скобки (листинг 19.17). Они ограничивают поиск теми символами, которые в них заключены, например, `[abc]`.

Листинг 19.17. Использование класса символов

```
mysql> 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-яё]`, иначе буква `ё` не попадет в диапазон.

> ***Замечание***
>
> Строки в СУБД MySQL не зависят от регистра, в том числе и в регулярных выражениях. Для того чтобы СУБД MySQL интерпретировала строки как регистро-зависимые, их необходимо предварять ключевым словом `BINARY` *(см. главу 14)*.

Точно таким же образом задается регулярное выражение, соответствующее любому числу:

```
'[0-9]'
```

Это выражение эквивалентно

```
'[0123456789]'
```

В листинге 19.18 приводится пример сравнения русской буквы Л и английской z с регулярным выражением `[а-я]`.

Листинг 19.18. Использование диапазонов в классах символов

```
mysql> SELECT 'Л' RLIKE '[а-яё]' AS a;
+---+
| a |
+---+
| 1 |
+---+
mysql> SELECT 'z' RLIKE '[а-яё]' AS a;
+---+
| a |
+---+
| 0 |
+---+
```

Выражение вида

```
'[а-яё0-9 ]'
```

соответствует либо букве русского алфавита, либо цифре, либо пробелу.

Если в начале класса помещается символ ^, то смысл выражения инвертируется (листинг 19.19). Так, выражение

```
'[^0-9]'
```

соответствует любому символу, кроме цифры.

Листинг 19.19. Отрицание в классах символов

```
mysql> 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 слеша `\`.

Кроме классов, которые могут создать разработчики, путем комбинирования отдельных символов и их диапазонов в стандарте POSIX предусмотрены специальные конструкции классов, представленные в табл. 19.2.

***Таблица 19.2.** 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)

***Таблица 19.2** (окончание)*

Класс	Описание
`[: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. 19.2 определяют диапазоны символов, так, например, регулярное выражение `[0-9]` эквивалентно `[[:digit:]]`. В листинге 19.20 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 19.20. Использование классов символов

```
mysql> SELECT '1' RLIKE '[[:digit:]]', 'a' RLIKE '[[:digit:]]';
+------------------------+------------------------+
| '1' RLIKE '[[:digit:]]' | 'a' RLIKE '[[:digit:]]' |
+------------------------+------------------------+
|                      1 |                      0 |
+------------------------+------------------------+
mysql> SELECT '1' RLIKE '[[:alpha:]]', 'a' RLIKE '[[:alpha:]]';
+------------------------+------------------------+
| '1' RLIKE '[[:alpha:]]' | 'a' RLIKE '[[:alpha:]]' |
+------------------------+------------------------+
|                      0 |                      1 |
+------------------------+------------------------+
mysql> 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 19.21)

```
'^СУБД|СУБД MySQL$'
```

до

```
'^СУБД( MySQL)?$'
```

Листинг 19.21. Использование квантификатора ?

```
mysql> 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 19.22 демонстрируется использование квантификатора `+`, который удовлетворяет последовательности из одного и большего числа символов.

Листинг 19.22. Использование квантификатора +

```
mysql> SELECT '1' RLIKE '^[[:digit:]]+$',
    -> '453455234' RLIKE '^[[:digit:]]+$';
+--------------------------+-----------------------------------+
| '1' RLIKE '^[[:digit:]]+$' | '453455234' RLIKE '^[[:digit:]]+$' |
+--------------------------+-----------------------------------+
|                        1 |                                 1 |
+--------------------------+-----------------------------------+
mysql> 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 19.23)

```
'^[[:digit:]]*$'
```

соответствует либо пустой строке, либо строке, содержащей только цифры, причем их количество не ограничено.

Листинг 19.23. Использование квантификатора *

```
mysql> SELECT '1' RLIKE '^[[:digit:]]*$',
    -> '453455234' RLIKE '^[[:digit:]]*$';
+--------------------------+-----------------------------------+
| '1' RLIKE '^[[:digit:]]*$' | '453455234' RLIKE '^[[:digit:]]*$' |
+--------------------------+-----------------------------------+
|                         1 |                                 1 |
+--------------------------+-----------------------------------+
mysql> 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 19.24 демонстрируется регулярное выражение, удовлетворяющее числу формата `###.##`. Целая часть может состоять из любого числа цифр, а дробная часть всегда состоит из двух.

Листинг 19.24. Поиск чисел формата ###.##

```
mysql> SELECT '123.90' RLIKE '^[[:digit:]]*\\.[[:digit:]]{2}$';
+-------------------------------------------------+
| '123.90' RLIKE '^[[:digit:]]*\\.[[:digit:]]{2}$' |
+-------------------------------------------------+
|                                               1 |
+-------------------------------------------------+
mysql> SELECT '123' RLIKE '^[[:digit:]]*\\.[[:digit:]]{2}$';
+----------------------------------------------+
| '123' RLIKE '^[[:digit:]]*\\.[[:digit:]]{2}$' |
+----------------------------------------------+
|                                            0 |
+----------------------------------------------+
```

Как видно из листинга 19.24, 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 19.25.

Листинг 19.25. Поиск чисел в форматах ### и ###.##

```
mysql> SELECT '123' RLIKE '^[[:digit:]]*(\\.[[:digit:]]{2})?$';
+-------------------------------------------------+
| '123' RLIKE '^[[:digit:]]*(\\.[[:digit:]]{2})?$' |
+-------------------------------------------------+
|                                               1 |
+-------------------------------------------------+
mysql> SELECT '123.90' RLIKE '^[[:digit:]]*(\\.[[:digit:]]{2})?$';
+----------------------------------------------------+
| '123.90' RLIKE '^[[:digit:]]*(\\.[[:digit:]]{2})?$' |
+----------------------------------------------------+
|                                                  1 |
+----------------------------------------------------+
```

Как видно из листинга 19.25, 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.

Следующий пример, приведенный в листинге 19.26, демонстрирует поиск строк в таблице `users` учебной базы данных `shop`, которые содержат e-mail вида ***someone@somwhere.dom***.

Листинг 19.26. Поиск e-mail

```
mysql> SELECT email FROM users;
+----------------------+
| email                |
+----------------------+
| ivanov@email.ru      |
| losev@email.ru       |
| simdyanov@softtime.ru |
| kuznetsov@softtime.ru |
| NULL                 |
| korneev@domen.ru     |
+----------------------+
mysql> SELECT email FROM users
    -> WHERE email RLIKE '^[-0-9a-z_\\.]+@[-0-9a-z_\\.]+\\.[a-z]{2,6}$';
+----------------------+
| email                |
+----------------------+
| ivanov@email.ru      |
| losev@email.ru       |
| simdyanov@softtime.ru |
| kuznetsov@softtime.ru |
| korneev@domen.ru     |
+----------------------+
```

У адреса имеются две составляющие — имя пользователя и имя домена, которые разделены знаком `@`. В имени пользователя могут присутствовать буквы нижнего и верхнего регистров, цифры, знаки подчеркивания, минуса и точки. Для проверки разделителя между именем пользователя и именем домена в выражение требуется добавить `@`. Таким образом, регулярное выражение, проверяющее имя пользователя и наличие разделителя, имеет следующий вид:

```
'[-0-9a-z_\\.]+@[-0-9a-z_\\.]+"
```

Для проверки последнего доменного имени (.ru, .com) необходимо добавить следующее выражение:

```
'\\.[a-z]{2,3}'
```

Символ . в регулярных выражениях используется для обозначения любого символа, поэтому для поиска соответствия точке в регулярном выражении этот символ экранируется: \\..

Объединяя эти строки и привязывая начало и конец регулярного выражения к началу и концу строки при помощи спецсимволов ^ и $, получаем выражение, представленное в листинге 19.26.

Можно создавать достаточно сложные SQL-запросы, комбинируя несколько операторов RLIKE при помощи логических операторов AND и OR. Пусть стоит задача выбора процессора марки Celeron, стоящего около 1900 рублей. Решение этой задачи может выглядеть так, как это представлено в листинге 19.27.

Листинг 19.27. Выбор процессоров марки Celeron стоимостью около 1900 рублей

```
mysql> SELECT name, price FROM products
    -> WHERE name RLIKE '^Celeron' AND
    ->       price RLIKE '^19[[:digit:]]{2}\\.[[:digit:]]{2}';
+---------------------+---------+
| name                | price   |
+---------------------+---------+
| Celeron 2.0GHz      | 1969.00 |
| Celeron D 320 2.4GHz | 1962.00 |
+---------------------+---------+
```

19.5. Оператор *NOT RLIKE*

Оператор NOT RLIKE аналогичен по действию оператору RLIKE и имеет следующий синтаксис:

expr NOT RLIKE *pat*

Оператор возвращает 0, если выражение *expr* соответствует выражению *pat*, и 1 в противном случае. То есть с его помощью можно извлечь записи, которые не удовлетворяют заданному условию.

> ***Замечание***
>
> Оператор NOT RLIKE имеет синоним NOT REGEXP.

> ***Замечание***
>
> Оператор NOT RLIKE можно представить в виде NOT (*expr* RLIKE *pat*).

Возвращаясь к учебной базе данных shop, извлечем из таблицы products записи, за исключением товарных позиций фирм Asustek и Gigabyte (листинг 19.28).

Листинг 19.28. Использование оператора `NOT RLIKE`

```
mysql> SELECT name, price FROM products
    -> WHERE name NOT RLIKE '^Asustek' AND name NOT RLIKE '^Gigabyte';
+------------------------------+---------+
| name                         | price   |
+------------------------------+---------+
| Celeron 1.8                  | 1595.00 |
| Celeron 2.0GHz               | 1969.00 |
| Celeron 2.4GHz               | 2109.00 |
| Celeron D 320 2.4GHz         | 1962.00 |
| Celeron D 325 2.53GHz        | 2747.00 |
| Celeron D 315 2.26GHz        | 1880.00 |
| Intel Pentium 4 3.2GHz       | 7259.00 |
| Intel Pentium 4 3.0GHz       | 6147.00 |
| Intel Pentium 4 3.0GHz       | 5673.00 |
| Epox EP-4PDA3I               | 2289.00 |
| SAPPHIRE 256MB RADEON 9550   | 2730.00 |
| Maxtor 6Y120P0               | 2456.00 |
| Maxtor 6B200P0               | 3589.00 |
| Samsung SP0812C              | 2093.00 |
| Seagate Barracuda ST3160023A | 3139.00 |
| Seagate ST3120026A           | 2468.00 |
| DDR-400 256MB Kingston       | 1085.00 |
| DDR-400 256MB Hynix Original | 1179.00 |
| DDR-400 256MB PQI            |  899.00 |
| DDR-400 512MB Kingston       | 1932.00 |
| DDR-400 512MB PQI            | 1690.00 |
| DDR-400 512MB Hynix          | 1717.00 |
+------------------------------+---------+
```

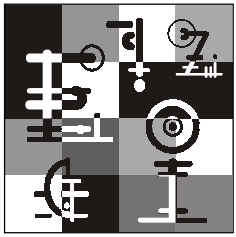
19.6. Функция *STRCMP()*

Функция `STRCMP()` предназначена для сравнения двух строк и имеет следующий синтаксис:

`STRCMP(`*expr1*`, `*expr2*`)`

Функция возвращает 0, если строки *expr1* и *expr2* идентичны, –1 — если *expr1* больше *expr2*, и 1 — если *expr1* меньше *expr2*.

> ***Замечание***
>
> Начиная с версии MySQL 4.0, для сравнения строк используется текущая кодировка, тогда как до версии 4.0 строки сравнивались как двоичные последовательности. То есть в текущих версиях сравнение не зависит от регистра, а в более ранних — зависит.



Глава 20

Полнотекстовый поиск

Помимо рассмотренного в предыдущей главе 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 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`.

20.1. 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`.

> ***Замечание***
>
> Синтаксис оператора `CREATE TABLE` подробно рассматривается в *главе 12*.

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 20.1 демонстрируется создание таблицы `catalogs` учебной базы данных `shop`. Столбец `name` в таблице `catalogs` 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 20.1. Индексирование столбца `name` индексом `FULLTEXT`

```
mysql> 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;
mysql> DESCRIBE catalogs;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| id_catalog | int(11)  | NO   | PRI | NULL    | auto_increment |
| name       | tinytext | NO   | MUL |         |                |
+------------+----------+------+-----+---------+----------------+
```

Индексом `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 20.2 приводится оператор `CREATE TABLE`, создающий таблицу `products` из учебной базы данных `shop`, в которой проиндексированы столбцы `name` и `description`.

Листинг 20.2. Индексирование нескольких столбцов

```
mysql> CREATE TABLE products (
    ->   id_product INT(11) NOT NULL AUTO_INCREMENT,
    ->   name TINYTEXT NOT NULL,
    ->   price DECIMAL(7,2) DEFAULT '0.00',
    ->   count INT(11) DEFAULT '0',
    ->   mark FLOAT(4,1) NOT NULL DEFAULT '0.0',
    ->   description TEXT,
    ->   id_catalog INT(11) NOT NULL DEFAULT '0',
    ->   PRIMARY KEY  (id_product),
    ->   INDEX id_catalog (id_catalog),
    ->   FULLTEXT INDEX search (name, description)
    -> ) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Имя индекса может быть любым, в листинге 20.2 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 20.3 приводятся операторы `ALTER TABLE` для создания индекса в рассмотренных выше таблицах `catalogs` и `products`.

> ***Замечание***
>
> Синтаксис оператора `ALTER TABLE` подробно рассматривается в *главе 13*.

Листинг 20.3. Создание индекса `FULLTEXT` при помощи `ALTER TABLE`

```
ALTER TABLE catalogs ADD FULLTEXT name (name);
ALTER TABLE products ADD FULLTEXT search (name, description);
```

В таблице не накладывается ограничений на число индексов, допустимо создание нескольких индексов, как показано в листинге 20.4.

Листинг 20.4. Создание нескольких индексов

```
ALTER TABLE products 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 20.5.

Листинг 20.5. Использование оператора `CREATE INDEX`

```
CREATE FULLTEXT INDEX name ON catalogs (name);
CREATE FULLTEXT INDEX search ON catalogs (name, description);
```

20.2. Конструкция *MATCH (...) AGAINST (...)*

Полнотекстовый поиск выполняется с помощью конструкции `MATCH (...) AGAINST (...)`, которая помещается в конструкцию `WHERE` оператора `SELECT`. В круглых скобках после ключевого слова `MATCH` указываются имена столбцов,

по которым производится поиск, а в скобках после AGAINST указывается 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. Так, если в таблице имеются только две записи, то поиск не даст результатов, т. к. каждое слово будет присутствовать в половине записей.

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 20.5) позволит производить поиск одновременно по этим двум столбцам (листинг 20.6).

Листинг 20.6. 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 (листинг 20.7).

Листинг 20.7. Полнотекстовый поиск по отдельным столбцам

```
SELECT * FROM products
         WEHRE MATCH (name) AGAINST ('Celeron');
SELECT * FROM products
         WEHRE MATCH (description) AGAINST ('Celeron');
```

Для осуществления данных запросов необходимо создать отдельные индексы по столбцу name и description, помимо уже существующего индекса по обоим столбцам.

В листинге 20.8 демонстрируется поиск слова "256MB" в таблице products учебной базы данных shop.

Листинг 20.8. Использование конструкции поиска

```
mysql> SELECT name FROM products
    -> WHERE MATCH (name, description) AGAINST ('256MB');
+-----------------------------+
| name                        |
```

```
+-----------------------------+
| DDR-400 256MB PQI           |
| DDR-400 256MB Kingston      |
| DDR-400 256MB Hynix Original |
| SAPPHIRE 256MB RADEON 9550  |
+-----------------------------+
```

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.

Таблица `products` учебной базы данных `shop` проиндексирована по двум столбцам: `name` и `description`. Каждое из полей содержит название процессора, а поле `description` содержит также слово "Процессор", поэтому вначале размещаются записи для процессоров Pentium и лишь затем Celeron (листинг 20.9).

Листинг 20.9. Использование конструкции `MATCH (...) AGAINST (...)`

```
mysql> SELECT name FROM products
    -> WHERE MATCH(name, description) AGAINST ('Процессор Pentium');
+-----------------------+
| name                  |
+-----------------------+
| Intel Pentium 4 3.0GHz |
| Intel Pentium 4 3.2GHz |
| Intel Pentium 4 3.0GHz |
| Celeron 1.8           |
| Celeron 2.0GHz        |
| Celeron 2.4GHz        |
| Celeron D 320 2.4GHz  |
| Celeron D 325 2.53GHz |
| Celeron D 315 2.26GHz |
+-----------------------+
```

Замечание

При полнотекстовом поиске можно использовать любые кодировки, в том числе и UTF8, однако кодировка UCS2 пока не поддерживается. Начиная с MySQL 4.1, полнотекстовый поиск корректно работает с национальными кодировками, но все столбцы, входящие в FULLTEXT-индекс, должны использовать одни и те же кодировку и сортировку.

Если поместить конструкцию `MATCH()` в список столбцов, следующих за ключевым словом `SELECT`, можно получить численные значения коэффициента релевантности (листинг 20.10). Это не приводит к дополнительной нагрузке, поскольку оптимизатор MySQL определяет, что два вызова `MATCH()` идентичны, и выполняет только один поиск.

Листинг 20.10. Численные значения коэффициента релевантности

```
mysql> SELECT name,
    ->        MATCH(name, description)
    ->        AGAINST ('Процессор Pentium') AS coefficient
    -> FROM products
    -> WHERE MATCH(name, description) AGAINST ('Процессор Pentium');
+-----------------------+------------------+
| name                  | coefficient      |
+-----------------------+------------------+
| Intel Pentium 4 3.0GHz |  3.3197972520964 |
| Intel Pentium 4 3.2GHz |  3.2590784068991 |
| Intel Pentium 4 3.0GHz |  3.2590784068991 |
| Celeron 1.8           | 0.70367693948263 |
| Celeron 2.0GHz        | 0.62730217952272 |
| Celeron 2.4GHz        | 0.62730217952272 |
| Celeron D 320 2.4GHz  | 0.62730217952272 |
| Celeron D 325 2.53GHz | 0.62730217952272 |
| Celeron D 315 2.26GHz | 0.62730217952272 |
+-----------------------+------------------+
```

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 20.11) 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 20.11. 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. главу 19)*.

Можно составлять достаточно сложные SQL-запросы, прибегая к комбинации нескольких конструкций `MATCH (...) AGAINST (...)` при помощи логических операторов `AND`, `OR` и `NOT`. В листинге 20.12 приводится пример запроса, в котором слова "Процессор" и "Pentium" ищутся при помощи двух отдельных конструкций `MATCH (...) AGAINST (...)`, объединенных логическим оператором `AND`.

Листинг 20.12. Использование логических операторов при полнотекстовом поиске

```
mysql> SELECT name FROM products
    -> WHERE MATCH (name, description) AGAINST ('Процессор') AND
    ->       MATCH (name, description) AGAINST ('Pentium');
+-----------------------+
| name                  |
+-----------------------+
| Intel Pentium 4 3.0GHz |
| Intel Pentium 4 3.2GHz |
| Intel Pentium 4 3.0GHz |
+-----------------------+
```

Как видно из листинга 20.12, возвращаются только те строки, которые содержат и слово "Процессор" (в данном случае оно входит в состав поля `description`, начи-

нающегося фразами "Процессор Celeron", "Процессор Pentium") и слово "Pentium". По сравнению с запросом, представленным в листинге 20.8, возвращаются только три записи.

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.

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, который более подробно рассматривается в *разд. 20.3*;
- `WITH QUERY EXPANSION` — поиск с расширением запроса, введенный начиная с версии 4.1.1. Более подробно этот режим поиска рассматривается в *разд. 20.4*;
- `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION` — данное ключевое слово является синонимом для `WITH QUERY EXPANSION` и введено, начиная с версии 5.1.7.

20.3. 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 20.13.

Листинг 20.13. Поиск в логическом режиме

```
mysql> SELECT name FROM products
    -> WHERE MATCH (name, description)
    -> AGAINST ('Процессор +Pentium' IN BOOLEAN MODE);
+-----------------------+
| name                  |
+-----------------------+
| Intel Pentium 4 3.2GHz |
```

```
| Intel Pentium 4 3.0GHz |
| Intel Pentium 4 3.0GHz |
+------------------------+
```

По сравнению с листингом 20.10, запрос в листинге 20.13 выдал только три записи, это связано с тем, что ключевому слову "Pentium" предшествует знак +, который требует, чтобы слово входило в каждую запись. Поэтому записи, содержащие в поле `description` только слово "Процессор", но не содержащие в индексированных столбцах слово "Pentium", 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 20.14 представлен запрос, который возвращает все записи, в состав которых входит слово "Процессор", но из выборки исключаются записи, содержащие слово "Intel".

Листинг 20.14. Исключение слов из выборки

```
mysql> SELECT name FROM products
    -> WHERE MATCH (name, description)
    -> AGAINST ('Процессор -Intel' IN BOOLEAN MODE);
+----------------------+
| name                 |
+----------------------+
| Celeron 1.8          |
| Celeron 2.0GHz       |
| Celeron 2.4GHz       |
| Celeron D 320 2.4GHz |
| Celeron D 325 2.53GHz |
| Celeron D 315 2.26GHz |
+----------------------+
```

Важно отметить, что запрос, состоящий из одних слов, которые предваряет минус, вернет пустую выборку (листинг 20.15).

Листинг 20.15. Использование символа минус –

```
mysql> SELECT name FROM products
    -> WHERE MATCH (name, description)
    -> AGAINST ('-Процессор -Intel' 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 20.16 использовать ключевое слово `IN BOOLEAN MODE` не обязательно.

Листинг 20.16. Естественный режим поиска

```
mysql> SELECT description FROM products
    -> WHERE MATCH (name, description)
    ->       AGAINST ('Celeron Pentium' IN BOOLEAN MODE);
+--------------------------------------------------------------+
| description                                                  |
+--------------------------------------------------------------+
| Процессор Celeron® 1.8GHz, 128kb, 478-PGA, 400Mhz, OEM 0.18  |
| Процессор Celeron® 2.0GHz, 128KB, 478-PGA, 400MHz, OEM       |
| Процессор Celeron® 2.4GHz, 128kb, 478-PGA, 400Mhz, OEM       |
| Процессор Celeron® D 320 2.4GHz, 256kb, 478-PGA, 533Mhz, OEM |
| Процессор Celeron® D 325 2.53GHz, 256kb, 478-PGA, 533Mhz, OEM|
| Процессор Celeron® D 315 2.26GHz, 256kb, 478-PGA, 533Mhz, OEM|
| Процессор Intel® Pentium®4 3.2GHz, 1Mb, 478-PGA, 800Mhz, HT, OEM |
| Процессор Intel® Pentium®4 3.0GHz, 512Kb, 478-PGA, 800Mhz, HT, OEM |
| Процессор Intel® Pentium®4 3.0GHz, 1Mb, 478-PGA, 800Mhz, HT, OEM |
+--------------------------------------------------------------+
```

В листинге 20.16 выборка содержит записи, в которых встречается как слово "Celeron", так и слово "Pentium".

Помимо + и –, в логическом режиме предусмотрено еще несколько специальных символов, описание которых приведено в табл. 20.1.

Таблица 20.1. *Специальные символы логического режима*

Символ	Описание
+	Предшествующий слову знак + показывает, что это слово должно присутствовать в каждой возвращенной строке
–	Предшествующий слову знак – означает, 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

Таблица 20.1 *(окончание)*

Символ	Описание
~	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

В листинге 20.17 демонстрируется применение специальных символов > и < для управления коэффициентом релевантности.

Листинг 20.17. Управление коэффициентом релевантности

```
mysql> SELECT name FROM products
    -> WHERE MATCH (name, description)
    -> AGAINST ('Процессор (>Celeron <Pentium)' IN BOOLEAN MODE);
+-----------------------+
| name                  |
+-----------------------+
| Celeron 1.8           |
| Celeron 2.0GHz        |
| Celeron 2.4GHz        |
| Celeron D 320 2.4GHz  |
| Celeron D 325 2.53GHz |
| Celeron D 315 2.26GHz |
| Intel Pentium 4 3.2GHz|
| Intel Pentium 4 3.0GHz|
| Intel Pentium 4 3.0GHz|
+-----------------------+
```

SQL-запрос в листинге 20.17 ищет комбинацию слов "Процессор" и "Celeron" или "Процессор" и "Pentium", но ранг "Процессор Celeron" выше, чем "Процессор Pentium".

Выведем числовые значения коэффициента релевантности для данного запроса (листинг 20.18).

Листинг 20.18. Числовые значения коэффициента релевантности

```
mysql> SELECT name,
    ->        MATCH (name, description)
    ->        AGAINST ('Процессор (>Celeron <Pentium)' IN BOOLEAN MODE)
    ->        AS coefficient
    -> FROM products
    -> WHERE MATCH (name, description)
    ->       AGAINST ('Процессор (>Celeron <Pentium)' IN BOOLEAN MODE);
+-----------------------+-----------------+
| name                  | coefficient     |
+-----------------------+-----------------+
| Celeron 1.8           |             2.5 |
| Celeron 2.0GHz        |             2.5 |
| Celeron 2.4GHz        |             2.5 |
| Celeron D 320 2.4GHz  |             2.5 |
| Celeron D 325 2.53GHz |             2.5 |
| Celeron D 315 2.26GHz |             2.5 |
| Intel Pentium 4 3.2GHz | 1.6666667461395 |
| Intel Pentium 4 3.0GHz | 1.6666667461395 |
| Intel Pentium 4 3.0GHz | 1.6666667461395 |
+-----------------------+-----------------+
```

Однако без использования логического режима и символов управления коэффициентом релевантности порядок сортировки прямо противоположный: сначала следуют записи, соответствующие процессору Pentium и лишь затем — процессору Celeron (листинг 20.19).

Листинг 20.19. Сортировка записей в полнотекстовом режиме

```
mysql> SELECT name,
    ->        MATCH (name, description)
    ->        AGAINST ('Процессор Celeron Pentium') AS coefficient
    -> FROM products
    -> WHERE MATCH (name, description)
    -> AGAINST ('Процессор Celeron Pentium');
+-----------------------+-----------------+
| name                  | coefficient     |
+-----------------------+-----------------+
| Intel Pentium 4 3.0GHz | 3.3197972520964 |
```

```
| Intel Pentium 4 3.2GHz | 3.2590784068991 |
| Intel Pentium 4 3.0GHz | 3.2590784068991 |
| Celeron 1.8            | 2.6530160544406 |
| Celeron 2.0GHz         | 2.3650664294898 |
| Celeron 2.4GHz         | 2.3650664294898 |
| Celeron D 320 2.4GHz   | 2.3650664294898 |
| Celeron D 325 2.53GHz  | 2.3650664294898 |
| Celeron D 315 2.26GHz  | 2.3650664294898 |
+------------------------+-----------------+
```

Важно отметить, что при использовании логического режима без управляющих символов > и <, коэффициент релевантности для всех строк одинаковый и составляет величину 2 (листинг 20.20). Это связано с тем, что по умолчанию сортировка по коэффициенту релевантности в логическом режиме не производится, поэтому коэффициент для всех найденных записей одинаков.

Листинг 20.20. Числовые значения коэффициента релевантности

```
mysql> SELECT name,
    ->        MATCH (name, description)
    ->        AGAINST ('Процессор Celeron Pentium' IN BOOLEAN MODE)
    ->        AS coefficient
    -> FROM products
    -> WHERE MATCH (name, description)
    -> AGAINST ('Процессор Celeron Pentium' IN BOOLEAN MODE);
+------------------------+-------------+
| name                   | coefficient |
+------------------------+-------------+
| Celeron 1.8            |           2 |
| Celeron 2.0GHz         |           2 |
| Celeron 2.4GHz         |           2 |
| Celeron D 320 2.4GHz   |           2 |
| Celeron D 325 2.53GHz  |           2 |
| Celeron D 315 2.26GHz  |           2 |
| Intel Pentium 4 3.2GHz |           2 |
| Intel Pentium 4 3.0GHz |           2 |
| Intel Pentium 4 3.0GHz |           2 |
+------------------------+-------------+
```

В отличие от предыдущих спецсимволов, символ * всегда располагается в конце слова и указывает на то, что после слова может следовать любое количество других символов, составляющих слово. Например, последовательности `файл*` будут соответствовать слова "файл", "файловый", "файлообмен" и т. п.

> ***Замечание***
>
> Однако символ * не может использоваться для поиска слов, длина которых меньше четырех символов, если только не изменено значение системной переменной `ft_min_word_len` *(см. разд. 20.2)*.

Использование двойных кавычек (") требует поиска неразрывной фразы. В листинге 20.21 производится поиск фразы "Процессор Celeron".

Листинг 20.21. Поиск неразрывной фразы

```
mysql> SELECT description FROM products
    -> WHERE MATCH (name, description)
    -> AGAINST ('"Процессор Celeron"' IN BOOLEAN MODE);
+------------------------------------------------------------+
| description                                                |
+------------------------------------------------------------+
| Процессор Celeron® 1.8GHz, 128kb, 478-PGA, 400Mhz, OEM 0.18    |
| Процессор Celeron® 2.0GHz, 128KB, 478-PGA, 400MHz, OEM         |
| Процессор Celeron® 2.4GHz, 128kb, 478-PGA, 400Mhz, OEM         |
| Процессор Celeron® D 320 2.4GHz, 256kb, 478-PGA, 533Mhz, OEM   |
| Процессор Celeron® D 325 2.53GHz, 256kb, 478-PGA, 533Mhz, OEM  |
| Процессор Celeron® D 315 2.26GHz, 256kb, 478-PGA, 533Mhz, OEM  |
+------------------------------------------------------------+
```

Несмотря на то, что записи, соответствующие процессору Pentium, содержат слово "Процессор", они в выборку не попадают, т. к. в них отсутствует фраза "`Процессор Celeron`". Более того, поиск фразы "`Процессор Pentium`" с использованием двойных кавычек вернет пустую результирующую таблицу (листинг 20.22).

Листинг 20.22. Использование двойных кавычек

```
mysql> SELECT description FROM products
    -> WHERE MATCH (name, description)
    -> AGAINST ('"Процессор Pentium"' IN BOOLEAN MODE);
Empty set (0.00 sec)
```

К верному результату приведет лишь фраза '`Процессор Intel Pentium`', которая встречается в поле `description` соответствующего процессора (листинг 20.23).

Замечание

Важно отметить, что графический символ ®, не относящийся к числовым и алфавитным символам, воспринимается как символ-разделитель и не учитывается при полнотекстовом поиске.

Листинг 20.23. Поиск полной фразы

```
mysql> SELECT description FROM products
    -> WHERE MATCH (name, description)
    -> AGAINST ('"Процессор Intel Pentium"' IN BOOLEAN MODE);
+-------------------------------------------------------------------+
| description                                                       |
+-------------------------------------------------------------------+
| Процессор Intel® Pentium®4 3.2GHz, 1Mb, 478-PGA, 800Mhz, HT, OEM  |
| Процессор Intel® Pentium®4 3.0GHz, 512Kb, 478-PGA, 800Mhz, HT, OEM |
| Процессор Intel® Pentium®4 3.0GHz, 1Mb, 478-PGA, 800Mhz, HT, OEM  |
+-------------------------------------------------------------------+
```

20.4. Режим расширения запроса

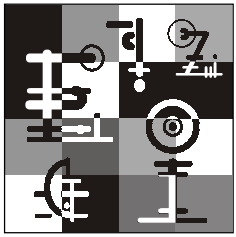
Режим расширения запроса применяется, как правило, когда поисковая фраза подразумевает более широкую трактовку запроса. Например, пользователь, выполняя поиск по слову "database", может иметь в виду, что записи, в состав которых входят фразы "MySQL", "Oracle", "DB2" и "MSSQL", тоже должны быть включены 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 20.24.

Листинг 20.24. Поиск в режиме расширения запроса

```
mysql> SELECT name FROM products
    ->             WHERE MATCH (name, description)
    ->             AGAINST ('Celeron');
+----------------------+
| name                 |
+----------------------+
| Celeron 1.8          |
| Celeron 2.0GHz       |
| Celeron 2.4GHz       |
| Celeron D 320 2.4GHz |
```

```
| Celeron D 325 2.53GHz |
| Celeron D 315 2.26GHz |
+-----------------------+
mysql> SELECT name FROM products
    ->             WHERE MATCH (name, description)
    ->             AGAINST ('Celeron' WITH QUERY EXPANSION);
+------------------------+
| name                   |
+------------------------+
| Celeron D 320 2.4GHz   |
| Celeron 2.4GHz         |
| Celeron D 325 2.53GHz  |
| Celeron D 315 2.26GHz  |
| Celeron 1.8            |
| Celeron 2.0GHz         |
| Intel Pentium 4 3.0GHz |
| Intel Pentium 4 3.0GHz |
| Intel Pentium 4 3.2GHz |
| Asustek P4C800-E Delux |
| Gigabyte GA-8IPE1000G  |
| ASUSTEK V9520X         |
+------------------------+
```

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 20.24, помимо процессора Celeron, найдены также процессоры Pentium, в записях которых имеется слово "Процессор". 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. Именно благодаря этому в результаты поиска попали также и видеокарты, которые были найдены по слову "400MHz".



Глава 21

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` *(см. главу 7)*. 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.

21.1. Функция *AVG()*

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 21.1 приводится запрос к таблице `products` учебной базы данных `shop`, который возвращает среднюю цену товарной позиции в магазине.

Листинг 21.1. Использование функции AVG()

```
mysql> SELECT AVG(price) FROM products;
+-------------+
| AVG(price)  |
+-------------+
| 2858.966667 |
+-------------+
```

При использовании конструкции GROUP BY функция AVG() вычисляет среднее значение для каждой из группы. Пусть группировка в таблице products происходит по полю id_catalog, тогда при помощи встроенной функции AVG() можно узнать среднюю цену по каждому из пяти каталогов электронного магазина. В листинге 21.2 выводится результирующая таблица с уникальными номерами id_catalog и соответствующие каждому из каталогов средние цены.

Листинг 21.2. Использование функции AVG() совместно с GROUP BY

```
mysql> SELECT id_catalog, AVG(price) FROM products GROUP BY id_catalog;
+------------+-------------+
| id_catalog | AVG(price)  |
+------------+-------------+
|          1 | 3482.333333 |
|          2 | 2801.166667 |
|          3 | 3843.500000 |
|          4 | 2749.000000 |
|          5 | 1417.000000 |
+------------+-------------+
```

Средние значения, полученные при помощи функции AVG(), могут использоваться в вычисляемых столбцах. Например, для того чтобы увеличить средние значения для каждого из каталогов на 20%, достаточно умножить либо столбец price, либо функцию AVG() на 1.2 (листинг 21.3).

Листинг 21.3. Использование функции AVG() в выражениях

```
mysql> SELECT id_catalog, AVG(price*1.2) FROM products
    -> GROUP BY id_catalog;
+------------+----------------+
| id_catalog | AVG(price*1.2) |
```

```
+------------+----------------+
|          1 | 4178.8000000   |
|          2 | 3361.4000000   |
|          3 | 4612.2000000   |
|          4 | 3298.8000000   |
|          5 | 1700.4000000   |
+------------+----------------+
5 rows in set (0.03 sec)
mysql> SELECT id_catalog, AVG(price)*1.2 FROM products
    -> GROUP BY id_catalog;
+------------+----------------+
| id_catalog | AVG(price)*1.2 |
+------------+----------------+
|          1 | 4178.8000000   |
|          2 | 3361.4000000   |
|          3 | 4612.2000000   |
|          4 | 3298.8000000   |
|          5 | 1700.4000000   |
+------------+----------------+
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 сначала осуществляется группировка значений и вычисление средних значений, и лишь после этого результат умножается на `1.2`. Как следствие, первый запрос выполняется 0,03 секунды, а второй 0,00 секунды.

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 21.4).

Листинг 21.4. Сортировка результирующей таблицы

```
mysql> SELECT id_catalog, AVG(price) AS price FROM products
    -> GROUP BY id_catalog ORDER BY price DESC;
+------------+-------------+
| id_catalog | price       |
+------------+-------------+
|          3 | 3843.500000 |
|          1 | 3482.333333 |
```

```
|           2 | 2801.166667 |
|           4 | 2749.000000 |
|           5 | 1417.000000 |
+-------------+-------------+
```

Если необходимо использовать в условии вычисляемые столбцы, следует указывать конструкцию `HAVING`, а не `WHERE` (листинг 21.5).

Листинг 21.5. Использование условия `HAVING`

```
mysql> SELECT id_catalog, AVG(price) AS price FROM products
    -> GROUP BY id_catalog HAVING price > 3000 ORDER BY price DESC;
+-------------+-------------+
| id_catalog | price        |
+-------------+-------------+
|           3 | 3843.500000 |
|           1 | 3482.333333 |
+-------------+-------------+
```

21.2. Функция *BIT_AND()*

Функция `BIT_AND()` возвращает побитовое И для всех битов в *`expr`* и имеет следующий синтаксис:

```
BIT_AND(expr)
```

Вычисление производится с 64-битовой (`BIGINT`) точностью (листинг 21.6).

Листинг 21.6. Использование функции `BIT_AND()`

```
mysql> SELECT BIT_AND(id_catalog) FROM catalogs;
+---------------------+
| BIT_AND(id_catalog) |
+---------------------+
|                   0 |
+---------------------+
```

Результат можно легко понять, если представить значения `id_catalog` в двоичном виде и обратиться к табл. 15.1, где приведены правила двоичного сложения разрядов при использовании логики И.

```
001 (1)
010 (2)
```

```
011 (3)
100 (4)
101 (5)
-------
000 (0)
```

При группировке по полю `id_catalog`, когда каждая запись образует собственную группу, функция возвращает исходные значения полей `id_catalog` (листинг 21.7).

Листинг 21.7. Использование функции `BIT_AND()` совместно с `GROUP BY`

```
mysql> SELECT BIT_AND(id_catalog) FROM catalogs GROUP BY id_catalog;
+--------------------+
| BIT_AND(id_catalog) |
+--------------------+
|                  1 |
|                  2 |
|                  3 |
|                  4 |
|                  5 |
+--------------------+
```

21.3. Функция *BIT_OR()*

Функция `BIT_OR()` (листинг 21.8) возвращает побитовое ИЛИ для всех битов в *`expr`* и имеет следующий синтаксис:

`BIT_OR(`*`expr`*`)`

Вычисление производится с 64-битовой (`BIGINT`) точностью.

Листинг 21.8. Использование функции `BIT_OR()`

```
mysql> SELECT BIT_OR(id_catalog) FROM catalogs;
+-------------------+
| BIT_OR(id_catalog) |
+-------------------+
|                 7 |
+-------------------+
```

Результат можно легко понять, если представить значения id_catalog в двоичном представлении и обратиться к табл. 15.2, где приведены правила двоичного сложения разрядов при использовании логики ИЛИ.

```
001 (1)
010 (2)
011 (3)
100 (4)
101 (5)
-------
111 (7)
```

Точно так же, как и в случае с функцией BIT_AND(), при группировке по полю id_catalog возвращаются исходные значения полей id_catalog, т. к. в этом случае аргумент один и сложения как такового не происходит.

21.4. Функция *BIT_XOR()*

Функция BIT_OR() (листинг 21.9) возвращает исключающее побитовое ИЛИ для всех битов в *expr* и имеет следующий синтаксис:

```
BIT_XOR(expr)
```

Вычисление производится с 64-битовой (BIGINT) точностью.

> ***Замечание***
>
> Функция BIT_XOR() введена в СУБД MySQL, начиная с версии 4.1.1.

Листинг 21.9. Использование функции BIT_XOR()

```
mysql> SELECT BIT_XOR(id_catalog) FROM catalogs;
+---------------------+
| BIT_XOR(id_catalog) |
+---------------------+
|                   1 |
+---------------------+
```

Результат можно легко понять, если представить значения id_catalog в двоичном представлении и обратиться к табл. 15.3, где приведены правила двоичного сложения разрядов при использовании логики исключающего ИЛИ.

```
001 (1)
010 (2)
011 (3)
```

```
100 (4)
101 (5)
-------
001 (7)
```

Точно так же, как и в случае с функцией `BIT_AND()`, при группировке по полю `id_catalog` возвращаются исходные значения полей `id_catalog`, т. к. в этом случае аргумент один и сложения как такового не происходит.

21.5. Функция *COUNT()*

Функция `COUNT()` подсчитывает число записей и имеет несколько форм, синтаксис которых представлен далее:

```
COUNT(expr)
COUNT(*)
COUNT(DISTINCT expr1, expr2, ...)
```

Первая форма возвращает число записей в таблице, поле *expr* для которых не равно `NULL`. В листинге 21.10 представлен запрос, возвращающий число непустых полей `id_user`, `email` и `url` таблицы `users` учебной базы данных `shop`.

Листинг 21.10. Использование функции `COUNT()`

```
mysql> SELECT id_user, email, url FROM users;
+---------+----------------------+-------------------------+
| id_user | email                | url                     |
+---------+----------------------+-------------------------+
|       1 | ivanov@email.ru      | NULL                    |
|       2 | losev@email.ru       | NULL                    |
|       3 | simdyanov@softtime.ru | http://www.softtime.ru/ |
|       4 | kuznetsov@softtime.ru | http://www.softtime.ru  |
|       5 | NULL                 | NULL                    |
|       6 | korneev@domen.ru     | NULL                    |
+---------+----------------------+-------------------------+
mysql> SELECT COUNT(id_user), COUNT(email), COUNT(url) FROM users;
+---------------+--------------+------------+
| COUNT(id_user) | COUNT(email) | COUNT(url) |
+---------------+--------------+------------+
|              6 |            5 |          2 |
+---------------+--------------+------------+
```

Как видно из листинга 21.10, для полей `id_user`, `email` и `url` возвращаются различные значения. Это связано с тем, что число `NULL`-полей во всех трех столбцах различается.

Форма функции `COUNT(*)` возвращает общее число строк в таблице, независимо от того, принимает какое-либо поле значение `NULL` или нет (листинг 21.11). Запись таблицы учитывается в результате, даже если все поля записи равны `NULL`.

Листинг 21.11. Использование функции `COUNT(*)`

```
mysql> SELECT COUNT(*) FROM users;
+----------+
| COUNT(*) |
+----------+
|        6 |
+----------+
```

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 21.12).

Листинг 21.12. Использование функции `COUNT()` совместно с `WHERE`

```
mysql> SELECT COUNT(*) FROM products WHERE price < 2000;
+----------+
| COUNT(*) |
+----------+
|       12 |
+----------+
```

В листинге 21.12 представлен запрос, извлекающий из таблицы `products` учебной базы `shop` записи, соответствующие товарным позициям, цена которых не превышает 2000 рублей.

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 21.13 представлен запрос, сообщающий о количестве товарных позиций в каждом из каталогов электронного магазина.

Листинг 21.13. Подсчет количества товарных позиций в каждом из каталогов

```
mysql> SELECT id_catalog, COUNT(*) AS total FROM products
    -> GROUP BY id_catalog ORDER BY total DESC;
+------------+-------+
| id_catalog | total |
+------------+-------+
|          1 |     9 |
|          5 |     6 |
|          2 |     6 |
|          4 |     5 |
|          3 |     4 |
+------------+-------+
```

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 21.14).

Листинг 21.14. Использование ключевого слова `DISTINCT` совместно с `COUNT()`

```
mysql> SELECT id_product FROM orders;
+------------+
| id_product |
+------------+
|          8 |
|         10 |
|         20 |
|         20 |
|         20 |
+------------+
mysql> SELECT COUNT(id_product), COUNT(DISTINCT id_product) FROM orders;
+-------------------+----------------------------+
| COUNT(id_product) | COUNT(DISTINCT id_product) |
+-------------------+----------------------------+
|                 5 |                          3 |
+-------------------+----------------------------+
```

Так как в поле id_product таблицы orders значение 20 встречается три раза, то функция COUNT(DISTINCT id_product) возвращает только три уникальных значения вместо пяти.

21.6. Функция *GROUP_CONCAT()*

Функция GROUP_CONCAT() объединяет значения отдельных групп, полученных в результате применения конструкции GROUP BY, в одну строку и 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 (листинг 21.15).

> ***Замечание***
>
> Функция появилась в MySQL, начиная с версии 4.1.0.

Листинг 21.15. Использование функции GROUP_CONCAT()

```
mysql> SELECT GROUP_CONCAT(id_catalog) FROM catalogs;
+--------------------------+
| GROUP_CONCAT(id_catalog) |
+--------------------------+
| 1,2,3,4,5                |
+--------------------------+
```

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 21.16 представлен запрос, извлекающий уникальные значения столбца id_product таблицы orders учебной базы данных shop с использованием в качестве разделителя символа -.

Листинг 21.16. Использование ключевых слов DISTINCT и SEPARATOR

```
mysql> SELECT GROUP_CONCAT(DISTINCT id_product SEPARATOR '-')
    -> FROM orders;
```

```
+-----------------------------------------------+
| GROUP_CONCAT(DISTINCT id_product SEPARATOR '-') |
+-----------------------------------------------+
| 8-10-20                                       |
+-----------------------------------------------+
```

Ключевое слово `ORDER BY` позволяет отсортировать значения в рамках возвращаемой строки. Отсортируем значения, представленные в листинге 21.16, в обратном порядке (листинг 21.17).

Листинг 21.17. Использование ключевого слова `ORDER BY`

```
mysql> SELECT GROUP_CONCAT(DISTINCT id_product
    ->                     ORDER BY id_product DESC
    ->                     SEPARATOR '-') AS str
    -> FROM orders;
+---------+
| str     |
+---------+
| 20-10-8 |
+---------+
```

Возвратим строки с ценами (в порядке убывания) на товарные позиции для каждого из каталогов учебного электронного магазина (листинг 21.18).

Листинг 21.18. Список цен товарных позиций в каталогах

```
mysql> SELECT GROUP_CONCAT(price ORDER BY id_product DESC)
    -> FROM products GROUP BY id_catalog;
+-------------------------------------------------------------------------+
| str                                                                     |
+-------------------------------------------------------------------------+
|5673.00,6147.00,7259.00,1880.00,2747.00,1962.00,2109.00,1969.00,1595.00|
|2289.00,2518.00,5395.00,2289.00,2420.00,1896.00                          |
|5886.00,2730.00,1602.00,5156.00                                          |
|2468.00,3139.00,2093.00,3589.00,2456.00                                  |
|1717.00,1690.00,1932.00,899.00,1179.00,1085.00                           |
+-------------------------------------------------------------------------+
```

21.7. Функция *MIN()*

Возвращает минимальное значение среди всех непустых значений выбранных строк в столбце *expr* и 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.

Пример запроса представлен в листинге 21.19.

Листинг 21.19. Использование функции MIN()

```
mysql> SELECT MIN(price) FROM products;
+------------+
| MIN(price) |
+------------+
| 899.00     |
+------------+
```

В листинге 21.19 запрос возвращает минимальную цену в таблице products. Использование конструкции GROUP BY id_catalog позволяет найти минимальную цену для каждого из элементов каталога (листинг 21.20).

Листинг 21.20. Использование функции MIN() совместно с конструкцией GROUP BY

```
mysql> SELECT id_catalog, MIN(price) FROM products GROUP BY id_catalog;
+------------+------------+
| id_catalog | MIN(price) |
+------------+------------+
|          1 | 1595.00    |
|          2 | 1896.00    |
|          3 | 1602.00    |
|          4 | 2093.00    |
|          5 | 899.00     |
+------------+------------+
```

Как видно из результатов запроса, товарная позиция с минимальной для всей таблицы ценой находится в пятом элементе каталога.

Функцию `MIN()` можно также использовать со строковыми столбцами. В этом случае возвращается минимальное лексикографическое значение (листинг 21.21).

Листинг 21.21. Использование функции `MIN()` совместно со строковыми столбцами

```
mysql> SELECT MIN(name) FROM products;
+--------------------+
| MIN(name)          |
+--------------------+
| ASUSTEK A9600XT/TD |
+--------------------+
```

21.8. Функция *MAX()*

Возвращает максимальное значение среди всех непустых значений выбранных строк в столбце *expr* и 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.

Пример запроса представлен в листинге 21.22.

Листинг 21.22. Использование функции `MAX()`

```
mysql> SELECT MAX(price) FROM products;
+------------+
| MAX(price) |
+------------+
| 7259.00    |
+------------+
```

В листинге 21.22 запрос возвращает максимальную цену в таблице `products`. Использование конструкции `GROUP BY id_catalog` позволяет найти максимальную цену для каждого из элементов каталога (листинг 21.23).

Листинг 21.23. Использование функции `MIN()` совместно с конструкцией `GROUP BY`

```
mysql> SELECT id_catalog, MAX(price) FROM products GROUP BY id_catalog;
+------------+------------+
| id_catalog | MAX(price) |
+------------+------------+
|          1 | 7259.00    |
|          2 | 5395.00    |
|          3 | 5886.00    |
|          4 | 3589.00    |
|          5 | 1932.00    |
+------------+------------+
```

21.9. Функция *STD()*

Функция `STD()` (листинг 21.24) возвращает стандартное среднеквадратичное отклонение значения в аргументе *expr* и имеет следующий синтаксис:

```
STD(expr)
```

> ***Замечание***
>
> Функция `STD()` имеет два синонима: первый — `STDDEV()` обеспечивает совместимость с СУБД Oracle, и второй — `STDDEV_POP()` обеспечивает совместимость со стандартом SQL.

> ***Замечание***
>
> Синоним `STDDEV_POP()` появился в СУБД MySQL, начиная с версии 5.0.3.

Листинг 21.24. Использование функции `STD()`

```
mysql> SELECT STD(price) FROM products;
+-------------+
| STD(price)  |
+-------------+
| 1653.531584 |
+-------------+
mysql> SELECT id_catalog, STD(price) FROM products GROUP BY id_catalog;
+------------+-------------+
| id_catalog | STD(price)  |
+------------+-------------+
|          1 | 2090.202808 |
```

```
|           2 | 1175.967604 |
|           3 | 1743.464009 |
|           4 |  539.044711 |
|           5 |  379.681006 |
+-------------+-------------+
```

Функция возвращает квадратный корень значения, которое возвращает функция `VAR_POP()`.

21.10. Функция *STDDEV_SAMP()*

Функция `STDDEV_SAMP()` (листинг 21.25) возвращает выборочное среднеквадратическое отклонение и имеет следующий синтаксис:

```
STDDEV_SAMP(expr)
```

> ***Замечание***
>
> Функция `STDDEV_SAMP()` появилась в СУБД MySQL, начиная с версии 5.0.3.

Листинг 21.25. Использование функции `STDDEV_SAMP()`

```
mysql> SELECT STDDEV_SAMP(price) FROM products;
+--------------------+
| STDDEV_SAMP(price) |
+--------------------+
|         1681.799129 |
+--------------------+
mysql> SELECT id_catalog, STDDEV_SAMP(price) FROM products
    -> GROUP BY id_catalog;
+------------+--------------------+
| id_catalog | STDDEV_SAMP(price) |
+------------+--------------------+
|          1 |        2216.994869 |
|          2 |        1288.207967 |
|          3 |        2013.178830 |
|          4 |         602.670308 |
|          5 |         415.919704 |
+------------+--------------------+
```

Функция возвращает квадратный корень значения, которое возвращается функцией `VAR_SAMP()`.

21.11. Функция *SUM()*

Функция `SUM()` (листинг 21.26) возвращает сумму величин в столбце *expr* и 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*.

> ***Замечание***
>
> Ключевое слово `DISTINCT` в функции `SUM()` добавлено, начиная с версии 5.0.0.

Листинг 21.26. Использование функции `SUM()`

```
mysql> SELECT SUM(price) FROM products;
+------------+
| SUM(price) |
+------------+
| 85769.00   |
+------------+
mysql> SELECT id_catalog, SUM(price) FROM products GROUP BY id_catalog;
+------------+------------+
| id_catalog | SUM(price) |
+------------+------------+
|          1 | 31341.00   |
|          2 | 16807.00   |
|          3 | 15374.00   |
|          4 | 13745.00   |
|          5 | 8502.00    |
+------------+------------+
```

21.12. Функция *VAR_POP()*

Функция `VAR_POP()` возвращает стандартное отклонение значения в столбце *expr* и имеет следующий синтаксис:

```
VAR_POP(expr)
```

> ***Замечание***
>
> Функция имеет синоним `VARIANCE()`.

Замечание

Функция `VARIANCE()` введена в СУБД MySQL, начиная с версии 4.1.0, в то время как функция `VAR_POP()` появилась лишь в версии 5.0.3 для совместимости со стандартом SQL.

21.13. Функция *VAR_SAMP()*

Функция `VAR_SAMP()` возвращает выборочное отклонение значения в аргументе *`expr`* и имеет следующий синтаксис:

```
VAR_SAMP(expr)
```

Замечание

Функция `VAR_SAMP()` введена в СУБД MySQL, начиная с версии 5.0.3.

21.14. Конструкция *WITH ROLLUP()*

При выборке с применением одной из агрегатных функций и конструкции `GROUP BY` выводится список строк с результатами для каждой из групп. Конструкция `WITH ROLLUP` позволяет добавить еще одну строку с суммой значений всех предыдущих строк (листинг 21.27).

Листинг 21.27. Использование конструкции `WITH ROLLUP`

```
mysql> SELECT id_catalog, SUM(price) FROM products
    -> GROUP BY id_catalog WITH ROLLUP;
+------------+------------+
| id_catalog | SUM(price) |
+------------+------------+
|          1 | 31341.00   |
|          2 | 16807.00   |
|          3 | 15374.00   |
|          4 | 13745.00   |
|          5 | 8502.00    |
|       NULL | 85769.00   |
+------------+------------+
```

Как видно из листинга 21.27, 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`.

Конструкция `WITH ROLLUP` может использоваться и в более сложном случае, когда в конструкции `GROUP BY` указывается не один, а сразу несколько столбцов. Тогда итоговая сумма генерируется для каждой из групп (листинг 21.28).

Листинг 21.28. Использование конструкции `WITH ROLLUP` при группировке по двум столбцам

```
mysql> SELECT id_catalog, count, SUM(price)
    -> FROM products
    -> GROUP BY id_catalog, count WITH ROLLUP;
+------------+-------+------------+
| id_catalog | count | SUM(price) |
+------------+-------+------------+
|          1 |     1 | 8109.00    |
|          1 |     2 | 1969.00    |
|          1 |     4 | 2109.00    |
|          1 |     5 | 7259.00    |
|          1 |     6 | 4627.00    |
|          1 |    10 | 1595.00    |
|          1 |    12 | 5673.00    |
|          1 |  NULL | 31341.00   |
|          2 |     2 | 2420.00    |
|          2 |     4 | 7291.00    |
|          2 |     5 | 2289.00    |
|          2 |     6 | 4807.00    |
|          2 |  NULL | 16807.00   |
|          3 |     2 | 5156.00    |
|          3 |     3 | 2730.00    |
|          3 |     6 | 7488.00    |
|          3 |  NULL | 15374.00   |
|          4 |     3 | 3139.00    |
|          4 |     4 | 3589.00    |
|          4 |     5 | 2093.00    |
|          4 |     6 | 2456.00    |
|          4 |     8 | 2468.00    |
|          4 |  NULL | 13745.00   |
|          5 |     8 | 1717.00    |
|          5 |    10 | 899.00     |
|          5 |    12 | 1690.00    |
|          5 |    15 | 1179.00    |
|          5 |    20 | 3017.00    |
```

```
|          5 |  NULL | 8502.00    |
|       NULL |  NULL | 85769.00   |
+------------+-------+------------+
```

В запросе из листинга 21.28 группировка производится по полю `id_catalog`, а также по полю `count`. При этом суперагрегатная строка с суммой выводится шесть раз: пять раз для групп `count` и один раз для группы `id_catalog`.

Важно отметить, что порядок следования столбцов в конструкции `GROUP BY` имеет принципиальное значение: сначала группировка производится для первого столбца, затем для второго и т. д. Так если поменять местами столбцы `id_catalog` и `count` в запросе из листинга 21.28, то результат будет совершенно другим: сначала группировка будет произведена по полю `count`, и лишь затем в полученных группах группировка выполнится по полю `id_catalog` (листинг 21.29).

Листинг 21.29. Изменение порядка следования столбцов в конструкции `GROUP BY`

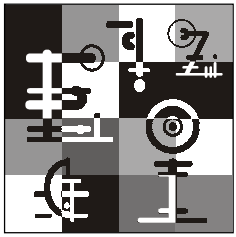
```
mysql> SELECT id_catalog, count, SUM(price)
    -> FROM products
    -> GROUP BY count, id_catalog WITH ROLLUP;
+------------+-------+------------+
| id_catalog | count | SUM(price) |
+------------+-------+------------+
|          1 |     1 | 8109.00    |
|       NULL |     1 | 8109.00    |
|          1 |     2 | 1969.00    |
|          2 |     2 | 2420.00    |
|          3 |     2 | 5156.00    |
|       NULL |     2 | 9545.00    |
|          3 |     3 | 2730.00    |
|          4 |     3 | 3139.00    |
|       NULL |     3 | 5869.00    |
|          1 |     4 | 2109.00    |
|          2 |     4 | 7291.00    |
|          4 |     4 | 3589.00    |
|       NULL |     4 | 12989.00   |
|          1 |     5 | 7259.00    |
|          2 |     5 | 2289.00    |
|          4 |     5 | 2093.00    |
|       NULL |     5 | 11641.00   |
|          1 |     6 | 4627.00    |
|          2 |     6 | 4807.00    |
```

```
|           3 |      6 | 7488.00    |
|           4 |      6 | 2456.00    |
|        NULL |      6 | 19378.00   |
|           4 |      8 | 2468.00    |
|           5 |      8 | 1717.00    |
|        NULL |      8 | 4185.00    |
|           1 |     10 | 1595.00    |
|           5 |     10 | 899.00     |
|        NULL |     10 | 2494.00    |
|           1 |     12 | 5673.00    |
|           5 |     12 | 1690.00    |
|        NULL |     12 | 7363.00    |
|           5 |     15 | 1179.00    |
|        NULL |     15 | 1179.00    |
|           5 |     20 | 3017.00    |
|        NULL |     20 | 3017.00    |
|        NULL |   NULL | 85769.00   |
+-------------+--------+------------+
```

Следует заметить, что совместно с конструкцией `GROUP BY ... WITH ROLLUP` допускается использовать конструкцию `LIMIT`, однако интерпретировать результаты в этом случае достаточно сложно, т. к. суперагрегатные строки считаются наряду с обычными строками результирующей таблицы (листинг 21.30).

Листинг 21.30. Использование конструкции `LIMIT` совместно с `WITH ROLLUP`

```
mysql> SELECT id_catalog, count, SUM(price)
    -> FROM products
    -> GROUP BY count, id_catalog WITH ROLLUP
    -> LIMIT 5;
+-------------+--------+------------+
| id_catalog | count | SUM(price) |
+-------------+--------+------------+
|           1 |      1 | 8109.00    |
|        NULL |      1 | 8109.00    |
|           1 |      2 | 1969.00    |
|           2 |      2 | 2420.00    |
|           3 |      2 | 5156.00    |
+-------------+--------+------------+
```



Глава 22

Разные функции

Данная глава завершает рассмотрение встроенных функций СУБД MySQL. Здесь представлены функции, которые не были описаны в предыдущих главах.

22.1. Функции управления потоком выполнения

Функции управления потоком предназначены для принятия решений в ходе выполнения SQL-инструкций.

22.1.1. Функция *CASE()*

Функция `CASE()` (листинг 22.1) имеет две формы записи, первая из которых выглядит следующим образом:

```
CASE value WHEN [compare-value] THEN result
          [WHEN [compare-value] THEN result ...]
          [ELSE result] END
```

Функция возвращает значение *result*, когда значение *value* совпадает со значением *compare-value*. Если значения не совпадают, управление передается следующей конструкции `WHEN...THEN`, число которых не ограничено, в конечном итоге управление передается конструкции `ELSE`. Если конструкция `ELSE` отсутствует, функция возвращает `NULL`.

Пример запроса представлен в листинге 22.1.

Листинг 22.1. Использование функции `CASE()`

```
mysql> SELECT CASE 1 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'more' END;
+-------------------------------------------------------------+
| CASE 1 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'more' END |
+-------------------------------------------------------------+
```

```
| one                                                          |
+--------------------------------------------------------------+
mysql> SELECT CASE 'C' WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
+-------------------------------------------+
| CASE 'C' WHEN 'a' THEN 1 WHEN 'b' THEN 2 END |
+-------------------------------------------+
|                                      NULL |
+-------------------------------------------+
```

Вторая форма функции `CASE()` имеет следующий синтаксис:

```
CASE WHEN [condition] THEN result
          [WHEN [condition] THEN result ...]
          [ELSE result] END
```

Вторая форма функции `CASE()` (листинг 22.2) аналогична первой, только после ключевого слова `WHEN` следует логическое условие *condition*. Если оно равно 1 (истина), возвращается значение *result*, если 0 (ложь) — управление передается следующей инструкции `WHEN` или `ELSE`. Так же как и в первом случае, если отсутствует подходящее значение *condition*, возвращается `NULL`.

Листинг 22.2. Альтернативное использование функции `CASE()`

```
mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
+----------------------------------------+
| CASE WHEN 1>0 THEN 'true' ELSE 'false' END |
+----------------------------------------+
| true                                   |
+----------------------------------------+
mysql> SELECT CASE WHEN 1<0 THEN 'true' END;
+------------------------------+
| CASE WHEN 1<0 THEN 'true' END |
+------------------------------+
| NULL                         |
+------------------------------+
```

22.1.2. Функция *IF()*

Функция `IF()` (листинг 22.3) предназначена для логического выбора и имеет следующий синтаксис:

```
IF(expr1, expr2, expr3)
```

Если выражение *expr1* истинно (не равно 0 или `NULL`), функция `IF()` возвращает *expr2*, в противном случае — *expr3*. Функция возвращает числовое или строковое значение в зависимости от контекста, в котором выполняется.

Листинг 22.3. Использование функции `IF()`

```
mysql> SELECT IF(1>2,2,3), IF(1<2,'yes','no');
+-------------+--------------------+
| IF(1>2,2,3) | IF(1<2,'yes','no') |
+-------------+--------------------+
|           3 | yes                |
+-------------+--------------------+
```

Наиболее часто данная функция применяется для условного вывода. Пусть стоит задача мониторинга количества комплектующих на складе в учебном электронном магазине `shop`. Если число одного вида товаров не менее 5, то напротив данной товарной позиции будет выведена надпись "Достаточно", в противном случае выведем надпись "Заканчивается" (листинг 22.4).

Листинг 22.4. Определение числа товарных позиций на складе

```
mysql> SELECT name,
    ->        count,
    ->        IF(count>5,'Достаточно','Заканчивается') AS status
    -> FROM products LIMIT 10;
+-----------------------+-------+---------------+
| name                  | count | status        |
+-----------------------+-------+---------------+
| Celeron 1.8           |    10 | Достаточно    |
| Celeron 2.0GHz        |     2 | Заканчивается |
| Celeron 2.4GHz        |     4 | Заканчивается |
| Celeron D 320 2.4GHz  |     1 | Заканчивается |
| Celeron D 325 2.53GHz |     6 | Достаточно    |
| Celeron D 315 2.26GHz |     6 | Достаточно    |
| Intel Pentium 4 3.2GHz|     5 | Заканчивается |
| Intel Pentium 4 3.0GHz|     1 | Заканчивается |
| Intel Pentium 4 3.0GHz|    12 | Достаточно    |
| Gigabyte GA-8I848P-RS |     4 | Заканчивается |
+-----------------------+-------+---------------+
```

В листинге 22.4 информация выводится только для первых 10 товарных позиций из таблицы `products`.

22.1.3. Функция *IFNULL()*

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 22.5 приводятся примеры использования функции: везде, где в качестве первого аргумента передается `NULL` или выражение, приводящее к `NULL`, функция возвращает значение второго аргумента.

Листинг 22.5. Использование функции `IFNULL()`

```
mysql> SELECT IFNULL(0,1), IFNULL(NULL,1);
+-------------+----------------+
| IFNULL(0,1) | IFNULL(NULL,1) |
+-------------+----------------+
|           0 | 1              |
+-------------+----------------+
mysql> SELECT IFNULL(1/0,1), IFNULL(0/1,1);
+---------------+---------------+
| IFNULL(1/0,1) | IFNULL(0/1,1) |
+---------------+---------------+
| 1             | 0.00000       |
+---------------+---------------+
```

Возвращаясь к учебной базе данных `shop`, заменим при выводе из таблицы `users` ключевое слово `NULL` на более понятное неискушенному пользователю слово "отсутствует" (листинг 22.6).

Листинг 22.6. Замена ключевого слова `NULL` на слово "отсутствует"

```
mysql> SELECT phone, email, url FROM users;
+------------+----------------------+-------------------------+
| phone      | email                | url                     |
+------------+----------------------+-------------------------+
| 58-98-78   | ivanov@email.ru      | NULL                    |
| 9057777777 | losev@email.ru       | NULL                    |
| 9056666100 | simdyanov@softtime.ru | http://www.softtime.ru/ |
| NULL       | kuznetsov@softtime.ru | http://www.softtime.ru  |
```

```
| NULL       | NULL                  | NULL                    |
| 89-78-36   | korneev@domen.ru      | NULL                    |
+------------+-----------------------+-------------------------+
mysql> SELECT IFNULL(phone,'отсутствует') AS phone,
    ->        IFNULL(email,'отсутствует') AS email,
    ->        IFNULL(url,'отсутствует') AS url
    -> FROM users;
+-------------+----------------------+-------------------------+
| phone       | email                | url                     |
+-------------+----------------------+-------------------------+
| 58-98-78    | ivanov@email.ru      | отсутствует             |
| 9057777777  | losev@email.ru       | отсутствует             |
| 9056666100  | simdyanov@softtime.ru | http://www.softtime.ru/ |
| отсутствует | kuznetsov@softtime.ru | http://www.softtime.ru  |
| отсутствует | отсутствует          | отсутствует             |
| 89-78-36    | korneev@domen.ru     | отсутствует             |
+-------------+----------------------+-------------------------+
```

22.1.4. Функция *NULLIF()*

Функция `NULLIF(expr1, expr2)` возвращает `NULL`, если значение *expr1* совпадает с *expr2*, в противном случае возвращается значение *expr1* (листинг 22.7).

> ***Замечание***
>
> Функция `NULLIF(expr1, expr2)` аналогична выражению `CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END`.

Листинг 22.7. Использование функции `NULLIF()`

```
mysql> SELECT NULLIF(1,1),NULLIF(1,2);
+-------------+-------------+
| NULLIF(1,1) | NULLIF(1,2) |
+-------------+-------------+
|        NULL |           1 |
+-------------+-------------+
```

22.2. Информационные функции

Информационные функции возвращают различную служебную и системную информацию.

22.2.1. Функция *BENCHMARK()*

Функция `BENCHMARK(count, expr)` повторяет выполнение выражения *expr* заданное количество раз, указанное в аргументе *count*. Она 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.

Указанное в отчете время представляет собой время, подсчитанное на стороне клиента, а не затраченное центральным процессором (CPU time) на сервере. Может оказаться целесообразным выполнить `BENCHMARK()` несколько раз, чтобы выяснить, насколько интенсивно загружен серверный компьютер.

Пример запроса приведен в листинге 22.8.

Листинг 22.8. Использование функции `BENCHMARK()`

```
mysql> SELECT BENCHMARK(1000000,ENCODE("hello","goodbye"));
+-------------------------------------------+
| BENCHMARK(1000000,ENCODE("hello","goodbye")) |
+-------------------------------------------+
|                                         0 |
+-------------------------------------------+
1 row in set (0.61 sec)
```

22.2.2. Функция *CONNECTION_ID()*

Функция `CONNECTION_ID()` возвращает идентификатор (`thread_id`) для данного соединения (листинг 22.9).

Листинг 22.9. Использование функции `CONNECTION_ID()`

```
mysql> SELECT CONNECTION_ID();
+-----------------+
| CONNECTION_ID() |
+-----------------+
|               1 |
+-----------------+
```

Каждое соединение имеет собственный уникальный идентификатор. В листинге 22.9 функция вернула идентификатор, равный единице. Если открыть еще одно окно с консольным клиентом и вызвать в нем функцию `CONNECTION_ID()`, она вернет уже цифру 2.

22.2.3. Функция *CURRENT_USER()*

Функция `CURRENT_USER()` возвращает текущее имя, под которым пользователь аутентифицировался в текущей сессии, и имя хоста, с которого он обращается к серверу (листинг 22.10).

> ***Замечание***
>
> Сервер возвращает имя пользователя в кодировке UTF8.

Листинг 22.10. Использование функции `CURRENT_USER()`

```
mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| root@localhost |
+----------------+
```

Как видно из листинга 22.10, в текущей сессии пользователь зашел под именем `root` с локального компьютера.

22.2.4. Функция *DATABASE()*

Функция `DATABASE()` возвращает имя текущей базы данных (листинг 22.11).

> ***Замечание***
>
> Начиная с версии MySQL 5.0.2, для функции `DATABASE()` введен синоним `SCHEMA()`.

Листинг 22.11. Использование функции `DATABASE()`

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

Если в данное время нет активной базы данных, то функция `DATABASE()` возвращает `NULL`.

> ***Замечание***
>
> До версии СУБД MySQL 4.1.1 вместо `NULL` функция `DATABASE()` возвращала пустую строку.

22.2.5. Функция *FOUND_ROWS()*

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 опцию SQL_CALC_FOUND_ROWS. После этого можно вызывать функцию FOUND_ROWS().

> ***Замечание***
>
> Функция FOUND_ROWS() и опция SQL_CALC_FOUND_ROWS доступны, начиная с версии MySQL 4.0.0.

Пример запроса представлен в листинге 22.12.

Листинг 22.12. Использование функции FOUND_ROWS()

```
mysql> SELECT SQL_CALC_FOUND_ROWS name FROM products LIMIT 10;
+-----------------------+
| name                  |
+-----------------------+
| Celeron 1.8           |
| Celeron 2.0GHz        |
| Celeron 2.4GHz        |
| Celeron D 320 2.4GHz  |
| Celeron D 325 2.53GHz |
| Celeron D 315 2.26GHz |
| Intel Pentium 4 3.2GHz|
| Intel Pentium 4 3.0GHz|
| Intel Pentium 4 3.0GHz|
| Gigabyte GA-8I848P-RS |
+-----------------------+
mysql> SELECT FOUND_ROWS();
+--------------+
| FOUND_ROWS() |
+--------------+
|           30 |
+--------------+
```

Если опция SQL_CALC_FOUND_ROWS в операторе SELECT будет пропущена, функция вернет количество строк, подсчитанное оператором SELECT. Применительно к листингу 22.12 это число равно 10.

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).

22.2.6. Функция *LAST_INSERT_ID()*

Функция LAST_INSERT_ID() возвращает последнее автоматически сгенерированное значение для столбца, снабженного атрибутом AUTO_INCREMENT.

Замечание

Функция LAST_INSERT_ID() обсуждается также в *главе 6* при разборе синтаксиса оператора INSERT.

Следует отметить, что функция LAST_INSERT_ID() возвращает последнее сгенерированное значение в рамках данного сеанса. То есть функция вернет значение только в том случае, если новое значение было сгенерировано непосредственно перед вызовом функции LAST_INSERT_ID().

Последнее сгенерированное значение сохраняется на сервере для данного конкретного соединения и не может быть изменено другим клиентом. Оно не изменится даже при обновлении столбца AUTO_INCREMENT конкретной величиной (т. е. не равной NULL и 0). При использовании многострочного варианта оператора INSERT функция LAST_INSERT_ID() возвращает значение для первой внесенной строки. Причина этого заключается в том, что можно легко воспроизвести точно такую же команду INSERT на другом сервере. Пример использования функции LAST_INSERT_ID() приведен в листинге 22.13.

Листинг 22.13. Использование функции LAST_INSERT_ID()

```
mysql> INSERT INTO catalogs VALUES(NULL, 'Принтеры');
mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                6 |
+------------------+
```

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. Это можно использовать для создания последовательностей (листинг 22.14).

Листинг 22.14. Создание последовательностей

```
mysql> CREATE TABLE sequence (id INT NOT NULL);
mysql> INSERT INTO sequence VALUES (0);
mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
```

Оператор `UPDATE` увеличивает счетчик последовательности и заставляет следующий вызов `LAST_INSERT_ID()` возвращать измененное значение. Для столбца, снабженного атрибутом `AUTO_INCREMENT`, последовательность может быть сгенерирована автоматически без вызова функции `LAST_INSERT_ID()`.

> ***Замечание***
>
> Управлять значениями, которые будут возвращаться функцией `LAST_INSERT_ID()`, можно при помощи системных переменных `IDENTITY`, `INSERT_ID` и `LAST_INSERT_ID`, значения которых устанавливаются посредством оператора `SET` *(см. главу 29)*. Просмотреть текущее значение системных переменных можно при помощи запроса вида `SELECT @@IDENTITY`, в котором значение системной переменной предваряется двойным знаком `@`. Более подробно работа с переменными MySQL рассматривается в *главе 23*.

Вернуть последнюю вставленную посредством оператора `INSERT` запись можно при помощи запроса, представленного в листинге 22.15.

Листинг 22.15. Последняя вставленная запись

```
mysql> SELECT * FROM catalogs WHERE id_catalog = LAST_INSERT_ID();
+------------+----------+
| id_catalog | name     |
+------------+----------+
|          6 | Принтеры |
+------------+----------+
```

Однако механизм `AUTO_INCREMENT` позволяет также в этом случае использовать условие `id_catalog IS NULL` (листинг 22.16).

Листинг 22.16. Альтернативный способ извлечения последней записи

```
mysql> SELECT * FROM catalogs WHERE id_catalog IS NULL;
+------------+----------+
| id_catalog | name     |
+------------+----------+
|          6 | Принтеры |
+------------+----------+
```

Такое поведение СУБД MySQL организовано для совместимости с другими СУБД. Оно может быть отключено установкой системной переменной `SQL_AUTO_IS_NULL` в 0 (листинг 22.17).

Листинг 22.17. Управление системной переменной `SQL_AUTO_IS_NULL`

```
mysql> SELECT @@SQL_AUTO_IS_NULL;
+--------------------+
| @@sql_auto_is_null |
+--------------------+
|                  1 |
+--------------------+
mysql> SET SQL_AUTO_IS_NULL = 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 29*.

22.2.7. Функция *ROW_COUNT()*

Функция `ROW_COUNT()` возвращает число записей, которые подверглись изменению в последнем SQL-запросе. Под изменениями здесь подразумевается обновление, вставка или удаление.

Пример запроса представлен в листинге 22.18.

Листинг 22.18. Использование функции `ROW_COUNT()`

```
mysql> INSERT INTO catalogs VALUES (NULL, 'Раздел 1'),
    ->                             (NULL, 'Раздел 2'),
    ->                             (NULL, 'Раздел 3');
Query OK, 3 rows affected (0.03 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> SELECT ROW_COUNT();
+-------------+
| ROW_COUNT() |
+-------------+
|           3 |
+-------------+
1 row in set (0.00 sec)
mysql> SELECT ROW_COUNT();
```

```
+-------------+
| ROW_COUNT() |
+-------------+
|          -1 |
+-------------+
mysql> DELETE FROM catalogs;
Query OK, 8 rows affected (0.00 sec)
mysql> SELECT ROW_COUNT();
+-------------+
| ROW_COUNT() |
+-------------+
|           8 |
+-------------+
```

Как видно из листинга 22.18, функция `ROW_COUNT()` возвращает число затронутых записей только для операторов `INSERT`, `UPDATE` и `DELETE` (`TRUNCATE TABLE`). Если последний SQL-запрос не изменяет ни одной записи, то возвращается –1. Последнее необходимо на тот случай, если операторы `INSERT`, `UPDATE` и `DELETE` не затронут ни одной из записей, в этом случае функция `ROW_COUNT()` возвращает 0. Поэтому по результату функции всегда можно различить ситуации, когда осуществляется попытка изменить записи и когда такая попытка не производится. Так, например, вызов функции после оператора `SELECT` всегда приводит к значению –1 (листинг 22.19).

Листинг 22.19. После оператора `SELECT` функция `ROW_COUNT()` всегда возвращает –1

```
mysql> SELECT name FROM products;
+------------------------------+
| name                         |
+------------------------------+
| Celeron 1.8                  |
...
| Gigabyte GA-8I848P-RS        |
...
| DDR-400 512MB Hynix          |
+------------------------------+
30 rows in set (0.00 sec)
mysql> SELECT ROW_COUNT();
+-------------+
```

```
| ROW_COUNT() |
+-------------+
|          -1 |
+-------------+
```

Если функция `ROW_COUNT()` вызывается в условиях, когда не было осуществлено ни одного SQL-запроса, ее результат может принимать неопределенное значение (листинг 22.20).

Листинг 22.20. Вызов функции `ROW_COUNT()` без предварительных запросов

```
mysql> SELECT ROW_COUNT();
+---------------+
| ROW_COUNT()   |
+---------------+
| 2834678415360 |
+---------------+
```

22.2.8. Функция *USER()*

Функция `USER()` возвращает имя текущего пользователя и имя хоста, с которого он обращается к СУБД MySQL (листинг 22.21).

> ***Замечание***
>
> Функция `USER()` имеет два синонима: `SESSION_USER()` и `SYSTEM_USER()`.

Листинг 22.21. Использование функции `USER()`

```
mysql> SELECT USER();
+----------------+
| USER()         |
+----------------+
| root@localhost |
+----------------+
```

22.2.9. Функция *VERSION()*

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 22.22).

Листинг 22.22. Использование функции VERSION() в Windows

```
mysql> SELECT VERSION();
+---------------+
| VERSION()     |
+---------------+
| 5.0.6-beta-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 22.23.

Листинг 22.23. Использование функции VERSION() в Linux

```
mysql> 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.

22.3. Разные функции

В англоязычной литературе этот вид функций обозначается как Miscellaneous-функции, что переводится как смешанные функции, т. е. функции, которые не поддаются классификации.

22.3.1. Функция *DEFAULT()*

Функция `DEFAULT(col_name)` возвращает значение по умолчанию для столбца *`col_name`*, которое назначается при помощи ключевого слова `DEFAULT` *(см. главу 12)*.

> ***Замечание***
>
> Функция `DEFAULT()` введена в СУБД MySQL, начиная с версии 4.1.0. В версиях, начиная с 5.0.2, генерируется ошибка, если функция применяется к столбцу без значения по умолчанию.

Пример запроса с функцией `DEFAULT()` представлен в листинге 22.24.

Листинг 22.24. Использование функции `DEFAULT()`

```
mysql> DESCRIBE orders;
+------------+----------+------+-----+--------------------+--------------
+
| Field      | Type     | Null | Key | Default            | Extra
|
+------------+----------+------+-----+--------------------+--------------
+
| id_order   | int(11)  | NO   | PRI | NULL
|auto_increment|
| id_user    | int(11)  | NO   | MUL | 0                  |
|
| ordertime  | datetime | NO   |     | 0000-00-00 00:00:00 |
|
| number     | int(11)  | NO   |     | 0                  |
|
| id_product | int(11)  | NO   | MUL | 0                  |
|
+------------+----------+------+-----+--------------------+--------------
+
mysql> SELECT DEFAULT(ordertime) FROM orders LIMIT 1;
+---------------------+
| DEFAULT(ordertime)  |
+---------------------+
| 0000-00-00 00:00:00 |
+---------------------+
mysql> SELECT DEFAULT(id_order) FROM orders LIMIT 1;
+-------------------+
| DEFAULT(id_order) |
+-------------------+
|                 0 |
+-------------------+
```

Как видно из листинга 22.24, при использовании функции `DEFAULT()` совместно с оператором `SELECT` требуется ограничивать вывод при помощи инструкции `LIMIT`. В противном случае число строк в результирующей таблице будет совпадать с количеством строк в таблице `orders`.

22.3.2. Функция *GET_LOCK()*

Функция `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`.

> ***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.

Эта функция может использоваться для осуществления блокировок уровня приложения или 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. Пример использования функции `GET_LOCK()` приведен в листинге 22.25.

> ***Замечание***
>
> Имена блокировок `str` могут быть любыми и вовсе не должны совпадать с именами баз данных и таблиц, хотя это не запрещается.

Листинг 22.25. Использование функции `GET_LOCK()`

```
mysql> SELECT GET_LOCK('lock1',10), GET_LOCK('lock2',10);
+--------------------+--------------------+
| GET_LOCK('lock1',10) | GET_LOCK('lock2',10) |
+--------------------+--------------------+
|                  1 |                  1 |
+--------------------+--------------------+
mysql> 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 блокировки `GET_LOCK('lock2',10)`.

22.3.3. Функция *INET_ATON()*

Функция `INET_ATON(address)` принимает IP-адрес *address* и представляет его в виде целого числа (листинг 22.26). Для числа `XXX.YYY.ZZZ.WWW` результат функции вычисляется по формуле:

$$XXX \times 256^3 + YYY \times 256^2 + ZZZ \times 256 + WWW.$$

> ***Замечание***
>
> Для того чтобы можно было поместить в целочисленное поле весь диапазон IP-адресов, следует использовать тип `BIGINT`.

Листинг 22.26. Использование функции `INET_ATON()`

```
mysql> 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 22.27.

Листинг 22.27. Работа с сокращенной формой IP-адреса

```
mysql> SELECT INET_ATON('127.0.01'), INET_ATON('127.1');
+----------------------+--------------------+
| INET_ATON('127.0.01') | INET_ATON('127.1') |
+----------------------+--------------------+
|            2130706433 |         2130706433 |
+----------------------+--------------------+
```

22.3.4. Функция *INET_NTOA()*

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 22.28).

Листинг 22.28. Работа с функцией `INET_NTOA()`

```
mysql> SELECT INET_NTOA(1049707786), INET_NTOA(2130706433);
+----------------------+----------------------+
| INET_NTOA(1049707786) | INET_NTOA(2130706433) |
+----------------------+----------------------+
| 62.145.69.10          | 127.0.0.1            |
+----------------------+----------------------+
```

22.3.5. Функция *IS_FREE_LOCK()*

Функция `IS_FREE_LOCK(str)` (листинг 22.29) 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.

> ***Замечание***
>
> Функция может вернуть значение `NULL`, если произошла ошибка, такая как переполнение памяти или уничтожение потока.

Листинг 22.29. Использование функции `IS_FREE_LOCK()`

```
mysql> SELECT GET_LOCK('lock1',10), GET_LOCK('lock2',10);
+--------------------+--------------------+
| GET_LOCK('lock1',10) | GET_LOCK('lock2',10) |
+--------------------+--------------------+
|                  1 |                  1 |
+--------------------+--------------------+
mysql> SELECT IS_FREE_LOCK('lock1'), IS_FREE_LOCK('lock2');
+----------------------+----------------------+
| IS_FREE_LOCK('lock1') | IS_FREE_LOCK('lock2') |
+----------------------+----------------------+
|                    1 |                    0 |
+----------------------+----------------------+
```

Следует отметить, что вызов функции `IS_FREE_LOCK('lock1')` возвращает 1 (истина), т. к. блокировка `'lock1'` была автоматически снята установкой блокировки `GET_LOCK('lock2',10)`.

22.3.6. Функция *IS_USED_LOCK()*

Функция `IS_USED_LOCK(str)` проверяет, установлена ли блокировка с именем *str* (листинг 22.30). В случае, если блокировка установлена, функция возвращает идентификатор соединения клиента, который удерживает блокировку. Если блокировка не установлена, возвращается `NULL`.

Листинг 22.30. Использование функции `IS_USED_LOCK()`

```
mysql> SELECT GET_LOCK('lock1',10), GET_LOCK('lock2',10);
+--------------------+--------------------+
| GET_LOCK('lock1',10) | GET_LOCK('lock2',10) |
+--------------------+--------------------+
|                  1 |                  1 |
+--------------------+--------------------+
mysql> SELECT IS_USED_LOCK('lock1'), IS_USED_LOCK('lock2');
+----------------------+----------------------+
| IS_USED_LOCK('lock1') | IS_USED_LOCK('lock2') |
+----------------------+----------------------+
|                 NULL |                    2 |
+----------------------+----------------------+
```

Замечание

Функция `IS_USED_LOCK()` введена в СУБД MySQL, начиная с версии 4.1.0.

22.3.7. Функция *NAME_CONST()*

Функция `NAME_CONST()` принимает в качестве аргументов два параметра и имеет следующий синтаксис:

`NAME_CONST(`*name*`, `*value*`)`

Функция возвращает значение параметра *value*, назначая в качестве имени столбца строку *name*. Пример использования функции `NAME_CONST()` приводится в листинге 22.31.

Замечание

Функция `NAME_CONST()` введена в СУБД MySQL, начиная с версии 5.0.12.

Листинг 22.31. Использование функции NAME_CONST()

```
mysql> SELECT NAME_CONST('myname', 14);
+--------+
| myname |
+--------+
|     14 |
+--------+
```

Функция используется главным образом для внутренних потребностей сервера, точно так же, как функция PASSWORD(), которая применяется для шифрования паролей MySQL-пользователей и редко вызывается в клиентском коде. Сервер MySQL использует данную функцию для возвращения значений локальных переменных в хранимых процедурах, которые рассматриваются более подробно в *главе 33*.

22.3.8. Функция *RELEASE_LOCK()*

Функция 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.

22.3.9. Функция *SLEEP()*

Функция 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.

> ***Замечание***
>
> Функция SLEEP() введена в СУБД MySQL, начиная с версии 5.0.12.

22.3.10. Функция *UUID()*

Функция UUID() (листинг 22.32) возвращает *универсальный уникальный идентификатор* (Universal Unique Identifier, UUID), сгенерированный в соответствии со спецификацией "DCE 1.1: удаленный вызов процедур" (приложение А), 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 22.32. Использование функции `UUID()`

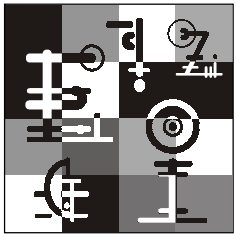
```
mysql> SELECT UUID();
+--------------------------------------+
| UUID()                               |
+--------------------------------------+
| 6627c9e4-2fec-1028-9007-551f9bad91cf |
+--------------------------------------+
mysql> SELECT UUID();
+--------------------------------------+
| UUID()                               |
+--------------------------------------+
| 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 23

Переменные и временные таблицы

Часто результаты запроса необходимо использовать в последующих запросах. Для этого полученные данные следует сохранить во временных структурах. Эту задачу решают переменные SQL и временные таблицы, синтаксис которых рассматривается в данной главе.

23.1. Переменные SQL

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 23.1).

Листинг 23.1. Объявление и использование переменных SQL

```
mysql> SELECT @total := COUNT(*) FROM products;
+--------------------+
| @total := COUNT(*) |
+--------------------+
|                 30 |
+--------------------+
mysql> SELECT @total;
+--------+
| @total |
+--------+
|     30 |
+--------+
```

В листинге 23.1 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 23.2 из таблицы `products` извлекается информация о товарной позиции, обладающей самой высокой ценой.

Листинг 23.2. Извлечение товарной позиции с самой высокой ценой

```
mysql> SELECT @price := MAX(price) FROM products;
+----------------------+
| @price := MAX(price) |
+----------------------+
|              7259.00 |
+----------------------+
mysql> SELECT id_product, name, price, count
    -> FROM products WHERE price = @price;
+------------+-----------------------+---------+-------+
| id_product | name                  | price   | count |
+------------+-----------------------+---------+-------+
|          7 | Intel Pentium 4 3.2GHz | 7259.00 |     5 |
+------------+-----------------------+---------+-------+
```

Если в качестве значения переменной передается имя столбца, то переменная получит последнее значение (листинг 23.3).

Листинг 23.3. Передача в качестве значения переменной имени столбца

```
mysql> SELECT @id := id_catalog FROM catalogs;
+-------------------+
| @id := id_catalog |
+-------------------+
|                 1 |
|                 2 |
|                 3 |
|                 4 |
|                 5 |
+-------------------+
```

```
mysql> SELECT @id;
+------+
| @id  |
+------+
|    5 |
+------+
mysql> SELECT * FROM catalogs WHERE id_catalog = @id;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          5 | Оперативная память |
+------------+--------------------+
```

В качестве значения переменной необязательно должны выступать результаты выполнения функций и значения столбцов таблицы, можно использовать произвольные числовые и строковые значения (листинг 23.4).

Листинг 23.4. Использование для инициализации переменной числового значения

```
mysql> SELECT @id := 3;
+----------+
| @id := 3 |
+----------+
|        3 |
+----------+
mysql> SELECT * FROM catalogs WHERE id_catalog = @id;
+------------+---------------+
| id_catalog | name          |
+------------+---------------+
|          3 | Видеоадаптеры |
+------------+---------------+
```

Следует помнить, что имена переменных чувствительны к регистру (листинг 23.5).

Листинг 23.5. Имена переменных чувствительны к регистру

```
mysql> SELECT @id := 5, @ID := 3;
+----------+----------+
| @id := 5 | @ID := 3 |
```

```
+----------+----------+
|        5 |        3 |
+----------+----------+
mysql> SELECT @id, @ID;
+------+------+
| @id  | @ID  |
+------+------+
|    5 |    3 |
+------+------+
```

Переменные также могут объявляться при помощи оператора `SET`, как это продемонстрировано в листинге 23.6.

> ***Замечание***
>
> При использовании оператора `SET` в качестве оператора присваивания может выступать как `:=`, так и обычный знак равенства `=`.

> ***Замечание***
>
> Синтаксис оператора `SET` рассматривается в *главе 29*.

Листинг 23.6. Использование оператора `SET` для объявления переменной

```
mysql> SET @last = CURDATE() - INTERVAL 7 DAY;
mysql> SELECT CURDATE(), @last;
+------------+------------+
| CURDATE()  | @last      |
+------------+------------+
| 2005-06-18 | 2005-06-11 |
+------------+------------+
```

Удобство применения оператора `SET` заключается в том, что в отличие от оператора `SELECT` он не возвращает результирующую таблицу.

При использовании переменных следует помнить, что не рекомендуется одновременно присваивать и использовать переменную в одном запросе, т. к. это может приводить к ошибкам, которые трудно обнаружить (листинг 23.7).

Листинг 23.7. Особенности присвоения и использования переменных в одном запросе

```
mysql> SET @a = 'test';
mysql> SELECT @a, (@a := 20);
+------+------------+
| @a   | (@a := 20) |
```

```
+------+------------+
| test |         20 |
+------+------------+
mysql> SET @a = 'test';
mysql> SELECT (@a := 20), @a;
+------------+------+
| (@a := 20) | @a   |
+------------+------+
|         20 | 20   |
+------------+------+
```

Как видно из листинга 23.7, порядок следования выражений с участием переменных имеет значение для результата, возвращаемого запросом.

23.2. Временные таблицы

Переменная SQL позволяет сохранить одно промежуточное значение. Когда необходимо сохранить результирующую таблицу, прибегают к временным таблицам.

Создание временных таблиц осуществляется при помощи оператора `CREATE TEMPORARY TABLE`, синтаксис которого ничем не отличается от синтаксиса оператора `CREATE TABLE` *(см. главу 12)*.

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 остается скрытой, пока не будет удалена временная таблица). Пример приведен в листинге 23.8.

Листинг 23.8. Создание временной таблицы `temp`

```
mysql> CREATE TEMPORARY TABLE temp (id_catalog INT, name TINYINT);
mysql> SHOW TABLES;
+---------------+
| Tables_in_shop |
+---------------+
| catalogs       |
| orders         |
| products       |
| users          |
+---------------+
```

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

В листинге 23.8 демонстрируется создание таблицы `temp`. Если возникает необходимость удалить таблицу до момента завершения соединения с сервером, можно воспользоваться оператором `DROP TABLE` (листинг 23.9).

Листинг 23.9. Удаление временной таблицы `temp`

```
mysql> DROP TABLE temp;
```

Временные таблицы можно заполнять традиционными способами, описанными в *главе 6*, но наиболее удобно производить заполнение таких таблиц при помощи вложенных запросов, когда результат запроса `SELECT` непосредственно помещается во временную таблицу (листинг 23.10).

> ***Замечание***
>
> Более подробно вложенные запросы рассматриваются в *главе 24*.

Листинг 23.10. Создание временной таблицы `temp` при помощи вложенного запроса

```
mysql> CREATE TEMPORARY TABLE temp
    -> SELECT id_catalog, COUNT(id_catalog) AS total
    -> FROM products GROUP BY id_catalog;
mysql> SELECT * FROM temp;
+------------+-------+
| id_catalog | total |
+------------+-------+
|          1 |     9 |
|          2 |     6 |
|          3 |     4 |
|          4 |     5 |
|          5 |     6 |
+------------+-------+
```

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

Временная таблица `temp` может выступать в качестве предмета запроса. В листинге 23.11 подсчитывается сумма значений столбца `total`.

Листинг 23.11. Запрос к временной таблице `temp`

```
mysql> SELECT SUM(total) FROM temp;
+------------+
| SUM(total) |
+------------+
|         30 |
+------------+
```

Поместив в переменную `@total` общее число товарных позиций в таблице `products`, можно получить процентное соотношение числа товарных позиций в разных отделах учебного электронного магазина (листинг 23.12).

Листинг 23.12. Процентное отношение числа товарных позиций по отделам

```
mysql> SELECT @total := SUM(total) FROM temp;
+----------------------+
| @total := SUM(total) |
+----------------------+
|                   30 |
+----------------------+
mysql> SELECT id_catalog, FORMAT((total/@total)*100,2) AS percent
    -> FROM temp;
+------------+---------+
| id_catalog | percent |
+------------+---------+
|          1 | 30.00   |
|          2 | 20.00   |
|          3 | 13.33   |
|          4 | 16.67   |
|          5 | 20.00   |
+------------+---------+
```

Системная переменная `BIG_TABLES` позволяет сообщить СУБД MySQL, где должны храниться временные таблицы: в оперативной памяти или на диске. Если переменная

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 23.13).

Листинг 23.13. Управление системной переменной `BIG_TABLES`

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

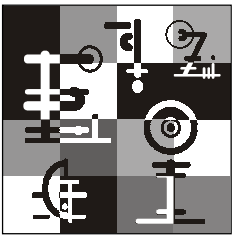
Для того чтобы извлечь текущую системную переменную, ее имя следует предварить двумя символами `@`. Более подробно системные переменные и синтаксис оператора `SET` обсуждается в *главе 29*.

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.

Замечание

В MySQL 5.0.1 появилась серьезная альтернатива временным таблицам в виде представлений *(см. главу 35)*. На самом деле представления часто также являются временными таблицами, только об их создании и уничтожении заботится СУБД MySQL.



Глава 24

Вложенные запросы

Вложенный запрос позволяет использовать результат, возвращаемый одним запросом, в другом запросе. Возможность применения одного запроса внутри другого и была причиной появления слова "структурированный" в названии языка SQL. Так как результат возвращает только оператор `SELECT`, то в качестве "вложенного" запроса всегда выступает `SELECT`-запрос (информационные SQL-запросы с использованием нестандартных операторов `SHOW`, `DESCRIBE` во внимание не принимаются). В качестве внешнего запроса может выступать запрос с участием любого SQL-оператора: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE TABLE` и др.

> ***Замечание***
>
> Термин "вложенный запрос" имеет различные варианты в русскоязычной литературе, такого рода запросы часто называют подчиненными запросами и подзапросами.

> ***Замечание***
>
> Вложенные запросы появились в СУБД MySQL, начиная с версии 4.1.

Пусть требуется вывести название товарных позиций из таблицы `products` для элемента "Процессоры" таблицы `catalogs`. Опираясь на предыдущие главы, решить эту задачу можно, например, при помощи двух SQL-запросов (листинг 24.1). Первый запрос извлекает первичный ключ (`id_catalog`) записи из таблицы `catalogs`, соответствующей элементу "Процессоры". Второй запрос выводит названия товарных позиций и их цены для данного каталога.

Листинг 24.1. Товарные позиции элемента "Процессоры"

```
mysql> SELECT @id := id_catalog FROM catalogs WHERE name = 'Процессоры';
+------------------+
| @id := id_catalog |
+------------------+
|                1 |
+------------------+
```

```
mysql> SELECT name, price FROM products
    -> WHERE id_catalog = @id
    -> ORDER BY price;
+-----------------------+---------+
| name                  | price   |
+-----------------------+---------+
| Celeron 1.8           | 1595.00 |
| Celeron D 315 2.26GHz | 1880.00 |
| Celeron D 320 2.4GHz  | 1962.00 |
| Celeron 2.0GHz        | 1969.00 |
| Celeron 2.4GHz        | 2109.00 |
| Celeron D 325 2.53GHz | 2747.00 |
| Intel Pentium 4 3.0GHz | 5673.00 |
| Intel Pentium 4 3.0GHz | 6147.00 |
| Intel Pentium 4 3.2GHz | 7259.00 |
+-----------------------+---------+
```

> ***Замечание***
>
> Синтаксис пользовательских переменных и их применение подробно рассматриваются в *главе 23*.

Как видно из листинга, промежуточное значение помещается в переменную `@id`. Вложенные запросы позволяют осуществить подобную выборку в одном запросе (листинг 24.2).

Листинг 24.2. Применение вложенного запроса

```
mysql> SELECT name, price FROM products
    -> WHERE id_catalog = (SELECT id_catalog FROM catalogs
    ->                     WHERE name = 'Процессоры')
    -> ORDER BY price;
+-----------------------+---------+
| name                  | price   |
+-----------------------+---------+
| Celeron 1.8           | 1595.00 |
| Celeron D 315 2.26GHz | 1880.00 |
| Celeron D 320 2.4GHz  | 1962.00 |
| Celeron 2.0GHz        | 1969.00 |
| Celeron 2.4GHz        | 2109.00 |
| Celeron D 325 2.53GHz | 2747.00 |
| Intel Pentium 4 3.0GHz | 5673.00 |
```

```
| Intel Pentium 4 3.0GHz | 6147.00 |
| Intel Pentium 4 3.2GHz | 7259.00 |
+------------------------+---------+
```

Здесь в качестве внешнего запроса выступает

```
SELECT name, price 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.

Получить аналогичный результат можно при помощи многотабличного запроса, представленного в листинге 24.3.

Листинг 24.3. Решение задачи при помощи многотабличного запроса

```
mysql> SELECT products.name, products.price FROM catalogs, products
    -> WHERE catalogs.id_catalog = products.id_catalog AND
    ->       catalogs.name = 'Процессоры'
    -> ORDER BY price;
+------------------------+---------+
| name                   | price   |
+------------------------+---------+
| Celeron 1.8            | 1595.00 |
| Celeron D 315 2.26GHz  | 1880.00 |
| Celeron D 320 2.4GHz   | 1962.00 |
| Celeron 2.0GHz         | 1969.00 |
| Celeron 2.4GHz         | 2109.00 |
| Celeron D 325 2.53GHz  | 2747.00 |
| Intel Pentium 4 3.0GHz | 5673.00 |
| Intel Pentium 4 3.0GHz | 6147.00 |
| Intel Pentium 4 3.2GHz | 7259.00 |
+------------------------+---------+
```

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. Они будут рассмотрены в данной главе.

Вложенный запрос может применяться не только с условием WHERE, но также в конструкциях DISTINCT, GROUP BY, ORDER BY, LIMIT, JOIN, UNION, в функциях и т. д.

24.1. Вложенный запрос как скалярный операнд

В данный вид вложенного запроса возвращается одно-единственное значение, которое используется во внешнем запросе. Такой вложенный запрос можно применять в любом месте, где допустимо использование скалярного значения или литерала.

Пусть требуется определить название элемента каталога, в котором присутствует самая дорогая товарная позиция (листинг 24.4).

Листинг 24.4. Определение названий каталога с самой дорогой товарной позицией

```
mysql> SELECT name FROM catalogs
    -> WHERE id_catalog = (SELECT id_catalog FROM products
    ->                     WHERE price = (SELECT MAX(price)
    ->                                    FROM products));
+------------+
| name       |
+------------+
| Процессоры |
+------------+
```

Как видно из листинга 24.4, степень вложения запросов не ограничена вторым уровнем, могут использоваться трехуровневые и многоуровневые вложенные запросы. Первый внутренний запрос определяет максимальную цену в таблице products:

```
SELECT MAX(price) FROM products
```

Внешний по отношению к нему запрос определяет значение первичного ключа (id_catalog) для товарной позиции с максимальным значением цены (7259.00):

```
SELECT id_catalog FROM products WHERE price = 7259.00
```

Полученное значение поля id_catalog подставляется во внешний запрос:

```
SELECT name FROM catalogs WHERE id_catalog = 1
```

> ***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 24.5).

Листинг 24.5. Использование вложенного запроса в списке столбцов

```
mysql> SELECT id_catalog, (SELECT MAX(price) FROM products)
    -> FROM catalogs;
+------------+----------------------------------+
| id_catalog | (SELECT MAX(price) FROM products) |
+------------+----------------------------------+
|          1 | 7259.00                          |
|          2 | 7259.00                          |
|          3 | 7259.00                          |
|          4 | 7259.00                          |
|          5 | 7259.00                          |
|          6 | 7259.00                          |
+------------+----------------------------------+
```

В листинге 24.5 вложенный запрос `SELECT MAX(price) FROM products` возвращает максимальное значение цены из таблицы `products` (`7259.00`) и подставляет это значение на место запроса, поэтому запрос из листинга 24.5 можно представить, как запрос со скалярной величиной в качестве столбца (листинг 24.6).

Листинг 24.6. Запрос со скалярной величиной в качестве столбца

```
mysql> SELECT id_catalog, 7259.00 FROM catalogs;
+------------+---------+
| id_catalog | 7259.00 |
+------------+---------+
|          1 | 7259.00 |
|          2 | 7259.00 |
|          3 | 7259.00 |
|          4 | 7259.00 |
|          5 | 7259.00 |
|          6 | 7259.00 |
+------------+---------+
```

Допустимы также запросы вида, представленного в листинге 24.7. Вложенный запрос возвращает название каталога с первичным ключом `id_catalog = 1`, а внешний запрос, состоящий из одного оператора `SELECT`, выводит его.

Листинг 24.7. Использование вложенного запроса как скалярной величины

```
mysql> 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 24.8 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 24.8. Использование вложенного запроса в качестве аргумента функции

```
mysql> 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 (=, <>, <, <=, >, >=), подробное описание которых можно найти в *главе 15*. Вложенный запрос, участвующий в операции сравнения, должен возвращать в качестве результата единичное значение (листинг 24.9).

Листинг 24.9. Использование вложенных запросов в операциях сравнения

```
mysql> SELECT name FROM catalogs
    -> WHERE id_catalog > (SELECT id_catalog FROM products
    ->                     WHERE id_product = 10);
+--------------------+
| name               |
+--------------------+
| Видеоадаптеры      |
```

```
| Жесткие диски      |
| Оперативная память |
+--------------------+
mysql> SELECT name FROM catalogs
    -> WHERE (SELECT id_catalog FROM products
    ->        WHERE id_product = 10) < id_catalog;
+--------------------+
| name               |
+--------------------+
| Видеоадаптеры      |
| Жесткие диски      |
| Оперативная память |
+--------------------+
```

В листинге 24.9 возвращаются названия элементов каталога, значение первичного ключа `id_catalog` для которых больше значения первичного ключа элемента каталога, в котором находится товарная позиция с `id_product = 10`.

> ***Замечание***
>
> В листинге 24.9 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.

24.2. 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 24.10).

Листинг 24.10. Вложенный запрос возвращает более одной строки

```
mysql> SELECT name FROM catalogs
    -> WHERE id_catalog = (SELECT id_catalog FROM products);
ERROR 1242: Subquery returns more than 1 row
```

24.2.1. Ключевое слово *IN*

Для того чтобы выбрать строки из таблицы `catalogs`, у которых первичный ключ `id_catalog` совпадает с одним из значений, возвращаемых вложенным запросом, следует воспользоваться конструкцией `IN` (листинг 24.11), которая подробно рассмотрена в *главе 7*.

Листинг 24.11. Использование конструкции `IN`

```
mysql> SELECT name FROM catalogs
    -> WHERE id_catalog IN (SELECT id_catalog FROM products
    ->                      GROUP BY id_catalog);
+--------------------+
| name               |
+--------------------+
| Процессоры         |
| Материнские платы  |
| Видеоадаптеры      |
| Жесткие диски      |
| Оперативная память |
+--------------------+
```

Запрос, представленный в листинге 24.11, аналогичен запросу, показанному в листинге 24.12.

Листинг 24.12. Использование конструкции `IN` совместно со списком скалярных величин

```
mysql> SELECT name FROM catalogs WHERE id_catalog IN (1,2,3,4,5);
+--------------------+
| name               |
+--------------------+
| Процессоры         |
| Материнские платы  |
| Видеоадаптеры      |
| Жесткие диски      |
| Оперативная память |
+--------------------+
```

Для того чтобы возвратить строки, которые отсутствуют в результирующей таблице, возвращаемой вложенным запросом, следует воспользоваться оператором `NOT IN`, представленным в листинге 24.13.

Листинг 24.13. Использование оператора NOT IN

```
mysql> SELECT name FROM catalogs
    -> WHERE id_catalog NOT IN (SELECT DISTINCT id_catalog
    ->                          FROM products WHERE id_catalog<3);
+--------------------+
| name               |
+--------------------+
| Видеоадаптеры      |
| Жесткие диски      |
| Оперативная память |
+--------------------+
```

24.2.2. Ключевое слово *ANY (SOME)*

Конструкция IN позволяет осуществить поиск величины в списке и выражает логику запроса, представленного в листинге 24.10. Однако на месте оператора = в этом запросе может стоять другой оператор сравнения (листинг 24.14).

Листинг 24.14. Вложенный запрос возвращает более одной строки

```
mysql> 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 24.15).

> ***Замечание***
>
> Ключевое слово ANY имеет синоним SOME, однако предпочтительнее использовать именно ANY.

Листинг 24.15. Использование ключевого слова ANY

```
mysql> 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 24.15 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*...".

Запрос, представленный в листинге 24.16, возвращает имена и фамилии покупателей из таблицы `users`, совершивших хотя бы одну покупку.

Листинг 24.16. Извлечение списка покупателей, совершивших хотя бы одну покупку

```
mysql> SELECT name, surname FROM users
    -> WHERE id_user = ANY (SELECT id_user FROM orders);
```

```
+-----------+----------+
| name      | surname  |
+-----------+----------+
| Александр | Иванов   |
| Игорь     | Симдянов |
| Александр | Корнеев  |
+-----------+----------+
```

На практике сравнение с использованием ключевого слова `ANY` может приводить к ошибкам, которые трудно выявить, особенно когда применяется оператор сравнения "не равно" (<> или !=). Так, если потребуется вернуть список покупателей, не совершавших ни одной покупки, возникнет искушение заменить оператор равенства = (листинг 24.16) на оператор неравенства <> (листинг 24.17).

Листинг 24.17. Извлечение списка покупателей, не совершавших покупок

```
mysql> SELECT name, surname FROM users
    -> WHERE id_user <> ANY (SELECT id_user FROM orders);
+-----------+-----------+
| name      | surname   |
+-----------+-----------+
| Александр | Иванов    |
| Сергей    | Лосев     |
| Игорь     | Симдянов  |
| Максим    | Кузнецов  |
| Анатолий  | Нехорошев |
| Александр | Корнеев   |
+-----------+-----------+
```

Результатом является полный список покупателей как совершавших, так и не совершавших покупок, т. к. значение `id_user` сравнивается с каждым значением, которое возвращает вложенный запрос. Среди этих значений всегда найдется число, которое не будет равно текущему значению `id_user`. Решить данную задачу можно при помощи ключевых слов `EXISTS` и `NOT EXISTS`, которые рассмотрены в *разд. 24.3*.

24.2.3. 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 24.18 представлен запрос, возвращающий все товарные позиции из таблицы `products` базы данных `shop`, цена которых превышает среднюю цену каждого из элементов каталога.

Листинг 24.18. Использование ключевого слова `ALL`

```
mysql> SELECT name, price FROM products
    -> WHERE price > ALL (SELECT AVG(price) FROM products
    ->                    GROUP BY id_catalog);
+-------------------------+---------+
| name                    | price   |
+-------------------------+---------+
| Intel Pentium 4 3.2GHz  | 7259.00 |
| Intel Pentium 4 3.0GHz  | 6147.00 |
| Intel Pentium 4 3.0GHz  | 5673.00 |
| Asustek P4C800-E Delux  | 5395.00 |
| ASUSTEK A9600XT/TD      | 5156.00 |
| GIGABYTE AGP GV-N59X128D | 5886.00 |
+-------------------------+---------+
```

Для того чтобы понять логику запроса, представленного в листинге 24.18, полезно выполнить вложенный запрос отдельно (листинг 24.19).

Листинг 24.19. Вложенный запрос

```
mysql> SELECT AVG(price) FROM products GROUP BY id_catalog;
+-------------+
| AVG(price)  |
+-------------+
| 3482.333333 |
| 2801.166667 |
| 3843.500000 |
| 2749.000000 |
| 1417.000000 |
+-------------+
```

Условие ALL в листинге 24.18 эквивалентно требованию вывести все товарные позиции, цена которых больше каждого из значений результирующей таблицы запроса, приведенного в листинге 24.19.

То есть запрос вида:

```
"WHERE X > ALL (SELECT Y...)"
```

можно толковать как "где *X* больше любого выбранного *Y*", а запрос вида

```
"WHERE X < ALL (SELECT Y...)"
```

следует читать "где *X* меньше, чем все выбранные *Y*...".

Ошибки, которые могут возникнуть, если проверка ANY содержит оператор <>, происходят и при анализе ALL.

24.3. Проверка 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 возвращает более одной строки, EXISTS дает 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.

Пусть требуется вывести названия товарных позиций из таблицы products, которые были выбраны покупателями, т. е. для которых имеется отметка о покупке в таблице orders (листинг 24.20).

Листинг 24.20. Использование ключевого слова EXISTS

```
mysql> SELECT id_product, name, price FROM products
    -> WHERE EXISTS (SELECT * FROM orders
    ->               WHERE orders.id_product = products.id_product);
+------------+-----------------------+---------+
| id_product | name                  | price   |
+------------+-----------------------+---------+
|          8 | Intel Pentium 4 3.0GHz | 6147.00 |
|         10 | Gigabyte GA-8I848P-RS  | 1896.00 |
|         20 | Maxtor 6Y120P0         | 2456.00 |
+------------+-----------------------+---------+
```

Внешний запрос последовательно перебирает все строки таблицы `products` и для каждой товарной позиции выполняет вложенный запрос, который проверяет, имеется ли в таблице `orders` текущая товарная позиция. Число записей, которые возвращает вложенный запрос, может быть более одной (листинг 24.21).

Листинг 24.21. Содержимое таблицы `orders`

```
mysql> SELECT id_product FROM orders;
+------------+
| id_product |
+------------+
|          8 |
|         10 |
|         20 |
|         20 |
|         20 |
+------------+
```

Как видно из листинга 24.21, для первых двух строк результирующей таблицы листинга 24.20 вложенный запрос вернет по одной строке (`id_product` = 8 и `id_product` = 10), для последней их будет три (`id_product` = 20). То есть ключевое слово `EXISTS` в действительности не использует результаты вложенного запроса, проверяется только число возвращаемых строк. Это означает, что 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 24.22).

Листинг 24.22. Использование скалярного значения во вложенном запросе

```
mysql> SELECT id_product, name, price FROM products
    -> WHERE EXISTS (SELECT 7 FROM orders
    ->               WHERE orders.id_product = products.id_product);
+------------+-----------------------+---------+
| id_product | name                  | price   |
+------------+-----------------------+---------+
|          8 | Intel Pentium 4 3.0GHz | 6147.00 |
|         10 | Gigabyte GA-8I848P-RS  | 1896.00 |
|         20 | Maxtor 6Y120P0         | 2456.00 |
+------------+-----------------------+---------+
```

Важно отметить, что в `WHERE`-условии вложенного запроса допускается использование столбцов внешнего запроса при обращении к ним по полному имени. В листин-

ге 24.22 к текущей товарной позиции из внешнего запроса во вложенном запросе можно обратиться по имени `products.id_product`.

> ***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* и рассматриваются в *разд. 24.4*.

Задачу на формирование списка покупателей, совершивших хотя бы одну покупку (листинг 24.16), можно решить при помощи запроса, представленного в листинге 24.23.

Листинг 24.23. Извлечение списка покупателей, совершивших хотя бы одну покупку, с помощью ключевого слова `EXISTS`

```
mysql> SELECT name, surname FROM users
    -> WHERE EXISTS (SELECT * FROM orders
    ->               WHERE orders.id_user = users.id_user);
+-----------+----------+
| name      | surname  |
+-----------+----------+
| Александр | Иванов   |
| Игорь     | Симдянов |
| Александр | Корнеев  |
+-----------+----------+
```

Обратную задачу, т. е. извлечение списка покупателей, еще не совершавших ни одной покупки, можно легко решить, воспользовавшись отрицанием — `NOT EXISTS` (листинг 24.24).

Листинг 24.24. Извлечение списка покупателей, не совершавших покупок, с помощью ключевого слова `NOT EXISTS`

```
mysql> SELECT name, surname FROM users
    -> WHERE NOT EXISTS (SELECT * FROM orders
    ->                   WHERE orders.id_user = users.id_user);
+----------+-----------+
| name     | surname   |
+----------+-----------+
| Сергей   | Лосев     |
| Максим   | Кузнецов  |
| Анатолий | Нехорошев |
+----------+-----------+
```

24.4. Коррелированные запросы

Во вложенном запросе часто требуется ссылаться на значение столбца в текущей строке внешнего запроса. Рассмотрим запрос, извлекающий из таблицы `products` названия приобретенных товаров, для которых число купленных товарных позиций совпадает с запасами на складе (листинг 24.25).

Листинг 24.25. Товарные позиции и их запас

```
mysql> SELECT name, count FROM products
    -> WHERE count = (SELECT SUM(count) FROM orders
    ->                WHERE orders.id_product = products.id_product);
+-----------------------+-------+
| name                  | count |
+-----------------------+-------+
| Intel Pentium 4 3.0GHz |     1 |
| Gigabyte GA-8I848P-RS  |     4 |
+-----------------------+-------+
```

Во вложенном запросе вместо значения `products.id_product` подставляется текущее значение `id_product` из внешнего запроса. Вложенный запрос содержит ссылку на столбец таблицы `products`, несмотря на то, что конструкция `FROM` вложенного запроса не упоминает таблицу `products`. Такая связь внешнего запроса с внутренним называется *внешней ссылкой*. Вложенный запрос, содержащий внешнюю ссылку, называется *коррелированным вложенным запросом*, т. к. его результаты оказываются коррелированными с каждой строкой таблицы во внешнем запросе. По той же причине внешняя ссылка называется иногда *коррелирующей*.

Вложенный запрос может иметь внешнюю ссылку на таблицу в конструкции `FROM` любого запроса, который содержит данный вложенный запрос, независимо от его уровня вложенности. Например, имя столбца во вложенном запросе четвертого уровня может относиться к одной из таблиц, указанных в конструкции `FROM` внешнего запроса, или к таблице в любом вложенном запросе, содержащем данный вложенный запрос четвертого уровня.

24.5. 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 24.26.

Листинг 24.26. Пример строчных запросов

```
mysql> SELECT * FROM tbl1 WHERE (1,2) = (SELECT col1, col2 FROM tbl2);
mysql> 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 24.27, где из таблицы `products` извлекаются параметры товарных позиций, число которых на складе равно 6, а оценка 4.

Листинг 24.27. Применение конструктора строки

```
mysql> SELECT name, price, count, mark FROM products
    -> WHERE ROW(6,4) = (count,mark);
+--------------------------+---------+-------+------+
| name                     | price   | count | mark |
+--------------------------+---------+-------+------+
| Asustek P4P800-VM\L i865G | 2518.00 |     6 |  4.0 |
| ASUSTEK V9520X           | 1602.00 |     6 |  4.0 |
+--------------------------+---------+-------+------+
```

Запрос из листинга 24.27 можно переписать так, как это представлено в листинге 24.28.

Листинг 24.28. Альтернативный запрос без применения конструктора строки

```
mysql> SELECT name, price, count, mark FROM products
    -> WHERE count = 5 AND mark = 4;
+----------------+---------+-------+------+
| name           | price   | count | mark |
+----------------+---------+-------+------+
| Epox EP-4PDA3I | 2289.00 |     5 |  4.0 |
| Samsung SP0812C | 2093.00 |     5 |  4.0 |
+----------------+---------+-------+------+
```

Пусть требуется выяснить время оформления заказа на товар, цена которого превышает 1000 рублей, всеми посетителями со статусом `gold` (листинг 24.29).

Листинг 24.29. Использование конструктора строки совместно с вложенным запросом

```
mysql> SELECT ordertime FROM orders
    -> WHERE (id_user, id_product) IN (SELECT users.id_user,
    ->                                        products.id_product
    ->                                 FROM users, products
    ->                                 WHERE products.price > 1000 AND
    ->                                       users.status = 'gold');
+---------------------+
| ordertime           |
+---------------------+
| 2005-02-10 09:40:29 |
+---------------------+
```

Как видно из запроса, представленного в листинге 24.29, после конструктора строки используется конструкция `IN`. Это связано с тем, что вложенный запрос может вернуть более одной строки и применение знака равенства = приведет к возникновению ошибки 1242 — "Вложенный запрос возвращает более одной строки".

> ***Замечание***
>
> При сравнении конструктора строки и результатов, которые вернул вложенный запрос, не следует применять операторы, отличные от равенства = и `IN`, т. к. результат может быть непредсказуемым.

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 24.30.

Листинг 24.30. Вывод строк таблицы `tbl1`, присутствующих в таблице `tbl2`

```
mysql> SELECT col1, col2, col3 FROM tbl1
mysql>                          WHERE ROW(col1, col2, col3)
mysql>                          IN (SELECT col1, col2, col3 FROM tbl2);
```

24.6. 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 24.31 представлен запрос, в котором в качестве одной из таблиц используется результирующая таблица вложенного запроса.

Листинг 24.31. Использование вложенных запросов в конструкции FROM

```
mysql> SELECT usr.surname, usr.name, orders.ordertime
    -> FROM orders,
    ->      (SELECT * FROM users) AS usr
    -> WHERE orders.id_user = usr.id_user;
+----------+----------+---------------------+
| surname  | name     | ordertime           |
+----------+----------+---------------------+
| Симдянов | Игорь    | 2005-01-04 10:39:38 |
| Корнеев  | Александр | 2005-02-10 09:40:29 |
| Иванов   | Александр | 2005-02-18 13:41:05 |
| Симдянов | Игорь    | 2005-03-10 18:20:00 |
| Симдянов | Игорь    | 2005-03-17 19:15:36 |
+----------+----------+---------------------+
```

Ключевое слово AS, назначающее псевдоним результирующей таблице, является обязательным, т. к. каждая таблица в конструкции FROM должна иметь имя. Например, запрос, выводящий список всех сделок, с фамилиями и инициалами покупателей, наименованием товарных позиций и каталога, может выглядеть так, как это представлено в листинге 24.32. Запрос является трехтабличным, причем в качестве двух таблиц (users и goods) выступают вложенные запросы. Результирующая таблица users содержит лишь два столбца id_user и name, в которые заносятся фамилия покупателя и его инициалы, выделенные при помощи функции SUBSTRING() и объединенные в одну строку посредством функции CONCAT(). Таблица goods имеет три поля:

- id_product — уникальный номер товарной позиции;
- productname — название товарной позиции;
- catalogname — название элемента каталога.

Интересно отметить, что таблица goods сама сформирована при помощи двухтабличного запроса из таблиц products и catalogs.

Листинг 24.32. Использование вложенных запросов в конструкции FROM

```
mysql> SELECT users.name,
    ->        goods.productname,
    ->        goods.catalogname
    -> FROM orders,
    ->      (
    ->         SELECT id_user,
    ->                CONCAT(surname, " ", SUBSTRING(name,1,1), ".",
```

```
    ->                         SUBSTRING(patronymic,1,1), ".") AS name
    ->           FROM users
    ->       ) AS users,
    ->       (
    ->           SELECT products.id_product AS id_product,
    ->                  products.name AS productname,
    ->                  catalogs.name AS catalogname
    ->           FROM products, catalogs
    ->           WHERE products.id_catalog = catalogs.id_catalog
    ->       ) AS goods
    -> WHERE orders.id_user = users.id_user AND
    ->       orders.id_product = goods.id_product;
+--------------+-----------------------+-------------------+
| name         | productname           | catalogname       |
+--------------+-----------------------+-------------------+
| Симдянов И.В. | Intel Pentium 4 3.0GHz | Процессоры        |
| Корнеев А.А.  | Gigabyte GA-8I848P-RS  | Материнские платы |
| Иванов А.В.   | Maxtor 6Y120P0         | Жесткие диски     |
| Симдянов И.В. | Maxtor 6Y120P0         | Жесткие диски     |
| Симдянов И.В. | Maxtor 6Y120P0         | Жесткие диски     |
+--------------+-----------------------+-------------------+
```

Рассмотрим другой пример. Пусть имеется запрос, который возвращает общее число товаров на складе для каждого из элементов каталога (листинг 24.33).

Листинг 24.33. Число товаров на складе для каждого из элементов каталога

```
mysql> SELECT id_catalog, SUM(count) FROM products GROUP BY id_catalog;
+------------+------------+
| id_catalog | SUM(count) |
+------------+------------+
|          1 | 47         |
|          2 | 27         |
|          3 | 17         |
|          4 | 26         |
|          5 | 85         |
+------------+------------+
```

Требуется определить среднее значение сумм в сгруппированной таблице. Вариант, представленный в листинге 24.34, недопустим.

Листинг 24.34. Ошибочный запрос

```
mysql> SELECT id_catalog, AVG(SUM(count))
    -> FROM products GROUP BY id_catalog;
ERROR 1111: Invalid use of group function
```

Выходом из подобной ситуации является использование запроса, представленного в листинге 24.33, в качестве вложенного запроса в предложении `FROM` (листинг 24.35).

Листинг 24.35. Среднее значение сумм

```
mysql> SELECT AVG(sumtbl.sum) FROM (SELECT id_catalog, SUM(count) AS sum
    ->                              FROM products
    ->                              GROUP BY id_catalog) AS sumtbl;
+-----------------+
| AVG(sumtbl.sum) |
+-----------------+
| 40.4000         |
+-----------------+
```

24.7. Вложенные запросы в операторе *CREATE TABLE*

При создании таблицы при помощи оператора `CREATE TABLE` ее можно заполнить, используя вложенный запрос, обращающийся к другой таблице.

> ***Замечание***
>
> Синтаксис оператора `CREATE TABLE` подробно рассматривается в *главе 12*.

В листинге 24.36 представлен запрос, создающий таблицу `new_orders`, в которой в отличие от таблицы `orders` вместо внешних ключей, связывающих таблицу с другими таблицами (`users`, `products`), размещены столбцы для фамилии (`surname`), имени (`name`), отчества (`patronymic`) покупателя и названия товарной позиции (`product`).

Листинг 24.36. Заполнение таблицы при создании

```
mysql> CREATE TABLE new_orders (
    -> id_order INT(11) NOT NULL AUTO_INCREMENT,
    -> user TINYTEXT,
    -> ordertime DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
    -> number INT(11) NOT NULL DEFAULT '0',
```

```
    -> product TINYTEXT,
    -> PRIMARY KEY (id_order))
    -> ENGINE=MyISAM DEFAULT CHARSET=cp1251
    -> SELECT orders.id_order AS id_order,
    ->        CONCAT(users.surname, " ", SUBSTRING(users.name,1,1), ".",
    ->               SUBSTRING(users.patronymic,1,1), ".") AS user,
    ->        orders.ordertime AS ordertime,
    ->        orders.number AS number,
    ->        products.name AS product
    -> FROM orders, users, products
    -> WHERE orders.id_user = users.id_user AND
    ->       orders.id_product = products.id_product;
mysql> SELECT * FROM new_orders;
+---+--------------+--------------------+---+-----------------------+
| id_order | user   | ordertime          | number | product           |
+---+--------------+--------------------+---+-----------------------+
| 1 | Симдянов И.В. | 2005-01-04 10:39:38 | 1 | Intel Pentium 4 3.0GHz |
| 2 | Корнеев А.А.  | 2005-02-10 09:40:29 | 2 | Gigabyte GA-8I848P-RS  |
| 3 | Иванов А.В.   | 2005-02-18 13:41:05 | 4 | Maxtor 6Y120P0         |
| 4 | Симдянов И.В. | 2005-03-10 18:20:00 | 1 | Maxtor 6Y120P0         |
| 5 | Симдянов И.В. | 2005-03-17 19:15:36 | 1 | Maxtor 6Y120P0         |
+---+--------------+--------------------+---+-----------------------+
```

Как видно из листинга 24.36, для того чтобы заполнить только что созданную таблицу `new_orders`, достаточно в конце оператора `CREATE TABLE` поместить вложенный запрос. Столбцам во вложенном запросе необходимо назначать псевдонимы при помощи оператора `AS`. Если этого не сделать, запрос может завершиться аварийно или же будут получены нежелательные имена в конечной таблице.

> ***Замечание***
>
> Данный вид запросов появился в версии 3.23, поэтому синтаксис не требует заключать вложенный запрос в обязательные круглые скобки.

Следует отметить, что столбцы, которые присутствуют в определении таблицы, но отсутствуют во вложенном запросе, получают значение по умолчанию (листинг 24.37).

Листинг 24.37. Пропуск столбца `number` во вложенном запросе

```
mysql> CREATE TABLE new_orders (
    -> id_order INT(11) NOT NULL AUTO_INCREMENT,
    -> user TINYTEXT,
    -> ordertime DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
```

```
    -> number INT(11) NOT NULL DEFAULT '0',
    -> product TINYTEXT,
    -> PRIMARY KEY (id_order))
    -> ENGINE=MyISAM DEFAULT CHARSET=cp1251
    -> SELECT orders.id_order AS id_order,
    ->        CONCAT(users.surname, " ", SUBSTRING(users.name,1,1), ".",
    ->               SUBSTRING(users.patronymic,1,1), ".") AS user,
    ->        orders.ordertime AS ordertime,
    ->        products.name AS product
    -> FROM orders, users, products
    -> WHERE orders.id_user = users.id_user AND
    ->       orders.id_product = products.id_product;
+---+--------------+---------------------+---+-----------------------+
| id_order | user   | ordertime           | number | product           |
+---+--------------+---------------------+---+-----------------------+
| 1 | Симдянов И.В. | 2005-01-04 10:39:38 | 0 | Intel Pentium 4 3.0GHz |
| 2 | Корнеев А.А.  | 2005-02-10 09:40:29 | 0 | Gigabyte GA-8I848P-RS  |
| 3 | Иванов А.В.   | 2005-02-18 13:41:05 | 0 | Maxtor 6Y120P0         |
| 4 | Симдянов И.В. | 2005-03-10 18:20:00 | 0 | Maxtor 6Y120P0         |
| 5 | Симдянов И.В. | 2005-03-17 19:15:36 | 0 | Maxtor 6Y120P0         |
+---+--------------+---------------------+---+-----------------------+
```

В листинге 24.37 в структуре таблицы `new_order` присутствует столбец `number`, однако во вложенном запросе, заполняющем только что созданную таблицу, столбец `number` пропущен. Это приводит к тому, что он получает значение по умолчанию — 0.

Важной особенностью применения вложенных запросов совместно с оператором `CREATE TABLE` является тот факт, что столбцы, не определенные в структуре оператора `CREATE TABLE`, но присутствующие в результирующей таблице вложенного запроса, добавляются в создаваемую таблицу (листинг 24.38).

Листинг 24.38. Структура результирующей таблицы определяет структуру создаваемой таблицы

```
mysql> CREATE TABLE new_orders (
    -> id_order INT(11) NOT NULL AUTO_INCREMENT,
    -> PRIMARY KEY (id_order))
    -> ENGINE=MyISAM DEFAULT CHARSET=cp1251
    -> SELECT orders.id_order AS id_order,
    ->        CONCAT(users.surname, " ", SUBSTRING(users.name,1,1), ".",
    ->               SUBSTRING(users.patronymic,1,1), ".") AS user,
```

```
    ->        orders.ordertime AS ordertime,
    ->        products.name AS product
    -> FROM orders, users, products
    -> WHERE orders.id_user = users.id_user AND
    ->       orders.id_product = products.id_product;
mysql> SELECT * FROM new_orders;
+--------+--------------+---------------------+-----------------------+
| id_order | user         | ordertime           | product               |
+--------+--------------+---------------------+-----------------------+
|      1 | Симдянов И.В. | 2005-01-04 10:39:38 | Intel Pentium 4 3.0GHz |
|      2 | Корнеев А.А.  | 2005-02-10 09:40:29 | Gigabyte GA-8I848P-RS  |
|      3 | Иванов А.В.   | 2005-02-18 13:41:05 | Maxtor 6Y120P0         |
|      4 | Симдянов И.В. | 2005-03-10 18:20:00 | Maxtor 6Y120P0         |
|      5 | Симдянов И.В. | 2005-03-17 19:15:36 | Maxtor 6Y120P0         |
+--------+--------------+---------------------+-----------------------+
```

Несмотря на то, что оператор `CREATE TABLE` определяет один-единственный столбец `id_order`, созданная таблица `new_orders` содержит четыре столбца, три из которых появились благодаря присутствию в результирующей таблице вложенного `SELECT`-запроса.

Вложенный `SELECT`-запрос можно предварять ключевыми словами `IGNORE` или `REPLACE`, чтобы указать, как следует обрабатывать записи с дублированными значениями уникальных ключей (в случае их присутствия в таблице). Если указывается ключевое слово `IGNORE`, новые записи отбрасываются при обнаружении в таблице записей с аналогичным значением уникального ключа. Если же указано ключевое слово `REPLACE`, новые записи заменяют собой существующие. Если не указано ни `IGNORE`, ни `REPLACE`, то появление записей с дублированными значениями ключей приводит к ошибке.

24.8. Вложенные запросы в операторе *INSERT*

С помощью вложенного `SELECT`-запроса оператор `INSERT` позволяет быстро вставлять множество записей из другой (других) таблицы.

> ***Замечание***
>
> Данный вид запросов появился в версии 3.23, поэтому синтаксис не требует заключать вложенный запрос в обязательные круглые скобки.

Представленные в предыдущем разделе вложенные запросы позволяли заполнять таблицу `new_orders` только в момент создания. Однако если уже после создания таблицы покупатели оформляют сделки, то информация в таблице `new_orders` теряет

свою актуальность. Выходом из ситуации будет создание таблицы `new_orders` при помощи обычного оператора `CREATE TABLE` (листинг 24.39) с последующим ее обновлением.

Листинг 24.39. Создание таблицы `new_orders`

```
mysql> CREATE TABLE new_orders (
    -> id_order INT(11) NOT NULL AUTO_INCREMENT,
    -> user TINYTEXT,
    -> ordertime DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
    -> number INT(11) NOT NULL DEFAULT '0',
    -> product TINYTEXT,
    -> PRIMARY KEY (id_order))
    -> ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Теперь, когда таблица `new_orders` создана, ее можно заполнить при помощи оператора `INSERT`, представленного в листинге 24.40.

Листинг 24.40. Использование вложенного запроса в операторе `INSERT`

```
mysql> INSERT INTO new_orders
    -> SELECT orders.id_order AS id_order,
    ->        CONCAT(users.surname, " ", SUBSTRING(users.name,1,1), ".",
    ->               SUBSTRING(users.patronymic,1,1), ".") AS user,
    ->        orders.ordertime AS ordertime,
    ->        orders.number AS number,
    ->        products.name AS product
    -> FROM orders, users, products
    -> WHERE orders.id_user = users.id_user AND
    ->       orders.id_product = products.id_product;
mysql> SELECT * FROM new_orders;
+---+--------------+---------------------+---+-------------------------+
| id_order | user   | ordertime           | number | product            |
+---+--------------+---------------------+---+-------------------------+
| 1 | Симдянов И.В. | 2005-01-04 10:39:38 | 1 | Intel Pentium 4 3.0GHz |
| 2 | Корнеев А.А.  | 2005-02-10 09:40:29 | 2 | Gigabyte GA-8I848P-RS  |
| 3 | Иванов А.В.   | 2005-02-18 13:41:05 | 4 | Maxtor 6Y120P0         |
| 4 | Симдянов И.В. | 2005-03-10 18:20:00 | 1 | Maxtor 6Y120P0         |
| 5 | Симдянов И.В. | 2005-03-17 19:15:36 | 1 | Maxtor 6Y120P0         |
+---+--------------+---------------------+---+-------------------------+
```

После того как таблица утратит свою актуальность, можно повторно выполнить запрос, добавив в условие `WHERE` ограничение по временному столбцу `ordertime`, которое отбросит записи, уже присутствующие в таблице.

Можно избежать дополнительного условия `ordertime`, воспользовавшись ключевыми словами `REPLACE` и `IGNORE`. В случае `REPLACE` все записи с дублирующим первичным ключом `id_order` будут заменены новыми. При использовании ключевого слова `IGNORE` записи в результирующей таблице вложенного запроса с первичным ключом `id_order`, которые уже присутствуют в таблице `new_orders`, будут отбрасываться (листинг 24.41).

Листинг 24.41. Использование ключевого слова `IGNORE`

```
mysql> INSERT IGNORE INTO new_orders
    -> SELECT orders.id_order AS id_order,
    ->        CONCAT(users.surname, " ", SUBSTRING(users.name,1,1), ".",
    ->               SUBSTRING(users.patronymic,1,1), ".") AS user,
    ->        orders.ordertime AS ordertime,
    ->        orders.number AS number,
    ->        products.name AS product
    -> FROM orders, users, products
    -> WHERE orders.id_user = users.id_user AND
    ->       orders.id_product = products.id_product;
```

В предложении `FROM` вложенного запроса может появляться имя таблицы, в которую производится добавление новых записей. Эта возможность появилась в СУБД MySQL, начиная с версии 4.0.14 (листинг 24.42).

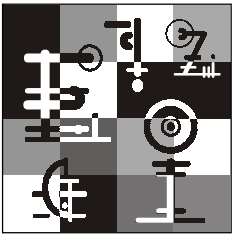
Листинг 24.42. Вставка строк из текущей таблицы

```
mysql> INSERT IGNORE INTO orders SELECT * FROM orders;
```

Если число столбцов в таблице, в которую происходит вставка новых записей, и во вложенном запросе не совпадает и часть столбцов получает значения по умолчанию, или порядок следования столбцов во вложенном запросе не совпадает с порядком следования столбцов в целевой таблице, можно воспользоваться конструкцией `VALUES` оператора `INSERT`, которая позволяет задать имена столбцов и их порядок при добавлении (листинг 24.43).

Листинг 24.43. Использование конструкции VALUES

```
mysql> INSERT INTO new_orders
    -> VALUES (id_order, ordertime, product, user)
    -> SELECT orders.id_order AS id_order,
    ->        orders.ordertime AS ordertime,
    ->        products.name AS product
    ->        CONCAT(users.surname, " ", SUBSTRING(users.name,1,1), ".",
    ->               SUBSTRING(users.patronymic,1,1), ".") AS user
    -> FROM orders, users, products
    -> WHERE orders.id_user = users.id_user AND
    ->       orders.id_product = products.id_product;
```



Глава 25

Внешние ключи и ссылочная целостность

В *главе 5* рассмотрены понятия первичного ключа и внешнего ключа, которые образуют связь между таблицами. Так, ключ `catalogs.id_catalog` является первичным ключом таблицы `catalogs`, а ключ `products.id_catalog` — внешний ключ таблицы `products`. Совокупность ключей `catalogs.id_catalog` и `products.id_catalog` образует связь между таблицами `catalogs` и `products`.

Содержимое таблицы `catalogs` представлено в листинге 25.1.

Листинг 25.1. Содержимое каталога `catalogs`

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

Пусть в связи с реорганизацией электронного магазина требуется удалить запись каталога "Жесткие диски". Для этого можно прибегнуть к оператору `DELETE` (листинг 25.2).

Листинг 25.2. Удаление каталога "Жесткие диски"

```
mysql> DELETE FROM catalogs WHERE name = 'Жесткие диски';
mysql> SELECT * FROM catalogs;
```

```
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          5 | Оперативная память |
+------------+--------------------+
```

После удаления раздела с первичным ключом `id_catalog`, равным 4, в таблице `products` остались товарные позиции, которые принадлежат удаленному разделу (листинг 25.3).

Листинг 25.3. Товарные позиции, которые не принадлежат ни одному элементу каталога

```
mysql> SELECT id_product, id_catalog, name FROM products
    -> WHERE id_catalog = 4;
+------------+------------+------------------------------+
| id_product | id_catalog | name                         |
+------------+------------+------------------------------+
|         20 |          4 | Maxtor 6Y120P0               |
|         21 |          4 | Maxtor 6B200P0               |
|         22 |          4 | Samsung SP0812C              |
|         23 |          4 | Seagate Barracuda ST3160023A |
|         24 |          4 | Seagate ST3120026A           |
+------------+------------+------------------------------+
```

Такую ситуацию называются *нарушением ссылочной целостности*. В рамках реляционной модели таблицу `catalogs` называют *предком*, а `products` — *потомком*. Всего существует четыре типа изменений базы данных, которые могут нарушить ссылочную целостность.

- Добавление новой строки-потомка. Добавление в электронный магазин товарной позиции, не принадлежащей ни одному из элементов каталога, т. е. если поле `id_catalog` таблицы `products` принимает, например, значение 104. Следует отметить, что добавление в таблицу `catalogs` строки, для которой не имеется соответствия в таблице `products`, не вызывает противоречий, т. к. существование каталога без товарных позиций вполне допустимое явление.
- Обновление внешнего ключа в строке-потомке. Это та же проблема, что и в предыдущей ситуации, но выраженная в иной форме. При обновлении поля `id_catalog` таблицы `products`, т. е. при переносе товарной позиции от одного

элемента каталога в другой необходимо, чтобы каталог с новым значением `id_catalog` существовал в таблице `catalogs`, иначе товарная позиция опять повисает в воздухе.

- Удаление строки-предка. Эта ситуация описана ранее в листингах 25.2 и 25.3. Удаление элемента каталога из таблицы `catalogs` приводит к тому, что ряд товарных позиций принадлежат к несуществующей позиции каталога.
- Обновление первичного ключа в строке-предке. Эта иная форма проблемы, рассмотренной в предыдущем пункте. При обновлении первичного ключа каталога у товарных позиций, которые относятся к данному каталогу, внешний ключ не меняется.

Средства поддержки ссылочной целостности языка SQL позволяют обрабатывать каждую из четырех описываемых ситуаций.

> ***Замечание***
>
> В СУБД MySQL ссылочная целостность поддерживается, начиная с версии 3.23.44. Однако поддержка реализована только для таблиц типа InnoDB.

Первая проблема (добавление новой строки-потомка) решается путем проверки значений в столбцах внешнего ключа перед выполнением оператора `INSERT`. Если передаваемые оператору `INSERT` значения не равны ни одному из значений первичного ключа, то возвращается ошибка добавления записи.

Вторая проблема (обновление таблицы-потомка) решается аналогично путем проверки нового значения внешнего ключа. Если такого значения среди первичных ключей таблицы-предка не обнаружено, возвращается ошибка.

Третья проблема (удаление строки-предка) может потребовать различной реакции в зависимости от контекста и предметной области. Так, применительно к электронному магазину при удалении записи каталога из таблицы `catalogs` могут быть применены следующие действия:

- не удалять элемент каталога до тех пор, пока принадлежащие ему товарные позиции из таблицы `products` не будут соотнесены с другой записью каталога;
- автоматически удалить все товарные позиции таблицы `products`, принадлежащие удаляемому элементу каталога;
- для товарных позиций, принадлежащих удаляемому элементу каталога, в поле `id_catalog` поместить значение `NULL`, которое будет сигнализировать о том, что элемент каталога для этих товарных позиций не известен;
- для товарных позиций, принадлежащих удаляемой записи каталога, в поле `id_catalog` поместить значение по умолчанию, отличное от `NULL`, например, значение `id_catalog` для элемента каталога "Разное".

Точно такие же проблемы возникают в четвертой ситуации (обновление первичного ключа в таблице-предке). Для задания всех этих действий в стандарте SQL предусмотрены специальные ключевые слова. До этого момента при определении структуры таблицы явным образом обозначался только первичный ключ. Однако для того

чтобы задать правила удаления и обновления, требуется также явное указание вторичного ключа, задаваемое 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 учебной базы данных shop внешними ключами (листинг 25.4). Следует помнить, что внешние ключи поддерживаются только таблицами типа InnoDB *(см. главу 11)*, поэтому все таблицы должны быть преобразованы к данному типу. Это можно осуществить либо при помощи оператора ALTER TABLE, либо создав таблицы заново с использованием оператора CREATE TABLE.

Листинг 25.4. Создание внешних ключей

```
mysql> CREATE TABLE catalogs (
    ->   id_catalog int(11) NOT NULL auto_increment,
    ->   name tinytext NOT NULL,
```

```
    ->   PRIMARY KEY (id_catalog)
    ->  ) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
mysql> CREATE TABLE products (
    ->   id_product int(11) NOT NULL auto_increment,
    ->   name tinytext NOT NULL,
    ->   price decimal(7,2) default '0.00',
    ->   count int(11) default '0',
    ->   mark float(4,1) NOT NULL default '0.0',
    ->   description text,
    ->   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 25.4 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 таблицы `catalogs` приводит к автоматическому обновлению таблицы `products` (листинг 25.5).

Листинг 25.5. Каскадное обновление

```
mysql> UPDATE catalogs SET id_catalog = 10 WHERE id_catalog = 4;
mysql> SELECT id_product, id_catalog, name FROM products;
+------------+------------+------------------------------+
| id_product | id_catalog | name                         |
+------------+------------+------------------------------+
|          1 |          1 | Celeron 1.8                  |
|          2 |          1 | Celeron 2.0GHz               |
|          3 |          1 | Celeron 2.4GHz               |
|          4 |          1 | Celeron D 320 2.4GHz         |
|          5 |          1 | Celeron D 325 2.53GHz        |
|          6 |          1 | Celeron D 315 2.26GHz        |
|          7 |          1 | Intel Pentium 4 3.2GHz       |
|          8 |          1 | Intel Pentium 4 3.0GHz       |
|          9 |          1 | Intel Pentium 4 3.0GHz       |
|         10 |          2 | Gigabyte GA-8I848P-RS        |
|         11 |          2 | Gigabyte GA-8IG1000          |
```

```
|          12 |           2 | Gigabyte GA-8IPE1000G          |
|          13 |           2 | Asustek P4C800-E Delux         |
|          14 |           2 | Asustek P4P800-VM\L i865G      |
|          15 |           2 | Epox EP-4PDA3I                 |
|          16 |           3 | ASUSTEK A9600XT/TD             |
|          17 |           3 | ASUSTEK V9520X                 |
|          18 |           3 | SAPPHIRE 256MB RADEON 9550     |
|          19 |           3 | GIGABYTE AGP GV-N59X128D       |
|          20 |          10 | Maxtor 6Y120P0                 |
|          21 |          10 | Maxtor 6B200P0                 |
|          22 |          10 | Samsung SP0812C                |
|          23 |          10 | Seagate Barracuda ST3160023A   |
|          24 |          10 | Seagate ST3120026A             |
|          25 |           5 | DDR-400 256MB Kingston         |
|          26 |           5 | DDR-400 256MB Hynix Original   |
|          27 |           5 | DDR-400 256MB PQI              |
|          28 |           5 | DDR-400 512MB Kingston         |
|          29 |           5 | DDR-400 512MB PQI              |
|          30 |           5 | DDR-400 512MB Hynix            |
+-------------+-------------+--------------------------------+
```

А удаление 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 будет удалено из таблицы (листинг 25.6).

Листинг 25.6. Каскадное удаление

```
mysql> DELETE FROM catalogs WHERE id_catalog > 2;
mysql> SELECT * FROM catalogs;
+------------+------------------+
| id_catalog | name             |
+------------+------------------+
|          1 | Процессоры       |
|          2 | Материнские платы |
+------------+------------------+
mysql> SELECT id_product, id_catalog, name FROM products;
+------------+------------+--------------------------+
| id_product | id_catalog | name                     |
+------------+------------+--------------------------+
|          1 |          1 | Celeron 1.8              |
```

```
|          2 |          1 | Celeron 2.0GHz            |
|          3 |          1 | Celeron 2.4GHz            |
|          4 |          1 | Celeron D 320 2.4GHz      |
|          5 |          1 | Celeron D 325 2.53GHz     |
|          6 |          1 | Celeron D 315 2.26GHz     |
|          7 |          1 | Intel Pentium 4 3.2GHz    |
|          8 |          1 | Intel Pentium 4 3.0GHz    |
|          9 |          1 | Intel Pentium 4 3.0GHz    |
|         10 |          2 | Gigabyte GA-8I848P-RS     |
|         11 |          2 | Gigabyte GA-8IG1000       |
|         12 |          2 | Gigabyte GA-8IPE1000G     |
|         13 |          2 | Asustek P4C800-E Delux    |
|         14 |          2 | Asustek P4P800-VM\L i865G |
|         15 |          2 | Epox EP-4PDA3I            |
+------------+------------+---------------------------+
```

Как видно из листинга 25.6, 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 25.7).

Листинг 25.7. Управление системной переменной `FOREIGN_KEY_CHECKS`

```
mysql> SELECT @@FOREIGN_KEY_CHECKS;
+----------------------+
| @@FOREIGN_KEY_CHECKS |
+----------------------+
|                    1 |
+----------------------+
mysql> FOREIGN_KEY_CHECKS = 1;
```

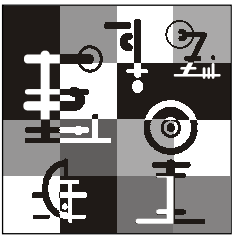
Для того чтобы извлечь текущее значение системной переменной, ее имя следует предварить двумя символами `@`. Более подробно системные переменные и синтаксис оператора `SET` обсуждается в *главе 29*.

К сожалению, внешние ключи доступны только при использовании таблиц типа InnoDB, которые уступают в скорости таблицам MyISAM. Однако каскадные удаления и обновления можно эмулировать и для других типов таблиц, используя вложенные запросы совместно с операторами `DELETE` и `UPDATE`. В листинге 25.8 представлен запрос, удаляющий из таблицы `orders` учебной базы данных `shop` все заказы, которые принадлежат покупателю с фамилией Симдянов.

Листинг 25.8. Использование вложенного запроса в операторе `DELETE`

```
mysql> SELECT id_order, surname FROM orders, users
    -> WHERE orders.id_user = users.id_user;
+----------+----------+
| id_order | surname  |
+----------+----------+
|        1 | Симдянов |
|        2 | Корнеев  |
|        3 | Иванов   |
|        4 | Симдянов |
|        5 | Симдянов |
+----------+----------+
mysql> DELETE FROM orders WHERE id_user = (SELECT id_user FROM users
    ->                                     WHERE surname = 'Симдянов');
mysql> SELECT id_order, surname FROM orders, users
    -> WHERE orders.id_user = users.id_user;
+----------+---------+
| id_order | surname |
+----------+---------+
|        2 | Корнеев |
|        3 | Иванов  |
+----------+---------+
```

Вложенный запрос возвращает первичный ключ таблицы `users` — `id_user` для покупателя Симдянова и подставляет его в условие `WHERE` внешнего запроса, по которому производится удаление из таблицы `orders`.



Глава 26

Транзакции и блокировки

Как правило, изменения базы данных, обусловленные внешними явлениями, требуют выполнения нескольких запросов. Так при покупке товара в электронном магазине `shop` требуется добавить запись в таблицу `orders` для данного заказа и уменьшить число товарных позиций на складе. В промышленных базах данных одно событие может затрагивать гораздо большее число таблиц и требовать выполнения многочисленных запросов.

Если на этапе выполнения одного из запросов происходит сбой, это может нарушить целостность базы данных. Например, товар может быть продан, а число товарных позиций на складе не обновлено; деньги могут быть сняты с одного счета, но не переведены на другой и т. п. 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.

26.1. 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 ISAM, MyISAM и HEAP не поддерживают транзакции. В настоящий момент их поддержка осуществляется только в таблицах 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.

Системы, поддерживающие возможности транзакций, часто еще характеризуют как обеспечивающие свойства ACID (ACID является сокращением Atomic (атомарный), Consistent (целостный), Isolated (изолированный), Durable (длительный)).

- Атомарность. Операторы транзакции формируют единый логический блок, каждый элемент которого невозможно выполнить без выполнения всех остальных элементов блока.
- Целостность. База данных является целостной до и после выполнения транзакции.
- Изоляция. Отдельные транзакции не влияют на работу друг друга.
- Длительность. При выполнении транзакции все ее результаты сохраняются в базе данных.

Для выяснения механизма транзакций рассмотрим ситуацию, которая возможна при работе с электронным магазином `shop`. Предположим, что два покупателя одновременно начинают покупку процессора Celeron D 320 2.4GHz, который остался на складе в одном экземпляре (листинг 26.1).

Листинг 26.1. Покупка Celeron D 320 2.4GHz

```
mysql> 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 26.1), 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` = 8 (таблица `products`). Для этого из переменной `@total` вычитается единица, и полученная цифра присваивается полю `count` товарной позиции (листинг 26.2).

Листинг 26.2. Обновление числа товарных позиций

```
mysql> 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 26.2).

После регистрации покупки вторым покупателем также происходит обновление таблицы `products` (листинг 26.2), но т. к. текущее соединение ничего не знает о запросе, параллельно выполняющемся в другом соединении, то баланс по результату двух покупок будет 1 процессор, 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.

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 26.3).

Листинг 26.3. 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 26.4).

Листинг 26.4. Изменение типа таблиц

```
mysql> ALTER TABLE catalogs ENGINE = InnoDB;
mysql> 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 модифи-

цирует таблицу, MySQL тут же сохраняет изменения на диске. Для объединения в транзакцию нескольких операторов необходимо отключить этот режим. Это можно осуществить при помощи системной переменной AUTOCOMMIT, значение которой выставляется при помощи оператора SET (листинг 26.5).

Листинг 26.5. Отключение режима автоматического завершения транзакций

```
mysql> 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 26.6). Для того чтобы включить режим автоматического завершения транзакций, необходимо выполнить оператор SET AUTOCOMMIT=1.

Листинг 26.6. Использование транзакций

```
mysql> SET AUTOCOMMIT=0;
mysql> INSERT INTO catalogs VALUES(NULL,'Переферия');
mysql> INSERT INTO catalogs VALUES(NULL,'Разное');
mysql> SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
|          6 | Периферия          |
|          7 | Разное             |
+------------+--------------------+
mysql> ROLLBACK;
mysql> SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
```

```
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
mysql> INSERT INTO catalogs VALUES(NULL,'Переферия');
mysql> INSERT INTO catalogs VALUES(NULL,'Разное');
mysql> COMMIT;
mysql> SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
|          9 | Периферия          |
|         10 | Разное             |
+------------+--------------------+
mysql> SET AUTOCOMMIT=1;
```

В листинге 26.6 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 26.7).

Листинг 26.7. Использование оператора `START TRANSACTION`

```
mysql> START TRANSACTION;
mysql> SELECT @total := count FROM products
    -> WHERE name = 'Celeron D 320 2.4GHz';
+-----------------+
| @total := count |
+-----------------+
|               1 |
+-----------------+
mysql> UPDATE products SET count = @total - 1
    -> WHERE name = 'Celeron D 320 2.4GHz';
mysql> 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 появился 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 версии MySQL 4.1.13 и 5.0.8.

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 26.8).

Листинг 26.8. Работа с именованными точками начала транзакции

```
mysql> SAVEPOINT cheops;
mysql> INSERT INTO catalogs VALUES(NULL,'Переферия');
mysql> INSERT INTO catalogs VALUES(NULL,'Разное');
mysql> ROLLBACK TO SAVEPOINT cheops;
```

В листинге 26.8 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.

26.2. 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 26.9).

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 26.9. Использование операторов `LOCK TABLES` и `UNLOCK TABLES`

```
mysql> LOCK TABLES catalogs WRITE;
mysql> INSERT INTO catalogs VALUES(NULL,'Переферия');
mysql> INSERT INTO catalogs VALUES(NULL,'Разное');
mysql> UNLOCK TABLES;
```

Листинг 26.9 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` указания 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 26.10).

Листинг 26.10. Детализация типа блокировки

```
mysql> LOCK TABLES catalogs WRITE;
mysql> INSERT INTO catalogs VALUES(NULL,'Переферия');
mysql> INSERT INTO catalogs VALUES(NULL,'Разное');
mysql> UNLOCK TABLES catalogs;
mysql> LOCK TABLES catalogs READ;
mysql> INSERT INTO catalogs VALUES(NULL,'Переферия');
mysql> INSERT INTO catalogs VALUES(NULL,'Разное');
mysql> 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 следует предусмотреть в клиентской программе (листинг 26.11).

Листинг 26.11. 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 26.12).

Листинг 26.12. Блокировка нескольких таблиц

```
mysql> LOCK TABLES catalogs WRITE, products WRITE;
mysql> SELECT * FROM catalogs, products
    -> WHERE products.id_catalog = catalogs.id_catalogs
mysql> INSERT INTO catalogs VALUES(NULL,'Разное');
mysql> 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 26.13).

Листинг 26.13. Ошибка при использовании псевдонима

```
mysql> LOCK TABLES products READ;
mysql> 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 26.14).

Листинг 26.14. 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
```

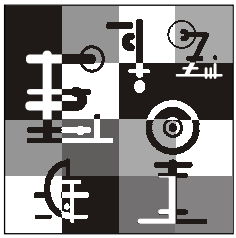
Блокировки по записи (`WRITE`) обычно имеют более высокий приоритет, чем блокировки по чтению (`READ`), чтобы гарантировать, что обновления данных пройдут как можно быстрее. Это означает, что если один или несколько клиентов устанавливают блокировку на чтение (`READ`), а затем другой клиент устанавливает 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 26.15).

Листинг 26.15. Использование ключевого слова `LOW_PRIORITY`

```
mysql> LOCK TABLES catalogs LOW_PRIORITY WRITE;
mysql> INSERT INTO catalogs VALUES(NULL,'Переферия');
mysql> 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 27

Управление учетными записями 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. Так, учетной записи редактора можно предоставить привилегии на просмотр таблицы, добавление новых записей и обновление уже существующих. Администратору базы данных можно предоставить более широкие полномочия, которые будут включать возможность создания таблиц, редактирования и удаления уже существующих. Для пользователя базы данных достаточно лишь просмотра таблиц. Данная глава посвящена созданию, удалению учетных записей, а также изменению уровня их привилегий.

27.1. Учетные записи СУБД MySQL

Следует отметить, 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 в виде `'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.

Все учетные записи хранятся в таблице `user` системной базы данных с именем `mysql`. После первой инсталляции содержимое таблицы `user` выглядит так, как это представлено в листинге 27.1.

Листинг 27.1. Содержимое таблицы `mysql.user`

```
mysql> SELECT Host, User, Password FROM mysql.user;
+---------------+------+----------+
| Host          | User | Password |
+---------------+------+----------+
| localhost     | root |          |
| %             | root |          |
+---------------+------+----------+
```

Если при инсталляции СУБД MySQL был отмечен флажок **Create An Anonymous Account** для создания анонимного пользователя, то к этим двум записям добавятся записи для анонимного пользователя — в качестве его имени выступает пустая строка (листинг 27.2).

Листинг 27.2. Анонимный пользователь и пользователь `root`

```
mysql> SELECT Host, User, Password FROM mysql.user;
+---------------+------+----------+
| Host          | User | Password |
```

```
+---------------+------+----------+
| localhost     | root |          |
| %             | root |          |
| localhost     |      |          |
| %             |      |          |
+---------------+------+----------+
```

Пользовать `root` является уникальным пользователем, обладающим всеми привилегиями, анонимный пользователь по умолчанию обладает минимальными привилегиями, проще говоря, он может только авторизоваться и выполнять простейшие запросы вроде `SELECT VERSION()`. Изменение уровня привилегий будет рассмотрено далее в *разд. 27.5*, посвященном оператору `GRANT`.

Как видно из листингов 27.1 и 27.2, по умолчанию учетные записи не защищены паролем, что позволяет не указывать его при авторизации. Однако, если к хосту, на котором установлен сервер MySQL, имеют доступ несколько человек, разумнее защитить учетные записи паролем. Это можно сделать при помощи оператора `SET PASSWORD`, который имеет следующий синтаксис:

```
SET PASSWORD = PASSWORD('pass')
SET PASSWORD FOR user = PASSWORD('pass')
```

В первой форме оператор `SET PASSWORD` устанавливает пароль для текущего пользователя. Любой клиент, подключенный к серверу с использованием неанонимной учетной записи, может изменять свой пароль.

Таким образом, пользователь `wet`, осуществив авторизацию, может поменять свой пароль при помощи запроса, представленного в листинге 27.3.

> ***Замечание***
>
> Пароль будет изменен для учетной записи с использованием того хоста, с которого авторизовался пользователь. То есть если пользователь меняет пароль на локальном хосте, это не приводит к смене пароля для сетевой учетной записи.

Листинг 27.3. Смена пароля текущим пользователем

```
mysql> SET PASSWORD = PASSWORD('Ht76W9yOWa');
```

Вторая форма оператора `SET PASSWORD` позволяет менять пароль пользователя для любой учетной записи (листинг 27.4). Только клиенты, имеющие доступ к базе данных `mysql`, имеют право использовать этот оператор. В качестве такого клиента, как правило, выступает `root`. Значение *`user`* задается в форме *`'username'@'host'`*.

Листинг 27.4. Смена пароля произвольного пользователя

```
mysql> SET PASSWORD FOR 'wet'@'%' = PASSWORD('Ht76W9yOWa');
```

Следует отметить, что для всех учетных записей пароль обрабатывается при помощи функции PASSWORD(). Это связано с тем, что MySQL не позволяет хранить пароли в незашифрованном виде. Пропуск функции PASSWORD() может привести к тому, что учетная запись будет недоступна для использования. Особенно это касается прямого редактирования таблицы user при помощи оператора UPDATE, при котором опустить вызов функции PASSWORD() очень легко (листинг 27.5).

Листинг 27.5. Прямое редактирование системной таблицы user

```
mysql> UPDATE mysql.user SET Password = PASSWORD('Ht76W9yOWa')
    -> WHERE User = 'wet' AND Host = '%';
    -> FLUSH PRIVILEGES;
```

Запросы, показанные в листингах 27.4 и 27.5, эквивалентны, но настоятельно рекомендуется использовать запрос, представленный в листинге 27.4. В листинге 27.5 указан оператор FLUSH PRIVILEGES — это связано с тем, что таблицы системной базы данных mysql хранятся в памяти, и для того чтобы изменения вступили в силу, необходимо сохранить их на жестком диске.

Из листингов 27.1 и 27.2 видно, что по умолчанию для учетных записей суперпользователя (root) и анонимного пользователя создается также сетевой вариант учетной записи (%), позволяющий получить доступ к серверу базы данных с любого хоста локальной сети. В целях безопасности рекомендуется удалить эти учетные записи при помощи операторов REVOKE и DROP USER, которые рассматриваются далее.

27.2. Оператор *CREATE USER*

Оператор CREATE USER позволяет создать новую учетную запись и имеет следующий синтаксис:

```
CREATE USER user [IDENTIFIED BY [PASSWORD] 'password']
```

Оператор создает учетную запись *user* с необязательным паролем *password*. В листинге 27.6 представлен запрос, создающий пользователя softtime.

> ***Замечание***
>
> Оператор CREATE USER появился в СУБД MySQL, начиная с версии 5.0.2.

Листинг 27.6. Создание новой учетной записи при помощи оператора CREATE USER

```
mysql> 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 27.7).

Листинг 27.7. 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 27.8).

Листинг 27.8. Защита учетной записи паролем

```
mysql> CREATE USER softtime IDENTIFIED BY '1234567';
```

> ***Замечание***
>
> На данный момент оператор `CREATE USER` поддерживает имена хостов до 60 символов. Имя пользователя не может быть длиннее 16 символов.

Пароль, заданный в листинге 27.8 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 27.9), которое сообщает, что пароль перед сохранением следует пропустить через функцию `PASSWORD()` *(см. главу 18)*.

> ***Замечание***
>
> При необратимом шифровании можно использовать только функцию `PASSWORD()`, применение других функций не предусмотрено.

Листинг 27.9. Использование ключевого слова `PASSWORD`

```
mysql> CREATE USER wet IDENTIFIED BY PASSWORD '123456';
```

При попытке создания учетной записи с уже существующим именем СУБД MySQL возвращает ошибку (листинг 27.10).

Листинг 27.10. Ошибка при создании уже существующей учетной записи

```
mysql> 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.

Один оператор CREATE USER позволяет создать сразу несколько записей. Для этого необходимо перечислить определения учетных записей через запятую, как это продемонстрировано в листинг 27.11, где создаются сразу три учетные записи.

Листинг 27.11. Создание нескольких учетных записей

```
mysql> CREATE USER wet IDENTIFIED BY PASSWORD '123456',
    ->              ret IDENTIFIED BY PASSWORD '654321',
    ->              tor IDENTIFIED BY PASSWORD '678451';
```

По умолчанию, 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 27.12).

Листинг 27.12. Создание нескольких учетных записей

```
mysql> CREATE USER 'wet'@'localhost' IDENTIFIED BY PASSWORD '123456',
    ->              'ret'@'localhost' IDENTIFIED BY PASSWORD '654321',
    ->              'tor'@'localhost' IDENTIFIED BY PASSWORD '678451';
```

27.3. Оператор *DROP USER*

Оператор DROP USER позволяет удалить учетную запись *user* и имеет следующий синтаксис:

```
DROP USER user
```

В качестве учетной записи *user* используется стандартная для MySQL запись вида *'username'@'host'* (листинг 27.13).

> ***Замечание***
>
> Оператор DROP USER появился в СУБД MySQL, начиная с версии 4.1.1.

Листинг 27.13. Использование оператора DROP USER

```
mysql> DROP USER 'wet'@'%';
```

Оператор DROP USER не закрывает автоматически 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 (листинг 27.14).

Листинг 27.14. Удаление нескольких учетных записей

```
mysql> DROP USER 'wet'@'%', 'ret'@'%', 'tor'@'%';
```

Первоначально оператор DROP USER удалял только тех пользователей, которые не обладают никакими правами, однако, начиная с версии 5.0.2, порядок работы оператора DROP USER был изменен, и он приобрел возможность удалять любого пользователя с любым уровнем привилегий. До версии 5.0.2 перед применением оператора DROP USER следовало убедиться, что пользователь не обладает никакими привилегиями. Для этого необходимо было выполнить оператор SHOW GRANTS (листинг 27.15). Если в результирующей таблице учетной записи нет, пользователь не обладает никакими правами, и его учетная запись может быть удалена.

> ***Замечание***
>
> Привилегии подробнее рассматриваются в *разд. 27.5*, посвященном оператору GRANT.

> ***Замечание***
>
> Синтаксис оператора SHOW рассматривается в *главе 30*.

Листинг 27.15. Использование оператора SHOW GRANTS

```
mysql> SHOW GRANTS;
+-----------------------------------------------------------------------+
| Grants for root@localhost                                             |
+-----------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+-----------------------------------------------------------------------+
```

Как видно из листинга 27.15, только пользователь 'root'@'localhost' обладает привилегиями, все остальные пользователи могут быть удалены при помощи оператора DROP USER (листинг 27.16).

Листинг 27.16. Текущие пользователи

```
mysql> SELECT Host, User, Password FROM mysql.user;
+-----------+----------+-------------------------------------------+
| Host      | User     | Password                                  |
```

```
+-----------+----------+-------------------------------------------+
| localhost | root     |                                           |
| %         | wet      | *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |
| localhost | softtime | *6A7A490FB9DC8C33C2B025A91737077A7E9CC5E5 |
+-----------+----------+-------------------------------------------+
```

До версии 4.1.1 удаление пользователя проводилось прямым редактированием таблицы `mysql.user` (листинг 27.17). Для этого также требовалось удалить все привилегии для удаляемого пользователя.

Листинг 27.17. Удаление пользователя прямым редактированием таблицы `user` из базы данных `mysql`

```
mysql> DELETE FROM mysql.user WHERE User = 'wet' AND Host = '%';
mysql> FLUSH PRIVILEGES;
```

27.4. Оператор *RENAME USER*

Оператор `RENAME USER` позволяет изменять имя пользователя в учетной записи и 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 27.18.

> ***Замечание***
>
> Оператор `RENAME USER` введен в СУБД MySQL, начиная с версии 5.0.2.

Листинг 27.18. Использование оператора `RENAME USER`

```
mysql> RENAME USER wet TO wet1;
mysql> SELECT Host, User, Password FROM mysql.user;
+-----------+----------+-------------------------------------------+
| Host      | User     | Password                                  |
+-----------+----------+-------------------------------------------+
| localhost | root     |                                           |
| %         | wet1     | *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |
| localhost | softtime | *6A7A490FB9DC8C33C2B025A91737077A7E9CC5E5 |
+-----------+----------+-------------------------------------------+
```

Замечание

На данный момент оператор `RENAME USER` поддерживает имена хостов до 60 символов. Имя пользователя не может быть длиннее 16 символов.

До версии 5.0.2 изменение имени пользователя в учетной записи проводилось прямым редактированием таблицы `mysql.user` (листинг 27.19).

Листинг 27.19. Изменение имени пользователя прямым редактированием

```
mysql> UPDATE mysql.user SET Uset = 'wet1'
    -> WHERE User = 'wet' AND Host = '%';
mysql> FLUSH PRIVILEGES;
```

27.5. Оператор *GRANT*

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 либо воспользоваться оператором `DROP USER`, либо удалить соответствующую запись из таблицы `mysql.user`.

В простейшем случае оператор `GRANT` выглядит так, как это представлено в листинге 27.20.

Листинг 27.20. Использование оператора `GRANT`

```
mysql> GRANT ALL ON *.* TO 'wet'@'localhost' IDENTIFIED BY 'pass';
```

Запрос в листинге 27.20 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 27.21.

Листинг 27.21. Использование оператора REVOKE

```
mysql> 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. 27.1.

***Таблица 27.1.** 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

Таблица 27.1 *(окончание)*

Привилегия	Операция, разрешенная привилегией
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
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 34)*
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 27.22.

Листинг 27.22. Предоставление привилегий для пользователя `editor`

```
mysql> 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 27.23). Это позволит избежать выполнения операций `DELETE` и `SELECT`, которыми может воспользоваться злоумышленник при помощи инъекционного кода.

Листинг 27.23. Предоставление привилегий `INSERT` и `UPDATE`

```
mysql> GRANT INSERT, UPDATE ON *.* TO program;
```

Для того чтобы назначить все привилегии сразу, следует воспользоваться запросами, представленными в листинге 27.24.

Листинг 27.24. Предоставление всех привилегий

```
mysql> GRANT ALL ON *.* TO superuser;
mysql> 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. 27.1 (листинг 27.25).

Листинг 27.25. 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. 27.1, вполне допустимо (листинг 27.26).

Листинг 27.26. Совместное использование ключевого слова `USAGE` с `SELECT`

```
mysql> GRANT USAGE, SELECT ON *.* TO superuser;
```

Следует отметить, что один оператор GRANT может быть использован для предоставления привилегий сразу нескольким пользователям (листинг 27.27).

Листинг 27.27. Предоставление привилегий сразу нескольким пользователям

```
mysql> 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. 27.2.

***Таблица 27.2.** 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 — данное предложение означает, что привилегии распространяются на таблицы базы данных *db*
ON *db*.*tbl*	Уровень таблицы — данное 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 *(см. разд. 30.13)*. Данный оператор введен в MySQL, начиная с версии 4.1.0.

Допустим, для работы с учебной базой электронного магазина требуется создать учетную запись. В этом случае можно воспользоваться запросом, представленным в листинге 27.28, где пользователю shop_editor предоставляются привилегии уровня базы данных.

Листинг 27.28. Предоставление привилегий уровня базы данных

```
mysql> GRANT ALL ON shop.* TO shop_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, который мы рассмотрим, — это привилегии на уровне таблицы. Пусть объем работ возрос настолько, что потребовалась новая учетная запись для менеджера по работе с клиентами, которому требуется только одна таблица — users. В этом случае запрос на создание такой учетной записи может выглядеть так, как это представлено в листинге 27.29.

Листинг 27.29. Предоставление привилегий уровня таблицы

```
mysql> GRANT ALL ON shop.user TO client_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 _, оба символа 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 слешем, 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 27.30.

Листинг 27.30. Символ '_' в имени базы данных

```
mysql> 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 столбца, требуется перечислить имена столбцов 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 `shop`. В этом случае запрос, создающий такого пользователя, может выглядеть так, как это представлено в листинге 27.31.

Листинг 27.31. Предоставление привилегий уровня столбца

```
mysql> GRANT SELECT(name) ON shop.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 27.32).

Листинг 27.32. Пользователь `wet` имеет право просматривать только столбец `name`

```
mysql> SELECT * FROM catalogs;
ERROR 1143: SELECT command denied to user 'wet'@'localhost' for column
'id_catalog' in table 'catalogs'
mysql> SELECT name FROM catalogs;
+--------------------+
| name               |
+--------------------+
| Процессоры         |
| Материнские платы  |
| Видеоадаптеры      |
| Жесткие диски      |
| Оперативная память |
+--------------------+
```

Как видно из листинга 27.32, 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 27.24), 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 27.33).

Листинг 27.33. Использование предложения `WITH GRANT OPTION`

```
mysql> GRANT ALL ON shop.* TO 'wet'@'localhost' WITH GRANT OPTION;
```

В результате выполнения запроса из листинга 27.33 пользователь `wet` наделяется правами вызова оператора `GRANT ALL` для предоставления привилегий другим пользователям на базу данных `shop` 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.

Ключевое слово `WITH` 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 27.34 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 27.34. Ограничение учетной записи

```
mysql> 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 27.35.

Листинг 27.35. Использование опции `REQUIRE SSL`

```
mysql> GRANT ALL ON *.* TO 'root'@'*'
    -> IDENTIFIED BY 'secret' REQUIRE SSL;
```

Требование использования сертификата представлено в листинге 27.36.

Листинг 27.36. Использование опции `REQUIRE X509`

```
mysql> 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 27.37. Важным требованием в этом случае является то, что строка *'provider'* должна быть представлена как единое целое и не может разрываться символом перевода строки.

Листинг 27.37. Использование опции `REQUIRE ISSUER 'provider'`

```
mysql> 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 27.38. Здесь, так же как и в предыдущем примере, не допускается разрыв символом перевода строки.

Листинг 27.38. Использование опции `REQUIRE SUBJECT 'subject'`

```
mysql> 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 27.39.

Листинг 27.39. Использование опции `REQUIRE CIPHER 'cod'`

```
mysql> 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 27.40.

Листинг 27.40. Комбинирование опций `SUBJECT`, `ISSURE` и `CIPHER`

```
mysql> 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`.

27.6. Оператор *REVOKE*

Для отмены привилегий учетной записи предназначен оператор `REVOKE` (листинг 27.41). 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 27.41. Использование оператора `REVOKE`

```
mysql> 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 либо оператором `DROP USER`, либо удалить пользователя из таблицы `mysql.user` при помощи оператора `DELETE`.

До версии 4.1.2 нельзя было удалить сразу все привилегии, необходимо было выполнить два оператора (листинг 27.42).

Листинг 27.42. Удаление всех привилегий

```
mysql> REVOKE ALL ON shop.* FROM 'wet'@'localhost';
mysql> 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 27.43).

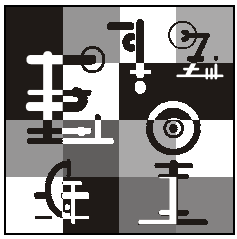
Листинг 27.43. Альтернативный синтаксис удаления всех привилегий

```
mysql> REVOKE ALL, GRANT OPTION ON shop.* FROM 'wet'@'localhost';
```

Синтаксис оператора `REVOKE` позволяет удалять привилегии сразу для нескольких пользователей (листинг 27.44).

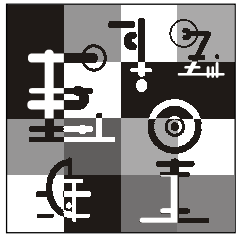
Листинг 27.44. Альтернативный синтаксис удаления всех привилегий

```
mysql> REVOKE ALL, GRANT OPTION ON shop.*
    -> FROM 'wet'@'localhost',
    ->      'toor'@'localhost',
    ->      'ret'@'localhost';
```



ЧАСТЬ III

СРЕДСТВА АДМИНИСТРИРОВАНИЯ СУБД MYSQL



Глава 28

Администрирование СУБД MySQL

Сервер MySQL является сложным программным продуктом, который должен работать одинаково эффективно в совершенно разных условиях и решать задачи различного характера. Поэтому сервер имеет гибкие настройки, которые позволяют сконфигурировать его под свои нужды без перекомпиляции исходного кода.

Сервер MySQL имеет два главных способа настройки: это параметры запуска, которые определяют режимы работы сервера, и переменные состояния, с помощью которых можно масштабировать сервер. В данной главе подробно рассматриваются как параметры запуска, так и переменные состояния сервера MySQL.

28.1. Параметры запуска сервера MySQL

При запуске сервера `mysqld` (mysqld-nt.exe для Windows) можно передавать параметры либо в командной строке, либо в одном из конфигурационных файлов my.ini и my.cnf. Конфигурационные файлы содержат разделы, которые относятся к разным частям, например, раздел `[mysqld]` относится к серверу MySQL, а раздел `[client]` — к консольному клиенту `mysql`.

> ***Замечание***
>
> Можно указать версию сервера, например, `[mysqld-5.0]` и `[mysqld-5.1]` задают опции для разных версий серверов MySQL.

MySQL-сервер считывает параметры из разделов `[mysqld]` и `[server]` конфигурационных файлов my.ini или my.cnf. MySQL-сервер принимает множество параметров командной строки. Чтобы просмотреть их, необходимо вывести их список при помощи команды, представленной в листинге 28.1.

> ***Замечание***
>
> В операционной системе Windows MySQL-сервер называется mysqld-nt.exe.

Листинг 28.1. Просмотр параметров MySQL-сервера

```
mysqld --verbose --help
```

До версии 4.1.1 для вывода полного списка команд достаточно команды

```
mysql --help
```

Однако в более поздних версиях такой вариант команды приводит лишь к выводу краткой справки.

Параметры в утилитах MySQL могут иметь две формы: полную, начинающуюся с двух дефисов `--user`, и краткую, которая начинается с одного дефиса `-u`. Можно применять оба варианта, но для ряда параметров имеется только полная форма. Далее приводится список основных параметров сервера.

- `--help`, `-?`. Запуск сервера с данным параметром приводит к выводу краткой справочной информации, после чего сервер завершает свою работу.
- `--ansi`. Данный параметр сообщает серверу о необходимости руководствоваться стандартом ANSI для синтаксиса SQL, а не синтаксисом MySQL. Параметр предназначен для совместимости со стандартами ANSI. Данный параметр эквивалентен использованию параметра `--sql-mode` с флагами `REAL_AS_FLOAT`, `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE`, `SERIALIZE` и `ONLY_FULL_GROUP_BY`.
- `--basedir=`*`path`*, `-b` *`path`*. Параметр задает путь к каталогу инсталляции MySQL, все остальные пути обычно определяются относительно данного пути.
- `--bind-address=`*`IP`*. Данный параметр позволяет указать IP-адрес, к которому должен привязываться MySQL-сервер. Обычно MySQL-сервер привязывается к IP-адресу хоста, на котором запущен сервер. С помощью этого параметра можно выбрать альтернативный адрес, если используемый хост имеет сразу несколько адресов.
- `--bootstrap`. Этот параметр используется скриптом `mysql_install_db` для создания таблиц привилегий без полноценного запуска MySQL-сервера.
- `--character-sets-dir=`*`path`*. Параметр указывает имя каталога *`path`*, в котором хранятся файлы кодировок.
- `--character-set-client-handshake`. Данный параметр требует от сервера не игнорировать информацию о кодировках и сортировках, которые присылает клиент при помощи SET-запросов. Для того чтобы сервер игнорировал SET-запросы клиента на установку кодировок и сортировок, необходимо использовать параметр `--skip-character-set-client-handshake`. Последний параметр предназначен для эмуляции работы MySQL-сервера версии 4.0 на серверах более высоких версий.
- `--character-set-filesystem=`*`charset_name`*. Данный параметр позволяет задать кодировку, используемую для именования файлов в файловой системе. Введен, начиная с версии 5.1.6.

- `--character-set-server=`*`charset_name`*, `-C` *`charset_name`*. Параметр задает кодировку *`charset_name`* для всего MySQL-сервера.
- `--chroot=`*`path`*. Параметр указывает путь *`path`* для помещения MySQL-сервера в chroot-окружение.
- `--collation-server=`*`collation_name`*. Параметр задает сортировку *`collation_name`* для всего MySQL-сервера.
- `--console`. Данный параметр используется для операционной системы Windows. Если его указать при старте сервера, окно консоли не будет закрыто, и можно будет изучить ошибки, которые выводятся в консольное окно при запуске сервера.
- `--core-file`. Требует создание дампа памяти при крахе MySQL-сервера.
- `--datadir=`*`path`*, `-h` *`path`*. Параметр позволяет задать путь *`path`* к каталогу данных, в котором располагаются базы данных. Этот параметр удобен, если необходимо вынести каталог данных на отдельный раздел.
- `--debug[=`*`debug_options`*`]`, `-# [`*`debug_options`*`]`. Данный параметр требует записи в журнал отладочной информации. Если значение *`debug_options`* не задается, информация выводится в формате `'d:t:o,file_name'`. Для работы параметра необходимо, чтобы СУБД MySQL была скомпилирована с отладочным выводом.
- `--default-storage-engine=`*`type`*. Данный параметр устанавливает тип таблицы *`type`* по умолчанию. Введен в MySQL, начиная с версии 4.1.2, и является синонимом для параметра `--default-table-type`. Подробнее с типами таблиц можно ознакомиться в *главе 11*.
- `--default-table-type=`*`type`*. Данный параметр является синонимом для `--default-storage-engine`.
- `--default-time-zone=`*`timezone`*. Устанавливает часовой пояс *`timezone`* для сервера. Если параметр не устанавливается, MySQL-сервер извлекает значение временной зоны из системной переменной `system_time_zone`.
- `--delay-key-write[=OFF|ON|ALL]`. Параметр устанавливает режим задержки записи ключей для таблиц типа MyISAM при использовании конструкции `DELAYED KEY` оператора `INSERT`. Если параметру передается значение `OFF`, MySQL-сервер игнорирует конструкцию `DELAYED KEY`, без задержки ключей. Если в качестве значения передается `ON`, запись ключей с задержкой осуществляется для тех операторов `INSERT`, в которых явно указывается конструкция `DELAYED KEY`. Если в качестве значения параметра передается значение `ALL`, активируется задержка для всех операторов `INSERT` не зависимо от того, используется конструкция `DELAYED KEY` или нет.
- `--des-key-file=`*`file_name`*. Данный параметр указывает путь к файлу с ключами *`file_name`*, используемому функциями `DES_ENCRYPT()` и `DES_DECRYPT()`.
- `--enable-named-pipe`. Данный параметр активирует поддержку именованных каналов. Параметр доступен только в Windows NT/2000/XP/Server 2003.

- `--event-scheduler`. Параметр включает планировщик заданий, доступный, начиная с версии 5.1.6.
- `--exit-info[=flags]`, `-T [flags]`. Данный параметр позволяет передать битовую маску с флагами. Используется для отладки сервера `mysqld`.
- `--external-locking`. Параметр разрешает внешнюю блокировку доступа к таблицам. Данная опция может приводить к зависанию `mysqld` в операционной системе Linux. Раньше параметр назывался `--enable-locking`.
- `--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.
- `--init-file=file`. При указании данного параметра сервер считывает файл `file` при запуске сервера. Каждый оператор должен занимать одну строку и не содержать комментарии.
- `--language=lang_name`, `-L lang_name`. Данный параметр позволяет задать язык `lang_name`, на котором будут выводиться сообщения об ошибках клиенту. Пример: `--language=russian`.
- `--large-pages`. Некоторые аппаратные и операционные средства поддерживают страницы памяти более 4 Кбайт. Данный параметр позволяет включить поддержку страниц памяти более 4 Кбайт. Этот параметр был введен, начиная с версии 5.0.3.
- `--log[=file_name]`, `-l [file_name]`. При использовании данного параметра, соединения и SQL-запросы регистрируются в журнальном файле `file_name`. Если файл не указывается, то в качестве имени файла будет использоваться имя хоста, на котором расположен MySQL-сервер — *hostname*.log. Более подробно журнальные файлы рассматриваются в *разд. 28.4*.
- `--log-bin=[base_name]`. Данный параметр позволяет задать имя файла бинарного журнала `base_name`. Этот файл используется для резервного копирования и репликации. Если файл не указывается, то в качестве его имени будет использоваться имя хоста, на котором расположен MySQL-сервер — *hostname*-bin. Более подробно журнальные файлы рассматриваются в *разд. 28.4.4*.
- `--log-bin-index[=file_name]`. Данный параметр позволяет указать индексный файл `file_name` для бинарного журнала запросов. Если имя файла не задано, то оно будет построено на основании имени хоста — *hostname*-bin.index. Более подробно журнальные файлы рассматриваются в *разд. 28.4.4*.
- `--log-bin-trust-function-creators[={0|1}]`. Параметр позволяет задать значение системной переменной `log_bin_trust_function_creators`. Если параметр не используется или используется со значением 1, системная переменная `log_bin_trust_function_creators` получает значение 1. Подробнее системные переменные обсуждаются в *разд. 28.2*. Данный параметр был введен, начиная с версии MySQL 5.0.16. Более подробно журнальные файлы рассматриваются в *разд. 28.4*.

- `--log-error[=file_name]`. Данный параметр позволяет указать имя файла `file_name` для записи сообщений об ошибках и времени запуска MySQL-сервера. Если имя файла не задано, то оно будет построено на основании имени хоста — *hostname*.err. Более подробно журнальные файлы рассматриваются в *разд. 28.4*.
- `--log-isam[=file_name]`. Данный параметр позволяет задать имя файла индекса `file_name` для регистрации изменений в таблицах MyISAM (используется для отладки). Если имя файла `file_name` не указывается, используется файл с именем myisam.log. Более подробно журнальные файлы рассматриваются в *разд. 28.4*.
- `--log-output[=value,...]`. Данный параметр позволяет определить, куда будет выводиться информация журнала запросов и журнала медленных запросов. В качестве пункта назначения может выступать как текстовый файл, так и таблица системной базы данных `mysql`. Допускается вывод и в файл, и в таблицу. Параметр введен, начиная с версии MySQL 5.1.6. Более подробно данный параметр рассматривается в *разд. 28.4.1*.
- `--log-queries-not-using-indexes`. Если вместе с данной опцией применяется параметр `--log-slow-queries`, тогда запросы, не использующие индексов, также будут регистрироваться в журнале медленных запросов. Данный параметр был введен в MySQL 4.1.0.
- `--log-short-format`. При использовании данного параметра в журнальные файлы записывается минимальное количество данных. Данный параметр был введен в MySQL 4.1.0.
- `--log-slow-admin-statements`. При использовании данного параметра в журнальные файлы помещаются записи об использовании медленных административных команд, таких как `OPTIMIZE TABLE`, `ANALYZE TABLE` и `ALTER TABLE`.
- `--log-slow-queries[=file_name]`. Данный параметр предписывает регистрировать в журнале все запросы, выполнение которых в секундах заняло больше времени, чем указано в системной переменной `long_query_time`. Если имя файла не задано, то оно будет построено на основании имени хоста — *hostname*-slow.log.
- `--log-warnings=[level]`, `-W [level]`. При указании данного параметра в журнальные файлы записываются некритические диагностические сообщения. Данный параметр был введен, начиная с версии 4.1.2. По умолчанию режим активирован, чтобы его отключить, необходимо использовать параметр `--skip-log-warnings`.
- `--low-priority-updates`. При использовании данного параметра для операций изменения в таблице (`INSERT`/`DELETE`/`UPDATE`) устанавливается меньший приоритет, чем для операций выборки (`SELECT`).
- `--memlock`. При использовании данного параметра сервер MySQL остается в оперативной памяти. Этот параметр применяется в операционной системе Solaris, которая поддерживает системный вызов `mlockall()`. Это может оказаться полезным в случае возникновения проблемы, когда операционная система сбрасывает

сервер MySQL на диск во время подкачки. Однако для запуска параметра с этой опцией придется запускать MySQL-сервер от имени суперпользователя (`root`).

- `--myisam-recover[=option[,option]...]]`. Данный параметр позволяет установить режим восстановления таблиц MyISAM. В качестве значения можно указать любую комбинацию значений: `DEFAULT`, `BACKUP`, `FORCE` или `QUICK`. Если задается сразу несколько значений, их следует разделять запятыми. Более подробно режимы восстановления обсуждаются в *главе 31*.
- `--old-passwords`. Начиная с версии 4.1, MySQL-сервер использует новый метод шифрования паролей для учетных записей. Существующие учетные записи, пароли которых зашифрованы с применением старого метода, по-прежнему поддерживаются, но новые пароли шифруются с помощью нового метода. При использовании параметра `--old-passwords` будет применяться старый метод шифрования паролей при создании новых учетных записей.
- `--one-thread`. Данный параметр требует от MySQL-сервера запускаться с одним потоком. Подобный запуск используется для отладки в операционной системе Linux, в которой для MySQL-сервера обычно запускаются, как минимум, три потока.
- `--open-files-limit=count`. Данный параметр позволяет ограничить количество используемых MySQL-сервером файловых дескрипторов *count*. Если параметр не задан или установлен со значением 0, MySQL-сервер будет резервировать `max_connections` × 5 или `max_connections` + `table_cache` × 2 файлов, выбирая больше значение. Здесь `max_connections` и `table_cache` — системные переменные. Параметр `--open-files-limit` необходимо использовать в том случае, если сервер выдает ошибку "Too many open files" ("Слишком много открытых файлов").
- `--pid-file=path`. Данный параметр предназначен для указания PID-файла, в котором хранится идентификатор главного процесса MySQL-сервера.
- `--port=port_num`, `-P port_num`. Параметр указывает номер порта *port_num*, по которому MySQL-сервер ожидает соединения с клиентом (по умолчанию 3306, если этот параметр не указан, производится попытка установить соединение именно по данному порту).
- `--port-open-timeout=num`. Данный параметр задает время ожидания соединения с портом TCP/IP в секундах *num*. Введен в MySQL, начиная с версии 5.1.5.
- `--safe-mode`. Данный параметр предписывает пропускать некоторые шаги оптимизации. Параметр используется в тех случаях, когда MySQL работает нестабильно или запуск сложных запросов приводит к выводу некорректных результатов.
- `--safe-user-create`. При использовании данного параметра пользователь не может создавать новых пользователей с помощью оператора `GRANT`, если он не имеет привилегии `INSERT` для таблицы `mysql.user` или для любого ее столбца, даже если ему передана привилегия `GRANT OPTION`.

- `--secure-auth`. Данный параметр запрещает аутентификацию для учетных записей, которые имеют старые пароли (созданные в MySQL версии меньше 4.1). Параметр доступен, начиная с версии MySQL 4.1.1.
- `--shared-memory`. Данный параметр позволяет осуществлять соединение с локальными клиентами через общую область памяти. Параметр доступен только в операционной системе Windows.
- `--shared-memory-base-name=`*`name`*. Данный параметр позволяет осуществить соединение с локальными клиентами через именованную область памяти *`name`*. По умолчанию в качестве идентификатора именованной области выступает `MYSQL` (имя зависит от регистра). Параметр доступен только в операционной системе Windows.
- `--skip-bdb`. Данный параметр позволяет отключить тип таблиц BDB. Это обеспечивает экономию памяти и увеличение скорости выполнения некоторых операций.
- `--skip-concurrent-insert`. Параметр отключает возможность одновременного выбора и вставки в таблицах MyISAM. Не рекомендуется использовать данный параметр.
- `--skip-external-locking`. Параметр предписывает MySQL-серверу не использовать системную блокировку.
- `--skip-grant-tables`. При установке данного параметра сервер игнорирует системную таблицу привилегий. Это предоставляет каждому полный доступ ко всем базам данных.
- `--skip-host-cache`. Параметр требует не использовать внутренний кэш имен хостов для более быстрого преобразования имен в IP-адреса. Вместо этого MySQL-сервер будет запрашивать DNS-сервер каждый раз, когда пользователь устанавливает соединение.
- `--skip-innodb`. Параметр позволяет отключить таблицы InnoDB. Это экономит память и может повысить скорость выполнения некоторых операций.
- `--skip-name-resolve`. Во время проверки соединений клиентов не будет выполняться преобразование имен хостов — будут использоваться только IP-адреса.
- `--skip-networking`. Данный параметр запрещает MySQL-серверу прослушивать TCP/IP-соединения. Взаимосвязь клиента с MySQL-сервером должна осуществляться через именованные каналы (в Windows) или сокеты (UNIX). Этот параметр особенно полезен, если планируется предоставлять доступ к MySQL-серверу только для локальных клиентов.
- `--standalone`. Данный параметр применяется в операционной системе Windows для того, чтобы игнорировать запуск MySQL-сервера через сервисы. Без использования этого параметра запустить MySQL-сервер не удастся.
- `--symbolic-links`, `--skip-symbolic-links`. Данные параметры включают или отключают поддержку символических ссылок.
- `--skip-safemalloc`. Если MySQL-сервер откомпилирован с параметром `--with-debug=full`, все MySQL-программы осуществляют проверку выхода

за границы массивов при каждой операции загрузки или освобождения памяти, что позволяет значительно увеличить надежность сервера. Процесс проверки происходит достаточно медленно, поэтому если такое поведение необходимо отключить, можно воспользоваться параметром `--skip-safemalloc`.

- `--skip-show-database`. Если MySQL-сервер запускается с данным параметром, то оператор `SHOW DATABASES` доступен только пользователям, имеющим привилегию `SHOW DATABASES`, и будет отображать имена всех баз данных. Без этого параметра `SHOW DATABASES` доступен всем пользователям, но отображает имя каждой базы данных, только если пользователь имеет привилегию `SHOW DATABASES` или какую-нибудь другую привилегию для базы данных.
- `--skip-stack-trace`. Данные трассировки стека записываться не будут. Этот параметр применяется при запуске MySQL-сервера в отладчике.
- `--skip-thread-priority`. Параметр отключает использование приоритетов потока с целью сократить время отклика MySQL-сервера.
- `--socket=`*`path`*. В UNIX данный параметр определяет файл сокета UNIX, который будет использоваться для локальных соединений. По умолчанию установлено значение `/tmp/mysql.sock`. В Windows этот параметр задает имя канала для локальных соединений, использующих именованный канал (по умолчанию имеет значение `MySQL`).
- `--sql-mode='`*`value`*`[,`*`value`*`[,`*`value`*`...]]'`. Устанавливает SQL-режим для MySQL. Более подробно SQL-режимы описываются в *разд. 28.3*.
- `--temp-pool`. Применение данного параметра позволяет решить проблему с утечкой памяти, возникающей в старых версиях Linux в результате создания большого количества новых файлов с разными именами.
- `--transaction-isolation=`*`level`*. Устанавливает уровень изоляции транзакции по умолчанию: `READ-UNCOMMITTED`, `READ-COMMITTED`, `REPEATABLE-READ` и `SERIALIZBLE`.
- `--tmpdir=`*`path`*, `-t` *`path`*. Параметр устанавливает путь к каталогу, который будет использоваться для создания временных файлов. Этот параметр может пригодиться, если каталог по умолчанию /tmp находится в слишком маленьком разделе диска. Начиная с версии MySQL 4.1, этот параметр может принимать несколько путей, которые используются по циклическому принципу. Пути должны разделяться символом двоеточия (`:`) в UNIX и точкой с запятой (`;`) в операционных системах Windows, NetWare и OS/2.
- `--user={`*`user_name`*`|`*`user_id`*`}`, `-u {`*`user_name`*`|`*`user_id`*`}`. Параметр `--user` позволяет задать пользователя *`user_name`* или его идентификатор *`user_id`*, по имени которого будет запущен MySQL-сервер. Данный параметр обязателен при запуске MySQL-сервера от имени суперпользователя (`root`).
- `--version`, `-V`. При использовании данного параметра отображается версия MySQL-сервера, после чего он прекращает работу.

Параметры сервера необязательно передавать во время запуска — их можно прописать в конфигурационном файле my.ini или my.cnf в разделе `[mysqld]`. В этом случае MySQL-сервер самостоятельно прочитает их из конфигурационного файла. Однако в конфигурационном файле параметры должны быть записаны в полном формате и без предваряющих символов тире. Например, для того чтобы перенести каталог данных из каталога C:\mysql5\Data\, скажем, в каталог D:\databases\, необходимо в конфигурационном файле my.ini определить параметр `datadir` (листинг 28.2).

> ***Замечание***
>
> Обратите внимание, что при запуске из командной строки для переопределения каталога данных необходимо было бы воспользоваться параметром `--datadir="C:/mysql/Data/"`.

Листинг 28.2. Изменение каталога данных в конфигурационном файле my.ini

```
[mysqld]
datadir="C:/mysql/Data/"
```

28.2. Системные переменные сервера

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 помощи оператора `SHOW VARIABLES`, который рассматривается в *разд. 30.18*.

В листинге 28.3 приводится частичный вывод оператора `SHOW VARIABLES`.

Листинг 28.3. Использование оператора `SHOW VARIABLES`

```
mysql> 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 28.4).

Листинг 28.4. 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.

> ***Замечание***
>
> Оператор `SET` обсуждается более подробно в *главе 29*. Там же можно найти список переменных, которые можно изменить с его помощью.

> ***Замечание***
>
> Системные переменные часто изменяются, часть переменных может исчезнуть в новых версиях, в то же время могут появиться новые системные переменные.

Рассмотрим значения системных переменных более подробно.

- `auto_increment_increment`. Системная переменная предназначена для установки приращения в механизме `AUTO_INCREMENT` *(см. разд. 6.1)*. 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 значение переменной, например, в 10 приводит к тому, что значение автоинкрементного счетчика увеличивается не на единицу, а сразу на 10 (листинг 28.5).

  > ***Замечание***
  >
  > Системная переменная `auto_increment_increment` введена в MySQL, начиная с версии 5.0.2.

Листинг 28.5. Влияние системной переменной `auto_increment_increment` на механизм `AUTO_INCREMENT`

```
mysql> SHOW VARIABLES LIKE 'auto_increment_increment';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 1     |
+--------------------------+-------+
mysql> CREATE TABLE autoinc1
    -> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
mysql> SET @@auto_increment_increment=10;
mysql> SHOW VARIABLES LIKE 'auto_increment_increment';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
+--------------------------+-------+
mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
mysql> 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. 6.1)*. 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 значение переменной, например, в 5 приводит к тому, что значение автоинкрементного счетчика в новой таблице начинается не с 1, а с 5 (листинг 28.6).

  Замечание

  Системная переменная `auto_increment_offset` введена в MySQL, начиная с версии 5.0.2.

Листинг 28.6. Влияние системной переменной `auto_increment_offset` на механизм `AUTO_INCREMENT`

```
mysql> SET @@auto_increment_offset=5;
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
mysql> CREATE TABLE autoinc2
    -> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
mysql> INSERT INTO autoinc2 VALUES (NULL), (NULL), (NULL), (NULL);
mysql> 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` *(см. разд. 28.1)*.

- 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 определяет 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 --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.

- `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 *(см. главу 14)*. 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 *(см. главу 14)*. 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 26*. 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. 16.47)*. Функция `WEEK()` может принимать в качестве второго необязательного параметра режим из табл. 16.7. Переменная позволяет задать номер режима, 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), то ключевое словао `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 используется 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 28.7).

Замечание

Переменная div_precision_increment была введена, начиная с версии 5.0.6.

Листинг 28.7. Использование переменной div_precision_increment

```
mysql> SELECT 1/7;
+--------+
| 1/7    |
+--------+
| 0.1429 |
+--------+
mysql> SET div_precision_increment = 30;
mysql> 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. 28.4*.
- `flush`. Переменная принимает значение `ON`, если MySQL-сервер запущен с параметром `--flush` *(см. разд. 28.1)*. 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 20)*. Значение переменной по умолчанию соответствует строке `'+ -><()~*:""&|'`. Для изменения значения применяются следующие правила:
  - функция операции определяется позицией в строке;
  - значения 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.
- 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_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 против-

ном случае переменная принимает значение NO. Переменная введена, начиная с версии MySQL 4.1.3. Пространственные расширения подробно обсуждаются в *приложении 1*.

- 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. Репликация подробно обсуждается в *главе 32*. 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, по умолчанию для каждого клиента включен режим автоматической фиксации транзакций. Глобальная переменная сервера, через которую указывалось бы отключение режима фиксации, отсутствует, но добиться такого эффекта можно с помощью init_connect: SET GLOBAL init_connect='SET AUTOCOMMIT=0'. Переменная init_connect также может быть установлена в командной строке или в конфигурационных файлах my.ini или my.cnf (листинг 28.8).

Замечание

Переменная init_connect введена, начиная с версии MySQL 4.1.2.

Листинг 28.8. Использование переменной `init_connect`

```
[mysqld]
init_connect='SET AUTOCOMMIT=0'
```

- `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. Подробнее репликация обсуждается в *главе 32*. 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.
- `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, пользователям 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 33*.
- `log_error`. Переменная содержит путь к журнальному файлу ошибок.
- `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. Подробнее репликация обсуждается в *главе 32*.
- `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` *(см. разд. 30.9 и 30.19)*. 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_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.
- `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).
- `myisam_max_sort_file_size`. Переменная задает размер временного файла, который используется для воссоздания индекса MyISAM (при использовании опе-

раторов 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.
- 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 привилегий 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.
- `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, подвергаются 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`.
- `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 текущий режим SQL-сервера. Более подробно с режимами SQL-сервера можно ознакомиться в *разд. 28.3*.
- `sql_slave_skip_counter`. Переменная определяет количество событий главного сервера, которые подчиненный сервер репликации должен проигнорировать.
- `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 увеличения производительности сервера MySQL. Переменная введена, начиная с версии MySQL 5.1.3.
- `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 кэша 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 отсутствует `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.

28.3. Режим SQL-сервера

MySQL-сервер может работать в нескольких режимах SQL, которые определяют, какой SQL-синтаксис должен поддерживаться и какой тип проверки корректности данных будет выполняться. Это позволяет добиться частичной эмуляции поведения других серверов и легче адаптировать MySQL к производственным условиям. Установить режим SQL по умолчанию можно, запустив MySQL-сервер с параметром `--sql-mode="`*`modes`*`"` *(см. разд. 28.1)*. Кроме этого режим можно установить при помощи оператора `SET` *(см. главу 29)*: `SET sql_mode='`*`modes`*`'`. Здесь *modes* — это список различных режимов, разделяемых запятой (`,`). По умолчанию ни один из режимов не установлен, и список пуст. Далее представлены режимы, которые поддерживаются сервером MySQL.

- `ALLOW_INVALID_DATES`. В данном режиме дата, вставляемая в таблицу базы данных, не проверяется на существование. Вместо этого осуществляется проверка, чтобы число месяца располагалось в интервале от 1 до 31, а месяц — от 1 до 12. Такой режим особенно удобен в Web-приложениях, когда под год, месяц и число отводятся три выпадающих списка, никак не связанных друг с другом. Этот режим распространяется только на типы данных `DATE` и `DATETIME`, не затрагивая `TIMESTAMP`, для которого по-прежнему необходима полностью корректная дата.
- `ANSI_QUOTES`. В данном режиме двойные кавычки (") трактуются как обратные кавычки (`` ` ``). В этом режиме нельзя использовать двойные кавычки для обрамления строк — только одинарные.

- `ERROR_FOR_DIVISION_BY_ZERO`. В данном режиме генерируется ошибка (или предупреждение, если не включен строгий режим) при делении числа на ноль во время выполнения операторов `INSERT` и `UPDATE`. В обычном режиме в качестве результата операции деления на ноль возвращается значение `NULL`. Данный режим появился, начиная с версии MySQL 5.0.2.
- `HIGH_NOT_PRECEDENCE`. Данный режим влияет на приоритет оператора `NOT`. Так в обычном режиме конструкция `NOT` *a* `BETWEEN` *b* `AND` *c* эквивалентна конструкции `NOT (`*a* `BETWEEN` *b* `AND` *c*`)`. Если включен режим `HIGH_NOT_PRECEDENCE`, вышеприведенная конструкция эквивалентна (`NOT a) BETWEEN` *b* `AND` *c*. Особенность режима демонстрируется в листинге 28.9: в обоих примерах в первом случае значение 1 попадает в интервал (–5, 5) и оператор `BETWEEN` возвращает истину (1), которая под воздействием оператора `NOT` превращается в ложь (0); во втором случае оператор `NOT` применяется к единице, возвращая 0, который тоже входит в интервал (–5, 5), и `BETWEEN` возвращает 1. Данный режим появился, начиная с версии MySQL 5.0.2.

Листинг 28.9. Особенность режима `HIGH_NOT_PRECEDENCE`

```
mysql> SET sql_mode = '';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
+-----------------------+
| NOT 1 BETWEEN -5 AND 5 |
+-----------------------+
|                     0 |
+-----------------------+
mysql> SET sql_mode = 'HIGH_NOT_PRECEDENCE';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
+-----------------------+
| NOT 1 BETWEEN -5 AND 5 |
+-----------------------+
|                     1 |
+-----------------------+
```

- `IGNORE_SPACE`. В главах, посвященных встроенным функциям MySQL, много раз упоминалось, что между именем функции и открывающей круглой скобкой `(` не должно быть пробела. В данном режиме такие пробелы допускаются, однако все имена таблиц и столбцов, которые совпадают с именами функций, становятся зарезервированными словами и их имена должны заключаться в кавычки. Например, при использовании функции `USER()`, имя таблицы `user` в базе данных `mysql` и имя столбца `User` в этой таблице становятся зарезервированными словами, которые должны быть заключены в кавычки:

```
SELECT 'User' FROM mysql.'user';
```

- NO_AUTO_CREATE_USER. При использовании данного режима новые пользователи не будут создаваться при помощи оператора GRANT *(см. главу 27)* до тех пор, пока не будет использоваться непустой пароль. Данный режим появился, начиная с версии MySQL 5.0.2.
- NO_AUTO_VALUE_ON_ZERO. Режим NO_AUTO_VALUE_ON_ZERO влияет на обработку столбцов, снабженных атрибутом AUTO_INCREMENT. Как описывалось в *главе 6*, для того чтобы при вставке такой столбец получил следующий порядковый номер, достаточно указать в качестве его значения либо NULL, либо 0. В режиме NO_AUTO_VALUE_ON_ZERO при использовании значения 0 механизм AUTO_INCREMENT срабатывать не будет, и в таблицу будет помещено значение 0.
- NO_BACKSLASH_ESCAPES. В данном режиме отключается возможность использования обратных слешей \ для экранирования символов в строке. Обратный слеш воспринимается как обычный символ, наряду со всеми остальными. Режим появился, начиная с версии MySQL 5.0.1.
- NO_DIR_IN_CREATE. В данном режиме во время создания таблиц игнорируются все директивы INDEX DERECTORY и DATA DIRECTORY. Этот режим удобен на подчиненных серверах репликации *(см. главу 32)*.
- NO_ENGINE_SUBSTITUTION. При данном режиме предотвращается использование типа таблиц по умолчанию, когда осуществляется попытка создания таблиц с указанием типа, который либо отключен, либо вообще не включен в дистрибутив MySQL. Данный режим появился, начиная с версии MySQL 5.0.8.
- NO_FIELD_OPTIONS. При использовании данного режима опции столбцов не отображаются в результирующей таблице оператора SHOW CREATE TABLE. Режим появился, начиная с версии MySQL 4.1.1.
- NO_KEY_OPTIONS. При использовании данного режима опции индексов не отображаются в результирующей таблице оператора SHOW CREATE TABLE. Режим появился, начиная с версии MySQL 4.1.1.
- NO_TABLE_OPTIONS. При использовании данного режима не отображаются параметры таблицы (такие как ENGINE, AUTO_INCREMENT и т. д.) в результирующей таблице оператора SHOW CREATE TABLE. Режим появился, начиная с версии MySQL 4.1.1.
- NO_UNSIGNED_SUBTRACTION. При использовании данного режима в операции вычитания результат не отмечается как UNSIGNED, если один из операторов не имеет знака.
- NO_ZERO_DATE. При использовании данного режима совместно со строгим режимом не допускается указание нулевых дат, вроде 0000-00-00. Если не установлен строгий режим, просто генерируется предупреждение. Данный режим появился, начиная с версии MySQL 5.0.2.
- NO_ZERO_IN_DATE. При использовании данного режима совместно со строгим режимом не допускаются нулевые значения для месяца и дня в дате. Если не используется строгий режим, просто генерируется предупреждение. Данный режим появился, начиная с версии MySQL 5.0.2.

- `ONLY_FULL_GROUP_BY`. В данном режиме запрещается ссылаться в конструкции `GROUP BY` на столбцы, которые не представлены в результирующей таблице.
- `PIPES_AS_CONCAT`. В данном режиме `||` трактуется как конкатенация строк, а не как синоним оператора `OR`.
- `REAL_AS_FLOAT`. В данном режиме тип `REAL` рассматривается как синоним `FLOAT`, а не `DOUBLE`.
- `STRICT_ALL_TABLES`. Данный режим включает строгий режим для всех типов таблиц. Режим появился, начиная с версии MySQL 5.0.2.
- `STRICT_TRANS_TABLES`. Данный режим включает строгий режим для транзакционных таблиц и, когда возможно, не для транзакционных. Режим появился, начиная с версии MySQL 5.0.2.

Строгий режим контролирует, как MySQL-сервер реагирует на ошибочные или пропущенные входные значения. Значение может быть ошибочным по нескольким причинам. Например, может иметь неверный тип данных для этого столбца или значение выходит за допустимые границы. В режимах `STRICT_ALL_TABLES` и `STRICT_TRANS_TABLES` сервер MySQL сообщает об ошибке, когда происходит попытка вставить ошибочное значение или когда значение пропускается.

> ***Замечание***
>
> Если не включен ни один из режимов `STRICT_ALL_TABLES` или `STRICT_TRANS_TABLES`, вместо пропущенного значения будет подставлено выражение по умолчанию.

Строгий режим не пропустит не корректную дату вроде `2006-04-31`, однако он разрешит вставить запись с датой вида `2006-04-00`. Для предотвращения этого следует дополнительно использовать режим `NO_ZERO_DATE` или `NO_ZERO_IN_DATE`.

Следующие специальные режимы предлагаются как сокращенные варианты для комбинаций режимов из предыдущего списка.

- `ANSI`. Данный режим эквивалентен набору режимов `REAL_AS_FLOAT`, `PIPES_AS_CONCAT`, `ANSI_QUOTES` и `IGNORE_SPACE`. В этом режиме синтаксис и поведение сервера MySQL изменяется таким образом, чтобы наиболее соответствовать SQL-стандарту. Данный режим появился, начиная с версии MySQL 4.1.
- `DB2`. Режим эквивалентен набору режимов `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE`, `NO_KEY_OPTIONS`, `NO_TABLE_OPTIONS`, `NO_FIELD_OPTIONS`.
- `MAXDB`. Режим эквивалентен набору режимов `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE`, `NO_KEY_OPTIONS`, `NO_TABLE_OPTIONS`, `NO_FIELD_OPTIONS`, `NO_AUTO_CREATE_USER`.
- `MSSQL`. Режим эквивалентен набору режимов `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE`, `NO_KEY_OPTIONS`, `NO_TABLE_OPTIONS`, `NO_FIELD_OPTIONS`.
- `MYSQL323`. Режим эквивалентен набору режимов `NO_FIELD_OPTIONS`, `HIGH_NOT_PRECEDENCE`.

- MYSQL40. Режим эквивалентен набору режимов NO_FIELD_OPTIONS, HIGH_NOT_PRECEDENCE.
- ORACLE. Режим эквивалентен набору режимов PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER.
- POSTGRESQL. Режим эквивалентен набору режимов PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.
- TRADITIONAL. Режим эквивалентен набору режимов STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER.

28.4. Журнальные файлы

MySQL поддерживает несколько видов журнальных файлов, в которых регистрируются различные события, происходящие на MySQL-сервере (табл. 28.1).

Таблица 28.1. *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, который рассматривается в *разд. 28.6*.

> ***Замечание***
>
> Как альтернативу оператору FLUSH LOGS можно использовать утилиту mysqladmin, которая рассматривается в *разд. 28.10*.

28.4.1. 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.

> ***Замечание***
>
> Таблицы `general_log` и `slow_log` имеют тип CSV, поэтому их можно просматривать непосредственно в программах, поддерживающих данный формат, например, в MS Excel.

Способ хранения журнальных записей выбирается при помощи параметра `--log-output[=value,...]` *(см. разд. 28.1)*. 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`.

28.4.2. Журнал ошибок

Журнал ошибок содержит информацию о том, когда запускался и останавливался сервер MySQL, а также 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.

Как уже упоминалось в *разд. 28.1*, при помощи параметра `--log-error[=file_name]` 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.

> ***Замечание***
>
> Более подробно оператор `FLUSH LOGS` рассматривается в *разд. 28.6*.

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

28.4.3. Общий журнал запросов

Общий журнал запросов позволяет регистрировать все соединения клиентов и их SQL-запросы. Для активации журнала сервер MySQL 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 запроса можно направлять не в файл, а в таблицу `general_log` системной базы `mysql` *(см. разд. 28.4.1)*.

Сервер MySQL записывает операторы в журнал в том порядке, в 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 обработки запроса.

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

28.4.4. Бинарный журнал регистраций

Бинарный журнал регистраций фиксирует 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 имеющееся у него значение.

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. 28.4.3)*.

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 содержаться 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 32)*.

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` *(см. разд. 28.2)*.

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.

Для удаления всех файлов бинарного журнала используется оператор `RESET MASTER`, а для удаления лишь некоторых из них — `PURGE MASTER LOGS`. Более подробно синтаксис данных операторов рассматривается в *главе 32*.

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 28.10, не попадет в бинарный журнал регистраций.

Листинг 28.10. Запрос, не попадающий в бинарный журнал регистраций

```
mysql> USE shop;
mysql> 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 28.11 попадет в бинарный журнал, несмотря на то, что сервер запускается с параметром `--binlog-ignore-db=sales`.

Листинг 28.11. Запрос, попадающий в бинарный журнал регистраций

```
mysql> USE shop;
mysql> 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, то осуществляется проверка, имеется ли текущая база данных (т. е. была или 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`, то проверяется, установлены ли какие-нибудь правила `binlog-do-db`?
   - Если да, то 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 `SET SQL_LOG_BIN=0`.

28.4.5. Утилита *mysqlbinlog*

Файлы бинарного журнала, которые генерирует сервер, записываются в двоичном формате. Для работы с такими файлами используется утилита `mysqlbinlog`. Синтаксис вызова утилиты `mysqlbinlog` выглядит следующим образом:

```
mysqlbinlog [parameters] filename
```

Здесь *parameters* — параметры утилиты, которые описываются далее, а *filename* — имя файла бинарного журнала. Например, для того чтобы отобразить содержимое бинарного журнала binlog.003, можно воспользоваться командой:

```
mysqlbinlog binlog.003
```

> ***Замечание***
>
> Как в UNIX-подобных операционных системах, так и в операционной системе Windows, для того чтобы направить вывод утилиты файл *file*, достаточно в конце команды добавить последовательность `>file`: `mysqlbinlog binlog.003 > file`. Для этого также можно использовать параметр `--result-file`, который описывается далее.

Вывод команды включает все операторы, содержащиеся в журнале binlog.003, вместе с дополнительной информацией, такой как время выполнения каждого оператора, идентификатор потока клиента, который его вызвал, временная метка и т. д.

Утилита `mysqlbinlog` поддерживает следующие параметры.

- `--help`, `-?`. Параметр выводит страницу справки и завершает работу.
- `--base64-output`. Данный параметр выводит бинарный журнал в стандартный вывод с использованием base64-кодирования. Параметр используется только для отладки и введен, начиная с версии MySQL 5.1.5.
- `--character-sets-dir=`*path*. Параметр позволяет передать путь к каталогу *path*, где установлены кодировки.
- `--database=`*db_name*, `-d` *db_name*. Выводит информацию, касающуюся только базы данных *db_name*.
- `--debug[=`*debug_options*`]`, `-#` `[`*debug_options*`]`. Данный параметр включает вывод отладочной информации. Строка *debug_options* задает формат строки вывода, обычно принимает значение `'d:t:o,`*file_name*`'`.
- `--disable-log-bin`, `-D`. При использовании данного параметра отключается ведение бинарного журнала регистраций. Этот параметр полезен при восстановлении сервера MySQL после повреждения, когда новые записи в бинарный журнал могут изменить его состояние. Для использования этого параметра MySQL-пользователь должен иметь привилегию `SUPER`. Параметр введен, начиная с версии MySQL 4.1.8.
- `--force-read`, `-f`. При использовании данного параметра, утилита `mysqlbinlog` читает даже те записи, которые не может распознать. Без использования данного параметра, если встречается неизвестная запись, которую невозможно расшифровать, работа утилиты прекращается.

- `--hexdump`, `-H`. При использовании данного параметра в комментариях к записям отображаются шестнадцатеричные коды. Параметр введен, начиная с версии MySQL .1.2.
- `--host=`*`host_name`*, `-h` *`host_name`*. В параметре указывается сетевое имя или IP-адрес компьютера *`host_name`*, на котором установлен MySQL-сервер.
- `--local-load=`*`path`*, `-l` *`path`*. При использовании данного параметра утилита `mysqlbinlog` подготавливает локальные временные файлы в указанном каталоге для оператора `LOAD DATA INFILE`.
- `--offset=`*`N`*, `-o` *`N`*. При использовании данного параметра, первые *`N`* записей журнала пропускаются.
- `--password[=`*`password`*`]`, `-p[`*`password`*`]`. В данном параметре указывается пароль *`password`* для доступа к MySQL-серверу.
- `--port=`*`port_num`*, `-P` *`port_num`*. Параметр указывает номер порта *`port_num`*, по которому MySQL-сервер ожидает соединения с клиентом (по умолчанию 3306, если параметр не указан, производится попытка установить соединение именно по данному порту).
- `--position=`*`N`*, `-j` *`N`*. Начинать читать бинарный журнал с позиции *`N`*. С версии 4.1.4 параметр считается устаревшим, на его смену пришел параметр `--start-position`.
- `--protocol={TCP|SOCKET|PIPE|MEMORY}`. Параметр позволяет задать протокол подключения к серверу. Параметр введен, начиная с версии MySQL 4.1.
- `--read-from-remote-server`, `-R`. Параметр сообщает, что бинарный журнал читается с удаленного сервера MySQL. Любые параметры подключения игнорируются (`--host`, `--password`, `--port`, `--protocol`, `--socket` и `--user`), если не указан этот параметр.
- `--result-file=`*`name`*, `-r` *`name`*. При использовании данного параметра вывод утилиты направляется в файл *`name`*.
- `--server-id=`*`id`*. Выводятся только те записи, которые были осуществлены сервером с номером *`id`*. Параметр введен, начиная с версии MySQL 5.1.4.
- `--short-form`, `-s`. При использовании данного параметра выводятся только SQL-операторы без дополнительной информации.
- `--socket=`*`path`*, `-S` *`path`*. Параметр позволяет указать путь к файлу сокета, который будет использоваться для подключения к MySQL-серверу.
- `--start-datetime=`*`datetime`*. При использовании данного параметра из журнального файла будут извлекаться только те записи, которые были помещены в журнал после даты *`datetime`*. Формат записи *`datetime`* должен соответствовать типу `DATETIME` или `TIMESTAMP`, например, `mysqlbinlog --start-datetime="2005-12-25 11:25:56" binlog.000003`. Параметр введен, начиная с версии MySQL 4.1.4.

- `--stop-datetime=`*`datetime`*. При использовании параметра из журнального файла будут извлекаться только те записи, которые были помещены в журнал до даты *`datetime`*. Формат записи *`datetime`* должен соответствовать типу `DATETIME` или `TIMESTAMP`. Параметр введен, начиная с версии MySQL 4.1.4.
- `--start-position=`*`N`*. При использовании данного параметра из журнального файла будут извлекаться записи, начиная с позиции *`N`*. Параметр введен, начиная с версии MySQL 4.1.4.
- `--stop-position=`*`N`*. При использовании данного параметра из журнального файла будут извлекаться записи до позиции *`N`*. Параметр введен, начиная с версии MySQL 4.1.4.
- `--to-last-log`, `-t`. При использовании данного параметра утилита не прекращает вывод информации при достижении конца текущего файла бинарного журнала, а находит следующий файл и распечатывает его. Это продолжается до тех пор, пока не будет распечатан последний бинарный журнал. Данный параметр работает только совместно с параметром `--read-from-remote-server`.
- `--user=`*`user_name`*, `-u` *`user_name`*. Параметр указывает пользователя *`user_name`*, от имени которого осуществляется соединение с MySQL-сервером.
- `--version`, `-V`. При использовании данного параметра, утилита `mysqlbinlog` выводит информацию о версии сервера и завершает работу.

28.4.6. 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 `logn_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`.

28.5. Оператор *CACHE INDEX*

Оператор `CACHE INDEX` назначает индексам таблиц специальный кэш ключей (что ускоряет их работу) и имеет следующий синтаксис:

```
CACHE INDEX
  tbl_index_list [, tbl_index_list] ...
  IN key_cache_name


tbl_index_list:
  tbl_name [[INDEX|KEY] (index_name[, index_name] ...)]
```

> ***Замечание***
>
> Оператор `CACHE INDEX` работает только с таблицами типа MyISAM. Оператор был введен, начиная с версии MySQL 4.1.1.

Оператор, представленный в листинге 28.12, назначает индексам в таблицах `t1`, `t2` и `t3` общий ключевой кэш с именем `hot_cache`.

Листинг 28.12. Использование оператора `CACHE INDEX`

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+---------+--------------------+----------+----------+
```

Несмотря на то, что синтаксис оператора `CACHE INDEX` позволяет привязать к кэшу отдельные индексы таблицы, в текущей реализации все индексы таблицы привязываются к указанному кэшу, поэтому нет причин уточнять индексы таблицы — достаточно только указать имя таблицы.

Кэш ключей `hot_chache` не появляется самопроизвольно — его требуется создать, в противном случае попытка использования его в операторе `CACHE INDEX` закончится ошибкой (листинг 28.13).

Листинг 28.13. Использование несуществующего кэша ключей

```
mysql> CACHE INDEX t1 IN non_existent_cache;
ERROR 1284 (HY000): Unknown key cache 'non_existent_cache'
```

> ***Замечание***
>
> Более подробно синтаксис оператора `SET` рассматривается в *главе 29*.

Кэш ключей может быть создан путем установки его размера при помощи оператора `SET` либо в настройках параметров сервера. В листинге 28.14 приводится пример создания кэша ключей `hot_cache`.

Листинг 28.14. Создание кэша ключей `hot_cache`

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

Созданный кэш является глобальным. Если одни клиент назначает индекс определенному кэшу, этот кэш будет использоваться при выполнении всех запросов независимо от того, какой клиент их инициировал.

28.6. Оператор *FLUSH*

Оператор `FLUSH` применяется, когда необходимо очистить внутренние кэши и журналы MySQL. Оператор имеет следующий синтаксис:

```
FLUSH [LOCAL | NO_WRITE_TO_BINLOG] flush_option[, flush_option] ...
```

> ***Замечание***
>
> Для выполнения оператора `FLUSH` необходимо обладать привилегией `RELOAD`.

Параметр *`flush_option`* (который может встретиться несколько раз) может принимать любое из перечисленных далее значений.

- `HOSTS`. При использовании данного параметра оператор `FLUSH` очищает кэш хостов. Использование `FLUSH HOSTS` рекомендуется, когда у какого-либо клиента меняется IP-адрес либо если сервер возвращает сообщение об ошибке "Host ... is blocked" ("Хост ... заблокирован"). Последнее сообщение может быть вызвано следующей причиной. Когда происходит больше, чем `max_connect_errors` ошибок при подключении клиента к серверу, MySQL предполагает, что с хостом клиента что-то не в порядке, и блокирует дальнейшие попытки подключения с данного IP-адреса. Сброс кэша хостов опять снимает эту блокировку и позволяет продолжить попытки подключения.
- `DES_KEY_FILE`. Оператор `FLUSH DES_KEY_FILE` перезагружает DES-ключи из файла, заданного параметром `--des-key-file` *(см. разд. 28.1)*.

- `LOGS`. Оператор `FLUSH LOGS` закрывает и заново открывает все журнальные файлы. Для бинарного журнала запросов открывается новый журнал, номер в имени которого увеличен на единицу по сравнению с предыдущим.
- `PRIVILEGES`. Оператор `FLUSH PRIVILEGES` перезагружает привилегии в оперативной памяти, загружая их из таблицы привилегий базы данных `mysql`.
- `QUERY CACHE`. Оператор `FLUSH QUERY CACHE` выполняет дефрагментацию кэша запросов с целью более эффективного использования памяти.
- `STATUS`. Оператор `FLUSH STATUS` сбрасывает большинство серверных переменных в ноль.
- `{TABLE | TABLES} [`*`tbl_name`*`[, `*`tbl_name`*`] ...]`. Когда не указана таблица *`tbl_name`*, оператор `FLUSH TABLES` закрывает все открытые таблицы и принуждает к закрытию все используемые таблицы. Дополнительно к этому очищается кэш запросов. Если после оператора перечисляется одна или несколько таблиц, сбрасываются только эти таблицы.
- `TABLES WITH READ LOCK`. Оператор `FLUSH TABLES WITH READ LOCK` закрывает все открытые таблицы и блокирует все таблицы во всех базах данных блокировкой по чтению до тех пор, пока не будет выполнен оператор `UNLOCK TABLES`.
- `USER_RESOURCES`. Оператор `FLUSH USER_RESOURCES` сбрасывает в ноль все пользовательские ресурсы. Это позволяет клиентам, которые достигли лимита по максимальному числу соединений, максимальному числу запросов или обновлений, продолжить работу.

До версии MySQL 4.1.1 операторы `FLUSH` не записывались в бинарный журнал регистраций. Начиная с MySQL 4.1.1, этот оператор записывается в бинарный журнал, если указывается необязательное слово `NO_WRITE_TO_BINLOG` (или его синоним `LOCAL`). Исключениями являются операторы `FLUSH LOG`, `FLUSH MASTER`, `FLUSH SLAVE` и `FLUSH TABLES WITH READ LOCK`, которые не регистрируются в любом случае, чтобы не помешать репликации на подчиненном сервере *(см. главу 32)*.

28.7. Оператор *KILL*

Оператор `KILL` позволяет прервать соединение по его идентификатору *`id`* и имеет следующий синтаксис:

```
KILL [CONNECTION | QUERY] id
```

Список всех текущих процессов, а также их идентификаторы, можно просмотреть при помощи оператора `SHOW PROCESSLIST` (см. листинг 30.22).

Начиная с MySQL версии 5.0.0, оператор `KILL` предусматривает необязательные модификаторы `CONNECTION` и `QUERY`:

- `KILL CONNECTION` — это то же самое, что оператор `KILL` без дополнительных модификаторов, при выполнении этого запроса прерывается соединение с идентификатором *`id`*;
- `KILL QUERY` — прерывается только оператор, который выполняется на данный момент в потоке с идентификатором *`id`*, но само соединение не прерывается.

28.8. Оператор *LOAD INDEX INTO CACHE*

Оператор `LOAD INDEX INTO CACHE` предварительно загружает индекс таблицы в кэш ключей, с которым он связан явным вызовом оператора `CACHE INDEX` *(см. разд. 28.5)*. Оператор имеет следующий синтаксис:

```
LOAD INDEX INTO CACHE
  tbl_index_list[, tbl_index_list] ...

tbl_index_list:
  tbl_name
    [[INDEX|KEY] (index_name[, index_name] ...)]
    [IGNORE LEAVES]
```

> ***Замечание***
>
> Оператор `LOAD INDEX INTO CACHE` работает только с таблицами типа MyISAM. Оператор был введен, начиная с версии MySQL 4.1.1.

Если указан модификатор `IGNORE LEAVES`, в кэш загружаются только блоки нелистовых узлов индексного дерева.

В листинге 28.15 в кэш загружаются все индексные блоки таблицы `t1` и нелистовые узлы таблицы `t2`.

Листинг 28.15. Использование оператора `LOAD INDEX INTO CACHE`

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+---------+--------------+----------+----------+
| Table   | Op           | Msg_type | Msg_text |
+---------+--------------+----------+----------+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+---------+--------------+----------+----------+
```

Несмотря на то, что синтаксис оператора `LOAD INDEX INTO CACHE` позволяет загрузить в кэш отдельные индексы таблицы, в текущей реализации все индексы таблицы загружаются в кэш, поэтому нет причин уточнять индексы таблицы — достаточно только указать имя таблицы.

28.9. Оператор *RESET*

Оператор `RESET` используется для очистки состояния различных операций сервера. Он также выступает как более строгая версия оператора `FLUSH` *(см. разд. 28.6)*. Оператор имеет следующий синтаксис:

```
RESET reset_option[, reset_option] ...
```

Замечание

Для использования оператора `RESET` необходимо обладать привилегией `RELOAD`.

Параметр *reset_option* может принимать одно из следующих значений.

- `MASTER`. При использовании данного параметра удаляются все бинарные журналы, перечисленные в индексном файле (.index), очищается индексный файл бинарного журнала и создается новый файл бинарного журнала.
- `QUERY CACHE`. При использовании данного параметра удаляются все записи из кэша запросов.
- `SLAVE`. Заставляет подчиненный сервер "забыть" свою позицию репликации в бинарном журнале главного сервера. Ранее оператор назывался `FLUSH SLAVE`.

28.10. Утилита *mysqladmin*

Утилита `mysqladmin` позволяет выполнять операции администрирования из командной строки. Как правило, утилита применяется для проверки конфигурации и текущего состояния сервера, создания и удаления баз данных и других административных функций.

Синтаксис запуска утилиты `mysqladmin` выглядит следующим образом:

```
mysqladmin [parameters] command [command-arg] [command [command-arg]] ...
```

Помимо традиционных параметров *parameters*, утилита принимает команды *command*. Команды в свою очередь могут принимать аргументы *command-arg*. Утилита `mysqladmin` может принимать сразу несколько команд *command*. Далее описываются команды, которые поддерживаются текущей версией `mysqladmin`.

- `create` *db_name*. Команда `mysqladmin create` *db_name* создает новую базу данных с именем *db_name*. Данная команда аналогична вызову оператора `CREATE DATABASE` *db_name*.
- `debug`. При использовании данная команда требует от MySQL-сервера, чтобы он записывал в журнал ошибок отладочную информацию.
- `drop` *db_name*. Команда `mysqladmin drop` *db_name* удаляет базу данных *db_name* и все ее таблицы. Данная команда аналогична вызову оператора `DROP DATABASE` *db_name*.
- `extended-status`. При использовании данной команды выводится таблица с системными переменными сервера и их значениями. Системные переменные подробно описываются в *разд. 28.2*. Данная команда аналогична вызову оператора `SHOW VARIABLES`.
- `flush-hosts`. При использовании данной команды сбрасывается вся информация из кэша хостов. Команда аналогична вызову оператора `FLUSH HOSTS`.
- `flush-logs`. При использовании данной команды сбрасываются все журналы MySQL. Использование команды аналогично вызову оператора `FLUSH LOGS`.

- `flush-privileges`. Команда перезагружает привилегии в оперативной памяти, загружая их из таблицы привилегий базы данных `mysql`. Использование команды аналогично вызову оператора `FLUSH PRIVILEGES`.
- `flush-status`. Команда сбрасывает большинство серверных переменных в ноль. Использование команды аналогично вызову оператора `FLUSH STATUS`.
- `flush-tables`. Команда сбрасывает буферы таблицы. Использование команды аналогично вызову оператора `FLUSH TABLES`.
- `flush-threads`. Команда сбрасывает кэш потоков.
- `kill` *`id`*`,`*`id`*`,...` Команда прерывает потоки сервера с идентификаторами *`id`*. Использование команды аналогично вызову оператора `KILL` *`id`*.
- `password` *`new-password`*. Команда устанавливает новый пароль для пользователя, от имени которого выполняется соединение с сервером. Если пароль *`new-password`* содержит пробелы, он заключается в кавычки (в Windows необходимо использовать двойные кавычки).
- `old-password` *`new-password`*. Данная команда аналогична команде `password`, однако пароль сохраняется в старом формате (до версии 4.1).
- `ping`. При использовании данной команды осуществляется проверка, работает ли сервер.
- `processlist`. При использовании данного параметра выводится список активных серверных потоков. Применение команды аналогично вызову оператора `SHOW PROCESSLIST`. Если указывается параметр `--verbose`, вывод будет подобен выводу оператора `SHOW FULL PROCESSLIST`.
- `reload`. Команда перезагружает привилегии в оперативной памяти, загружая их из таблицы привилегий базы данных `mysql`. Использование команды аналогично вызову оператора `FLUSH PRIVILEGES`.
- `refresh`. При использовании данного параметра сбрасываются буферы всех таблиц, закрываются и открываются файлы журналов.
- `shutdown`. Команда останавливает MySQL-сервер.
- `start-slave`. Команда запускает репликацию на подчиненном сервере.
- `status`. При использовании команды выводится краткое сообщение о состоянии сервера.
- `stop-slave`. Команда останавливает репликацию на подчиненном сервере.
- `variables`. При использовании данной команды выводится таблица с системными переменными сервера и их значениями. Системные переменные подробно описываются в *разд. 28.2*. Данная команда аналогична вызову оператора `SHOW VARIABLES`.
- `version`. Команда отображает версию сервера.

Все команды могут быть сокращены до любого уникального префикса (листинг 28.16).

Листинг 28.16. Использование сокращенной формой команды `mysqladmin`

```
shell> mysqladmin proc stat
+----+-------+-----------+----+---------+------+-------+----------------+
| Id | User  | Host      | db | Command | Time | State | Info           |
+----+-------+-----------+----+---------+------+-------+----------------+
| 51 | monty | localhost |    | Query   | 0    |       |show processlist|
+----+-------+-----------+----+---------+------+-------+----------------+
Uptime: 1473624  Threads: 1  Questions: 39487
Slow queries: 0  Opens: 541  Flush tables: 1
Open tables: 19  Queries per second avg: 0.0268
```

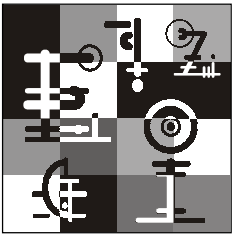
Команда `mysqladmin status` выводит следующие значения:

- `Uptime` — длительность работы сервера в секундах;
- `Threads` — количество активных потоков (клиентов);
- `Questions` — количество клиентских запросов с момента старта сервера;
- `Slow queries` — количество запросов, на выполнение которых потребовалось более `long_query_time` секунд;
- `Opens` — количество таблиц, открытых сервером;
- `Flush tables` — количество выполненных сервером команд `flush`, `refresh` и `reload`;
- `Open tables` — количество таблиц, открытых в данный момент;
- `Memory in use` — общий объем памяти, распределенной непосредственно кодом сервера MySQL. Это значение отображается, только если MySQL был скомпилирован с параметром `--with-debug-full`;
- `Maximum memory used` — максимальный объем памяти, распределенный непосредственно кодом MySQL. Это значение отображается, только если MySQL был скомпилирован с параметром `--with-debug-full`.

Утилита `mysqladmin` поддерживает следующие параметры.

- `--help`, `-?`. Параметр выводит страницу справки и завершает работу.
- `--character-sets-dir=`*`path`*. Параметр позволяет передать путь к каталогу *`path`*, где установлены кодировки.
- `--compess`, `-C`. При использовании данного параметра вся информация, передаваемая между клиентом и сервером, подвергается сжатию.
- `--count=`*`N`*, `-c` *`N`*. Параметр задает количество итераций *`N`*, которые необходимо выполнить. Параметр работает только с параметром `--sleep` (`-i`).
- `--debug[=`*`debug_options`*`]`, `-#` `[`*`debug_options`*`]`. Данный параметр включает вывод отладочной информации. Строка *`debug_options`* задает формат строки вывода, обычно принимает значение `'d:t:o,`*`file_name`*`'`.

- `--force`, `-f`. При использовании данного параметра, утилита `mysqladmin` не будет запрашивать подтверждения для команды `drop database`, а в случае возникновения ошибки утилита продолжает работу.
- `--host=`*`host_name`*, `-h` *`host_name`*. В параметре указывается сетевое имя или IP-адрес компьютера *`host_name`*, на котором установлен MySQL-сервер.
- `--password[=`*`password`*`]`, `-p[`*`password`*`]`. В данном параметре указывается пароль *`password`* для доступа к MySQL-серверу.
- `--port=`*`port_num`*, `-P` *`port_num`*. Параметр указывает номер порта *`port_num`*, по которому MySQL-сервер ожидает соединения с клиентом (по умолчанию 3306, если этот параметр не указан, производится попытка установить соединение именно по данному порту).
- `--protocol={TCP|SOCKET|PIPE|MEMORY}`. Параметр позволяет задать протокол подключения к серверу. Параметр введен, начиная с версии MySQL 4.1.
- `--relative`, `-r`. При использовании данного параметра показывается разница между текущим и предыдущим значениями (параметр применяется совместно с параметром `-i`). В настоящее время этот параметр работает только с командой `extended-status`.
- `--silent`, `-s`. При использовании данного параметра утилита завершает работу молча, если невозможно установить подключение к серверу.
- `--sleep=`*`delay`*, `-i` *`delay`*. Утилита выполняет команду с задержками *`delay`* между попытками.
- `--socket=`*`path`*, `-S` *`path`*. Параметр позволяет указать путь к файлу сокета, который будет использоваться для подключения к MySQL-серверу.
- `--user=`*`user_name`*, `-u` *`user_name`*. Параметр указывает пользователя *`user_name`*, от имени которого осуществляется соединение с MySQL-сервером.
- `--verbose`, `-v`. При использовании данного параметра утилита работает в режиме расширенных сообщений и генерирует более подробные отчеты.
- `--version`, `-V`. При использовании данного параметра, утилита `mysqladmin` выводит информацию о версии сервера и завершает работу.
- `--vertical`, `-E`. Параметр аналогичен параметру `--relative`, но печатает вывод вертикально.
- `--wait[=`*`count`*`]`, `-w[`*`count`*`]`. Если соединение не может быть установлено, утилита `mysqladmin` повторяет попытку соединения, вместо того, чтобы прервать работу. Необязательное значение *`count`* определяет количество повторных попыток.



Глава 29

Оператор *SET*

Оператор `SET` позволяет устанавливать системные переменные, которые влияют на работу сервера и клиента. В *главе 23* оператор `SET` уже рассматривался при определении пользовательских переменных (листинг 29.1).

> ***Замечание***
>
> Оператор `SET`, как и большинство административных операторов, не является стандартным SQL-оператором и применяется только в СУБД MySQL.

Листинг 29.1. Использование оператора `SET`

```
mysql> SET @total = 5;
mysql> SELECT @total;
+--------+
| @total |
+--------+
| 5      |
+--------+
```

Один оператор `SET` может использоваться для определения сразу нескольких значений, для этого переменные разделяются запятой (листинг 29.2).

Листинг 29.2. Определение нескольких переменных

```
mysql> SET @total = 5, @number = 30, @str = 'Hello, world!';
mysql> SELECT @total, @number, @str;
+--------+---------+---------------+
| @total | @number | @str          |
+--------+---------+---------------+
| 5      | 30      | Hello, world! |
+--------+---------+---------------+
```

Помимо пользовательских переменных, оператор `SET` позволяет динамически (без перезапуска сервера) изменять системные и сеансовые переменные. Сервер MySQL поддерживает два типа переменных: *глобальные*, которые влияют на сервер, и *сеансовые*, которые влияют на текущее соединение клиента с сервером.

> ***Замечание***
>
> Подробнее системные переменные описываются в *разд. 28.2*.

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 29.3).

> ***Замечание***
>
> Для изменения глобальных переменных требуется иметь привилегию `SUPER`.

Листинг 29.3. Установка глобальной переменной

```
mysql> SET GLOBAL read_buffer_size = 2097152;
```

В листинге 29.3 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 29.4.

> ***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 29.4. Альтернативный способ установки глобальных переменных

```
mysql> SET @@global.read_buffer_size = 2097152;
```

Запросы в листингах 29.3 и 29.4 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 29.5).

> ***Замечание***
>
> Установка сеансовых переменных не требует специальных привилегий.

Листинг 29.5. Установка сеансовой переменной

```
mysql> 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 используется оператор `SET GLOBAL` с переменной, которую можно использовать только с `SET SESSION`.

Помимо представленного в листинге 29.5 синтаксиса, оператор `SET` имеет несколько альтернативных форм для определения сессионной переменной (листинг 29.6).

> ***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 29.6. Установка сеансовой переменной

```
mysql> SET SESSION read_buffer_size = 2097152;
mysql> SET LOCAL read_buffer_size = 2097152;
mysql> SET read_buffer_size = 2097152;
mysql> SET @@session.read_buffer_size = 2097152;
```

Представленные в листинге 29.6 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.

Чтобы установить локальной переменной значение глобальной, достаточно присвоить локальной переменной ключевое слово `DEFAULT` или осуществить присвоение явно (листинг 29.7).

Листинг 29.7. Использование ключевого слова `DEFAULT`

```
mysql> SET read_buffer_size = DEFAULT;
mysql> SET @@session.read_buffer_size = @@global.read_buffer_size;
```

Для того чтобы узнать текущее значение глобальной переменной, к ней требуется обратиться либо с добавлением суффикса `@@global`, либо использовать оператор `SHOW` (листинг 29.8).

Листинг 29.8. Извлечение значения глобальной переменной

```
mysql> SELECT @@global.read_buffer_size;
+---------------------------+
| @@global.read_buffer_size |
+---------------------------+
|                   2097152 |
+---------------------------+
mysql> 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 29.9).

Листинг 29.9. Извлечение значения сеансовой переменной

```
mysql> SELECT @@read_buffer_size;
+--------------------+
| @@read_buffer_size |
+--------------------+
|            2097152 |
+--------------------+
mysql> SELECT @@session.read_buffer_size;
+----------------------------+
| @@session.read_buffer_size |
+----------------------------+
|                    2097152 |
+----------------------------+
```

```
mysql> SHOW SESSION VARIABLES LIKE 'read_buffer_size';
+------------------+---------+
| Variable_name    | Value   |
+------------------+---------+
| read_buffer_size | 2097152 |
+------------------+---------+
```

Не все системные переменные, которые описываются в *разд. 28.2*, могут быть установлены при помощи оператора `SET`. В табл. 29.1 описываются переменные, которые могут быть установлены без перезагрузки и перекомпиляции сервера, а также указывается, могут ли они использоваться на глобальном уровне (`GLOBAL`), сеансовом (`SESSION`) или сразу на обоих (`GLOBAL | SESSION`).

Таблица 29.1. *Системные переменные, значения которых можно изменять без перезапуска сервера*

Имя переменной	Тип переменной	Уровень
autocommit	boolean	SESSION
big_tables	boolean	SESSION
binlog_cache_size	numeric	GLOBAL
binlog_format	string	GLOBAL | SESSION
bulk_insert_buffer_size	numeric	GLOBAL | SESSION
character_set_client	string	GLOBAL | SESSION
character_set_connection	string	GLOBAL | SESSION
character_set_filesystem	string	GLOBAL | SESSION
character_set_results	string	GLOBAL | SESSION
character_set_server	string	GLOBAL | SESSION
collation_connection	string	GLOBAL | SESSION
collation_server	string	GLOBAL | SESSION
completion_type	numeric	GLOBAL | SESSION
concurrent_insert	boolean	GLOBAL
connect_timeout	numeric	GLOBAL
convert_character_set	string	GLOBAL | SESSION
default_week_format	numeric	GLOBAL | SESSION
delay_key_write	OFF | ON | ALL	GLOBAL
delayed_insert_limit	numeric	GLOBAL
delayed_insert_timeout	numeric	GLOBAL

Таблица 29.1 *(продолжение)*

Имя переменной	Тип переменной	Уровень
delayed_queue_size	numeric	GLOBAL
div_precision_increment	numeric	GLOBAL | SESSION
engine_condition_pushdown	boolean	GLOBAL | SESSION
error_count	numeric	SESSION
event_scheduler	boolean	GLOBAL
expire_logs_days	numeric	GLOBAL
flush	boolean	GLOBAL
flush_time	numeric	GLOBAL
foreign_key_checks	boolean	SESSION
ft_boolean_syntax	numeric	GLOBAL
group_concat_max_len	numeric	GLOBAL | SESSION
identity	numeric	SESSION
innodb_autoextend_increment	numeric	GLOBAL
innodb_commit_concurrency	numeric	GLOBAL
innodb_concurrency_tickets	numeric	GLOBAL
innodb_max_dirty_pages_pct	numeric	GLOBAL
innodb_max_purge_lag	numeric	GLOBAL
innodb_support_xa	boolean	GLOBAL | SESSION
innodb_sync_spin_loops	numeric	GLOBAL
innodb_table_locks	boolean	GLOBAL | SESSION
innodb_thread_concurrency	numeric	GLOBAL
innodb_thread_sleep_delay	numeric	GLOBAL
insert_id	boolean	SESSION
interactive_timeout	numeric	GLOBAL | SESSION
join_buffer_size	numeric	GLOBAL | SESSION
key_buffer_size	numeric	GLOBAL
last_insert_id	numeric	SESSION
local_infile	boolean	GLOBAL
log_warnings	numeric	GLOBAL
long_query_time	numeric	GLOBAL | SESSION
low_priority_updates	boolean	GLOBAL | SESSION

Таблица 29.1 *(продолжение)*

Имя переменной	Тип переменной	Уровень
max_allowed_packet	numeric	GLOBAL | SESSION
max_binlog_cache_size	numeric	GLOBAL
max_binlog_size	numeric	GLOBAL
max_connect_errors	numeric	GLOBAL
max_connections	numeric	GLOBAL
max_delayed_threads	numeric	GLOBAL
max_error_count	numeric	GLOBAL | SESSION
max_heap_table_size	numeric	GLOBAL | SESSION
max_insert_delayed_threads	numeric	GLOBAL
max_join_size	numeric	GLOBAL | SESSION
max_relay_log_size	numeric	GLOBAL
max_seeks_for_key	numeric	GLOBAL | SESSION
max_sort_length	numeric	GLOBAL | SESSION
max_tmp_tables	numeric	GLOBAL | SESSION
max_user_connections	numeric	GLOBAL
max_write_lock_count	numeric	GLOBAL
myisam_stats_method	enum	GLOBAL | SESSION
multi_read_range	numeric	GLOBAL | SESSION
myisam_data_pointer_size	numeric	GLOBAL
log_bin_trust_function_creators	boolean	GLOBAL
myisam_max_sort_file_size	numeric	GLOBAL | SESSION
myisam_repair_threads	numeric	GLOBAL | SESSION
myisam_sort_buffer_size	numeric	GLOBAL | SESSION
myisam_use_mmap	boolean	GLOBAL
ndb_extra_logging	numeric	GLOBAL
net_buffer_length	numeric	GLOBAL | SESSION
net_read_timeout	numeric	GLOBAL | SESSION
net_retry_count	numeric	GLOBAL | SESSION
net_write_timeout	numeric	GLOBAL | SESSION
old_passwords	numeric	GLOBAL | SESSION
optimizer_prune_level	numeric	GLOBAL | SESSION

Таблица 29.1 *(продолжение)*

Имя переменной	Тип переменной	Уровень
optimizer_search_depth	numeric	GLOBAL | SESSION
preload_buffer_size	numeric	GLOBAL | SESSION
query_alloc_block_size	numeric	GLOBAL | SESSION
query_cache_limit	numeric	GLOBAL
query_cache_size	numeric	GLOBAL
query_cache_type	enumeration	GLOBAL | SESSION
query_cache_wlock_invalidate	boolean	GLOBAL | SESSION
query_prealloc_size	numeric	GLOBAL | SESSION
range_alloc_block_size	numeric	GLOBAL | SESSION
read_buffer_size	numeric	GLOBAL | SESSION
read_only	numeric	GLOBAL
read_rnd_buffer_size	numeric	GLOBAL | SESSION
rpl_recovery_rank	numeric	GLOBAL
safe_show_database	boolean	GLOBAL
secure_auth	boolean	GLOBAL
server_id	numeric	GLOBAL
slave_compressed_protocol	boolean	GLOBAL
slave_net_timeout	numeric	GLOBAL
slave_transaction_retries	numeric	GLOBAL
slow_launch_time	numeric	GLOBAL
sort_buffer_size	numeric	GLOBAL | SESSION
sql_auto_is_null	boolean	SESSION
sql_big_selects	boolean	SESSION
sql_big_tables	boolean	SESSION
sql_buffer_result	boolean	SESSION
sql_log_bin	boolean	SESSION
sql_log_off	boolean	SESSION
sql_log_update	boolean	SESSION
sql_low_priority_updates	boolean	GLOBAL | SESSION
sql_max_join_size	numeric	GLOBAL | SESSION
sql_mode	enumeration	GLOBAL | SESSION

Таблица 29.1 *(окончание)*

Имя переменной	Тип переменной	Уровень
sql_notes	boolean	SESSION
sql_quote_show_create	boolean	SESSION
sql_safe_updates	boolean	SESSION
sql_select_limit	numeric	SESSION
sql_slave_skip_counter	numeric	GLOBAL
updatable_views_with_limit	enumeration	GLOBAL | SESSION
sql_warnings	boolean	SESSION
sync_binlog	numeric	GLOBAL
sync_frm	boolean	GLOBAL
storage_engine	enumeration	GLOBAL | SESSION
table_definition_cache	numeric	GLOBAL
table_open_cache	numeric	GLOBAL
table_type	enumeration	GLOBAL | SESSION
thread_cache_size	numeric	GLOBAL
time_zone	string	GLOBAL | SESSION
timestamp	boolean	SESSION
tmp_table_size	enumeration	GLOBAL | SESSION
transaction_alloc_block_size	numeric	GLOBAL | SESSION
transaction_prealloc_size	numeric	GLOBAL | SESSION
tx_isolation	enumeration	GLOBAL | SESSION
unique_checks	boolean	SESSION
wait_timeout	numeric	GLOBAL | SESSION
warning_count	numeric	SESSION

В приведенном далее списке перечислены переменные, которые не отображаются оператором `SHOW VARIABLES`, тем не менее, их также можно просмотреть при помощи оператора `SELECT` (за исключением `CHARACTER SET` и `NAMES`) и изменить при помощи оператора `SET`.

- `AUTOCOMMIT = {0 | 1}`. Переменная принимает либо значение 1, либо 0 и управляет режимом автоматического завершения транзакций (см. листинг 26.5). Если устанавливается значение 1, все изменения, произведенные операторами, вступают в силу немедленно. Если установлено значение 0, то изменения вступают в силу только в результате применения оператора `COMMIT` или при неявном завершении транзакции. Оператор `ROLLBACK` позволяет отменить действия, произ-

веденные в рамках текущей транзакции. Подробнее с механизмом транзакций можно ознакомиться в *главе 26*.

- `BIG_TABLES = {0 | 1}`. Переменная принимает либо значение 1, либо 0 и контролирует режим управления временными таблицами (см. листинг 23.13). Если устанавливается значение 1, все временные таблицы сохраняются на диске, вместо того, чтобы размещаться в оперативной памяти. Начиная с версии MySQL 4.0.0, ручное управление этой переменной, скорее всего, не потребуется, т. к. MySQL сама принимает решение о том, где размещать временную таблицу. Подробнее с временными таблицами можно ознакомиться в *главе 23*.
- `CHARACTER SET {`*`character_set`*` | DEFAULT}`. Переменная позволяет установить кодировку, в которой клиент предпочитает отправлять и получать данные. В качестве имени может использоваться любое значение из первого столбца результирующей таблицы, возвращаемой оператором `SHOW CHARACTER SET` (см. листинг 12.4). Для восстановления кодировки по умолчанию вместо имени кодировки следует использовать ключевое слово `DEFAULT`. Переменная имеет синоним `NAMES`. Следует помнить, что оператор полноценно работает, начиная с версии 4.1.0, в более ранних версиях единственным допустимым значением было `cp1551_koi8`. Более подробно работа с кодировками обсуждается в *главе 14*.
- `FOREIGN_KEY_CHECKS = {0 | 1}`. Переменная принимает либо значение 1, либо 0 и управляет ограничениями внешних ключей для таблиц InnoDB. Если устанавливается значение 1, при операциях `DELETE` и `UPDATE` СУБД MySQL проверяет связи между таблицами, если устанавливается значение 0, такая проверка игнорируется. Отключение проверки внешних ключей может быть полезным для перезагрузки таблицы InnoDB в порядке, отличающемся от того, который требуют связи первичный/внешний ключ. Подробнее с внешними ключами можно ознакомиться в *главе 25*.
- `IDENTITY =` *`num`*. Переменная `IDENTITY` является синонимом переменной `LAST_INSERT_ID` и устанавливает значение, которое будет возвращено встроенной функцией `LAST_INSERT_ID()` (см. листинг 22.13), возвращающей последнее значение, присвоенное в результате действия механизма `AUTO_INCREMENT`. Наряду с использованием функции `LAST_INSERT_ID()`, для извлечения этого значения можно воспользоваться запросом `SELECT @IDENTITY` или установить новое значение переменной *`num`* при помощи запроса `SET IDENTITY =` *`num`*. Переменная `IDENTITY` введена в СУБД MySQL для совместимости с другими базами данных.
- `INSERT_ID =` *`num`*. Устанавливает значение, которое будет использовано следующим оператором `INSERT` или `ALTER TABLE` для вставки в столбец `AUTO_INCREMENT`.
- `LAST_INSERT_ID =` *`num`*. Устанавливает значение, возвращаемое функцией `LAST_INSERT_ID()`, синтаксис которой описывается в *главе 22*. Данная переменная является синонимом `IDENTITY`.
- `NAMES {`*`character_set`*` | DEFAULT}`. Переменная позволяет установить кодировку, в которой клиент предпочитает отправлять и получать данные. В качестве

имени может использоваться любое значение из первого столбца результирующей таблицы, возвращаемой оператором SHOW CHARACTER SET (см. листинг 12.4). Для восстановления кодировки по умолчанию вместо имени кодировки следует использовать ключевое слово DEFAULT. Переменная имеет синоним CHARACTER SET. Эта переменная появилась в СУБД MySQL, начиная с версии 4.1.0. Более подробно работа с кодировками обсуждается в *главе 14*.

- ONE_SHOT. Данная опция является модификатором, а не полноценным параметром. Она служит для единовременного изменения переменных, ответственных за сортировку, кодировку и временную зону. То есть сразу после выполнения следующего за SET оператора, измененные переменные получают исходное значение. Более подробно синтаксис оператора SET с применением данного модификатора обсуждается в конце главы.
- SQL_AUTO_IS_NULL = {0 | 1}. Переменная принимает либо значение 1, либо 0 (см. листинг 22.17). Если значение установлено в 1 (по умолчанию), то поиск последней вставленной в таблицу строки, содержащей столбец, снабженный атрибутом AUTO_INCREMENT, можно осуществлять при помощи условия WHERE *col_auto* IS NULL (см. листинг 22.16). Установка переменной в 0 позволяет отключить такое поведение.
- SQL_BIG_SELECTS = {0 | 1}. Переменная принимает значение либо 1, либо 0 и управляет прерываниями неэффективных SELECT-запросов. Если значение переменной установлено в 0, СУБД MySQL прекращает выполнение SELECT-запросов, которые предположительно будут выполняться долго (т. е. для которых оптимизатор запросов ожидает, что количество проверяемых строк превысит значение системной переменной max_join_size). По умолчанию значение переменной установлено в 1, что разрешает выполнять любые операторы SELECT.
- SQL_BUFFER_RESULT = {0 | 1}. Переменная принимает либо значение 1, либо 0 (по умолчанию). Установка переменной в значение 1 требует от СУБД MySQL помещать результат SELECT-запросов во временные таблицы. Это поможет MySQL раньше освободить блокировки таблиц и окажется полезным в случаях, когда требуется значительное время для пересылки результирующей таблицы клиенту.
- SQL_LOG_BIN = {0 | 1}. Переменная принимает либо значение 1 (по умолчанию), либо 0. Если значение переменной сбрасывается в 0, то для клиента прекращает выполняться бинарная регистрация. Для управления данной системной переменной клиент должен иметь привилегию SUPER.
- SQL_LOG_OFF = {0 | 1}. Переменная принимает либо значение 1, либо 0 (по умолчанию). При установке данной переменной в 1 запросы текущего клиента не регистрируются в общем журнале. Для управления данной системной переменной клиент должен иметь привилегию SUPER.
- SQL_LOG_UPDATE = {0 | 1}. Переменная принимает либо значение 1 (по умолчанию), либо 0. Если значение переменной устанавливается в 0, регистрация запросов клиента в журнале обновлений не производится. Для управления данной системной переменной клиент должен иметь привилегию SUPER.

- `SQL_QUOTE_SHOW_CREATE = {0 | 1}`. Переменная принимает либо значение 1 (по умолчанию), либо 0. Если значение переменной устанавливается в 1, то оператор `SHOW CREATE TABLE` будет заключать в обратные кавычки имена таблиц и столбцов, иначе все имена выводятся без обратных кавычек.
- `SQL_SAFE_UPDATES = {0 | 1}`. Переменная принимает либо значение 1, либо 0 (по умолчанию). Если значение переменной устанавливается в 1, СУБД MySQL прерывает операторы `UPDATE` и `DELETE`, у которых отсутствует предложение `WHERE` с участием индексированных полей или конструкция `LIMIT`. Это позволяет перехватить ошибочные операторы `UPDATE` и `DELETE`, которые могут случайно изменить или удалить большое количество строк.
- `SQL_SELECT_LIMIT = {`*num* `| DEFAULT}`. Переменная позволяет установить максимальное значение *num*, которое может вернуть оператор `SELECT`. По умолчанию устанавливается значение "неограниченно", передача в качестве значения переменной ключевого слова `DEFAULT` позволяет установить значение по умолчанию. Конструкция `LIMIT` в операторе `SELECT` имеет больший приоритет по сравнению с переменной `SQL_SELECT_LIMIT`.
- `SQL_WARNINGS = {0 | 1}`. Переменная принимает значение либо 1, либо 0 (по умолчанию) и управляет режимом вывода предупреждений, генерируемых оператором `INSERT`. Если значение переменной устанавливается в 1, то число добавленных строк выводится даже для однострочного оператора `INSERT`; если значение переменной `SQL_WARNINGS` принимает значение 0, то число добавленных строк выводится только для многострочного оператора `INSERT`.
- `TIMESTAMP = {`*timestamp* `| DEFAULT}`. Устанавливает текущее время для клиента, которое требуется при обработке журнала обновлений. Значение *timestamp* должно быть временем в формате UNIX, т. е. показывать число секунд, прошедших с полуночи 1 января 1970 года.
- `UNIQUE_CHECKS = {0 | 1}`. Переменная принимает значение либо 1 (по умолчанию), либо 0 и управляет проверкой уникальности вторичных индексов в таблицах InnoDB. Если значение переменной устанавливается в 0, никакие проверки уникальности не выполняются.

Среди параметров оператора `SET` выделяется `ONE_SHOT`, который не имеет самостоятельного значения, но влияет на состояние системных переменных. Этот модификатор применяется совместно с другими параметрами из рассмотренного выше списка и устанавливает их действие на один ближайший оператор (оператор, отличный от `SET`). Применять его можно только с параметрами, влияющими на кодировки, сортировки и временные зоны. Если осуществляется попытка использовать модификатор `ONE_SHOT` совместно с другими параметрами, возникает ошибка (листинг 29.10).

Замечание

Модификатор `ONE_SHOT` доступен в MySQL, начиная с версии 5.0. Он используется главным образом для внутренних целей MySQL.

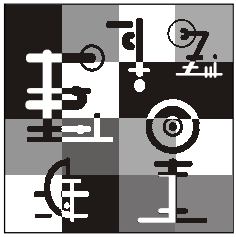
Листинг 29.10. Ошибочное использование модификатора ONE_SHOT

```
mysql> SET ONE_SHOT max_allowed_packet = 1;
ERROR 1382 (HY000): The 'SET ONE_SHOT' syntax is reserved for purposes
internal to the MySQL server
```

В листинге 29.11 приводится пример использования модификатора ONE_SHOT совместно с переменными character_set_connection и collation_connection.

Листинг 29.11. Использование модификатора ONE_SHOT в операторе SET

```
mysql> SET ONE_SHOT character_set_connection = cp1251;
mysql> SET ONE_SHOT collation_connection = cp1251_general_ci;
mysql> SHOW VARIABLES LIKE '%_connection';
+--------------------------+-------------------+
| Variable_name            | Value             |
+--------------------------+-------------------+
| character_set_connection | cp1251            |
| collation_connection     | cp1251_general_ci |
+--------------------------+-------------------+
mysql> SHOW VARIABLES LIKE '%_connection';
+--------------------------+-------------------+
| Variable_name            | Value             |
+--------------------------+-------------------+
| character_set_connection | latin1            |
| collation_connection     | latin1_swedish_ci |
+--------------------------+-------------------+
```



Глава 30

Оператор *SHOW*

Оператор `SHOW` генерирует многочисленные отчеты, по которым можно выяснить возможности текущей версии MySQL, значения отдельных системных переменных или просто содержимое той или иной базы данных или таблицы.

Оператор `SHOW`, как и большинство административных операторов, не является стандартным SQL-оператором и применяется только в СУБД MySQL. Оператор принимает множество форм:

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
SHOW CREATE DATABASE db_name
SHOW CREATE PROCEDURE procname
SHOW CREATE TABLE tbl_name
SHOW CREATE VIEW view_name
SHOW DATABASES [LIKE 'pattern']
SHOW ENGINE engine_name {LOGS | STATUS }
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW INNODB STATUS
SHOW [BDB] LOGS
SHOW PLUGIN
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW [GLOBAL | SESSION] STATUS [LIKE 'pattern']
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
SHOW [OPEN] TABLES [FROM db_name] [LIKE 'pattern']
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']
SHOW WARNINGS [LIMIT [offset,] row_count]
```

Все они будут рассматриваться в этой главе, остальные формы оператора SHOW будут рассмотрены в последующих главах:

```
SHOW TRIGGERS
SHOW PROCEDURE STATUS [LIKE 'pattern']
SHOW FUNCTION STATUS [LIKE 'pattern']
SHOW CREATE EVENT event_name
SHOW CREATE FUNCTION funcname
SHOW [FULL] EVENTS
SHOW BINLOG EVENTS
SHOW MASTER LOGS
SHOW MASTER STATUS
SHOW SLAVE HOSTS
SHOW SLAVE STATUS
```

30.1. Оператор *SHOW CHARACTER SET*

Оператор SHOW CHARACTER SET выводит список всех кодировок, которые поддерживает текущая версия MySQL (см. листинг 12.4). Данный оператор допускает необязательную конструкцию LIKE, которая позволяет задать шаблон для кодировок (листинг 30.1).

> ***Замечание***
>
> Оператор SHOW CHARACTER SET появился в СУБД MySQL, начиная с версии 4.1.0. Более подробно с кодировками можно ознакомиться в *главе 14*.

Листинг 30.1. Использование оператора SHOW CHARACTER SET

```
mysql> 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 результирующей таблицы содержит краткое название кодировки, используемое во всех командах MySQL, а второй столбец демонстрирует полное название кодировки, третий столбец — правила сравнения, которые назначаются для данной кодировки по умолчанию. Последний столбец содержит максимальное число байтов, отводимых под одну букву в кодировке.

Замечание

Результирующая таблица, которая выдается в ответ на запрос `SHOW CHARACTER SET`, дублирует представление `INFORMATION_SCHEMA.CHARACTER_SETS` *(см. разд. 36.1)*.

30.2. Оператор *SHOW COLLATION*

При сортировке национального текста, например, русского требуется задать порядок сравнения букв, например, 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. Названия сортировок можно узнать при помощи оператора `SHOW COLLATION`, который, как и `SHOW CHARACTER SET`, поддерживает конструкцию `LIKE` (листинг 30.2).

Замечание

Оператор `SHOW COLLATION` появился в СУБД MySQL, начиная с версии 4.1.0. Более подробно с кодировками можно ознакомиться в *главе 14*.

Листинг 30.2. Использование оператора `SHOW COLLATION`

```
mysql> SHOW COLLATION LIKE 'cp%';
+----------------------+---------+----+---------+----------+---------+
| Collation            | Charset | Id | Default | Compiled | Sortlen |
+----------------------+---------+----+---------+----------+---------+
| cp850_general_ci     | cp850   |  4 | Yes     |          |       0 |
| cp850_bin            | cp850   | 80 |         |          |       0 |
| cp1250_general_ci    | cp1250  | 26 | Yes     |          |       0 |
| cp1250_czech_cs      | cp1250  | 34 |         | Yes      |       2 |
| cp1250_croatian_ci   | cp1250  | 44 |         |          |       0 |
| cp1250_bin           | cp1250  | 66 |         |          |       0 |
| cp866_general_ci     | cp866   | 36 | Yes     |          |       0 |
| cp866_bin            | cp866   | 68 |         |          |       0 |
| cp852_general_ci     | cp852   | 40 | Yes     |          |       0 |
| cp852_bin            | cp852   | 81 |         |          |       0 |
| 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 |
| cp1256_general_ci    | cp1256  | 57 | Yes     |          |       0 |
| cp1256_bin           | cp1256  | 67 |         |          |       0 |
| cp1257_lithuanian_ci | cp1257  | 29 |         |          |       0 |
| cp1257_bin           | cp1257  | 58 |         |          |       0 |
| cp1257_general_ci    | cp1257  | 59 | Yes     |          |       0 |
| cp932_japanese_ci    | cp932   | 95 | Yes     | Yes      |       1 |
| cp932_bin            | cp932   | 96 |         | Yes      |       1 |
+----------------------+---------+----+---------+----------+---------+
```

Ключевое слово `LIKE` является необязательным и может быть опущено — результатом будет полный список сортировок. Первый столбец `Collation` содержит названия сортировок, второй столбец `Charset` — название кодировки, к которой относится текущая сортировка. Как видно из листинга, каждой кодировке соответствует несколько сортировок. Поле `Id` содержит уникальный идентификатор комбинации кодировка/сортировка. Если столбец `Default` отображает значение `Yes`, это означает, что данная сортировка назначается по умолчанию для текущей кодировки. Если столбец `Compiled` содержит значение `Yes`, это означает, что данная кодировка заранее скомпилирована на сервере. Столбец `Sortlen` сообщает о дополнительном числе байтов на символ, необходимых для сортировки текста с используемой кодировкой.

> ***Замечание***
>
> Результирующая таблица, которая выдается в ответ на запрос `SHOW COLLATION`, дублирует представление `INFORMATION_SCHEMA.COLLATIONS` *(см. разд. 36.2)*.

30.3. Оператор *SHOW COLUMNS*

Оператор `SHOW COLUMNS` выводит список столбцов заданной таблицы (листинг 30.3) и 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***
>
> Более полную информацию по столбцам можно извлечь из представления `INFORMATION_SCHEMA.COLUMNS` *(см. разд. 36.5)*.

Листинг 30.3. Использование оператора SHOW COLUMNS

```
mysql> SHOW COLUMNS FROM catalogs;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| id_catalog | int(11)  | NO   | PRI | NULL    | auto_increment |
| name       | tinytext | NO   |     |         |                |
+------------+----------+------+-----+---------+----------------+
```

Как видно из листинга 30.3, результирующая таблица оператора SHOW COLUMNS содержит число строк, равное числу столбцов таблицы catalogs, и включает следующие шесть столбцов:

- Field — имя столбца таблицы;
- Type — тип столбца;
- Null — информация о том, может ли столбец содержать значения NULL, принимает значение YES (да) и NO (нет);
- Key — показывает, индексирован столбец (PRI, MUL, UNI) или нет (пустое значение);
- Default — значение по умолчанию для столбца;
- Extra — дополнительные сведения о столбце, например, в нем содержится информация, снабжен ли столбец атрибутом AUTO_INCREMENT.

Вместо дополнительного ключевого слова FROM допускается использование расширенного имени таблицы. Так, представленные в листинге 30.4 запросы являются эквивалентными.

Листинг 30.4. Использование ключевого слова FROM

```
mysql> SHOW COLUMNS FROM catalogs FROM shop;
mysql> SHOW COLUMNS FROM shop.catalogs;
```

> ***Замечание***
>
> Оператор SHOW COLUMNS имеет два синонима — SHOW FIELDS и DESCRIBE.

Ключевое слово FULL позволяет вывести дополнительную информацию, связанную с привилегиями и комментариями столбцов (листинг 30.5). Помимо представленных выше, 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 — добиться этого можно за счет передачи клиенту `mysql` параметра `--vertical` или `-E` *(см. разд. 3.1)*. Результат продемонстрирован в листинге 30.5. В вертикальном представлении каждая строка начинается с серии звездочек, посередине которых указан номер строки, после чего друг под другом указываются имена столбцов результирующей таблицы и соответствующие им значения.

Листинг 30.5. Использование ключевого слова `FULL`

```
mysql> SHOW FULL COLUMNS FROM catalogs;
*************************** 1. row ***************************
     Field: id_catalog
      Type: int(11)
 Collation: NULL
      Null: NO
       Key: PRI
   Default: NULL
     Extra: auto_increment
Privileges: select,insert,update,references
   Comment:
*************************** 2. row ***************************
     Field: name
      Type: tinytext
 Collation: cp1251_general_ci
      Null: NO
       Key:
   Default:
     Extra:
Privileges: select,insert,update,references
   Comment:
```

Необязательное ключевое слово `LIKE`, так же как и в случае предыдущих операторов, позволяет вывести информацию только для столбцов, удовлетворяющих шаблону (листинг 30.6).

Листинг 30.6. Использование ключевого слова LIKE

```
mysql> SHOW COLUMNS FROM catalogs LIKE '%catalog';
+------------+---------+------+-----+---------+----------------+
| Field      | Type    | Null | Key | Default | Extra          |
+------------+---------+------+-----+---------+----------------+
| id_catalog | int(11) | NO   | PRI | NULL    | auto_increment |
+------------+---------+------+-----+---------+----------------+
```

30.4. Оператор *SHOW CREATE DATABASE*

Оператор `SHOW CREATE DATABASE` выводит оператор `CREATE DATABASE` для базы данных *dbase* и имеет следующий синтаксис:

```
SHOW CREATE DATABASE dbase
```

> ***Замечание***
>
> Добиться вертикального представления можно не только запуском клиента `mysql` с параметром `--vertical`, но и добавив в конец запроса символ `\G` (листинг 30.7). Подробнее данный параметр обсуждается в *разд. 3.1*.

Листинг 30.7. Использование оператора SHOW CREATE DATABASE

```
mysql> SHOW CREATE DATABASE shop\G;
*************************** 1. row ***************************
       Database: shop
Create Database: CREATE DATABASE `shop` /*!40100 DEFAULT CHARACTER SET
cp1251 */
```

Комментарий в конце оператора `CREATE DATABASE` начинается со служебной последовательности `/*!40100`, которая сообщает, что операторы в комментарии следует добавлять только в случае, если текущая версия сервера MySQL выше или равна 4.1.0. Это связано с тем, что для базы данных установка кодировки по умолчанию при помощи конструкции `DEFAULT CHARACTER SET` предусмотрена только начиная с версии 4.1.0.

30.5. Оператор *SHOW CREATE TABLE*

Оператор `SHOW CREATE TABLE` (листинг 30.8) выводит оператор `CREATE TABLE` для таблицы `tbl` и имеет следующий синтаксис:

```
SHOW CREATE TABLE tbl
```

Листинг 30.8. Использование оператора SHOW CREATE TABLE

```
mysql> 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=InnoDB DEFAULT CHARSET=cp1251 CHECKSUM=1
```

Оператор `SHOW CREATE TABLE` заключает в кавычки имена таблицы и столбцов. Отключить такое поведение можно, установив значение системной переменной `SQL_QUOTE_SHOW_CREATE` равным 0. Подробнее с данной переменной и методами работы с ней можно ознакомиться в *главе 29*.

30.6. Оператор *SHOW DATABASES*

Оператор `SHOW DATABASES` (листинг 30.9) выводит список баз данных, находящихся под управлением сервера MySQL, и имеет следующий синтаксис:

```
SHOW DATABASES [LIKE 'pattern']
```

Если пользователь, запускающий данный оператор, не имеет глобальной привилегии `SHOW DATABASES`, то он увидит только те базы данных, для доступа к которым он имеет привилегии.

> ***Замечание***
>
> Более полную информацию по базам данных можно извлечь из представления `INFORMATION_SCHEMA.SCHEMATA` *(см. разд. 36.9)*.

Листинг 30.9. Использование оператора SHOW DATABASES

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

Ключевое слово `LIKE` позволяет задать шаблон соответствия имени базы данных (листинг 30.10).

Листинг 30.10. Использование ключевого слова `LIKE`

```
mysql> SHOW DATABASES LIKE '%t';
+--------------+
| Database (%t) |
+--------------+
| qwert         |
| test          |
| wet           |
+--------------+
```

30.7. Оператор *SHOW ENGINES*

Оператор `SHOW ENGINES` (листинг 30.11) выводит информацию о типах таблиц СУБД MySQL и имеет следующий синтаксис:

```
SHOW [STORAGE] ENGINES
```

> ***Замечание***
>
> Оператор `SHOW ENGINES` введен в СУБД MySQL, начиная с версии 4.1.2.

Листинг 30.11. Использование оператора `SHOW ENGINES`

```
mysql> SHOW ENGINES;
+------------+---------+----------------------------------------------+
| Engine     | Support | Comment                                      |
+------------+---------+----------------------------------------------+
| MyISAM     | YES     | Default engine as of MySQL 3.23 with great
                         performance                                  |
| MEMORY     | YES     | Hash based, stored in memory, useful for
                         temporary tables                             |
| HEAP       | YES     | Alias for MEMORY                             |
| MERGE      | YES     | Collection of identical MyISAM tables        |
| MRG_MYISAM | YES     | Alias for MERGE                              |
| ISAM       | NO      | Obsolete storage engine, now replaced by MyISAM|
| MRG_ISAM   | NO      | Obsolete storage engine, now replaced by MERGE |
| InnoDB     | DEFAULT | Supports transactions, row-level locking, and
                         foreign keys                                 |
```

```
| INNOBASE   | YES     | Alias for INNODB                                  |
| BDB        | NO      | Supports transactions and page-level locking      |
| BERKELEYDB | NO      | Alias for BDB                                     |
| NDBCLUSTER | NO      | Clustered, fault-tolerant, memory-based tables    |
| NDB        | NO      | Alias for NDBCLUSTER                              |
| EXAMPLE    | NO      | Example storage engine                            |
| ARCHIVE    | NO      | Archive storage engine                            |
| CSV        | NO      | CSV storage engine                                |
| FEDERATED  | NO      | Federated MySQL storage engine                    |
| BLACKHOLE  | NO      | /dev/null storage engine (anything you write
                         to it disappears)                 |
+------------+---------+---------------------------------------------------+
```

Замечание

Необязательное ключевое слово `STORAGE` не несет дополнительной смысловой нагрузки.

Замечание

Оператор `SHOW ENGINES` имеет синоним `SHOW TABLE TYPES`.

Оператор `SHOW ENGINES` удобно использовать для просмотра типов таблиц, доступных в текущей версии MySQL. Название типа приводится в столбце `Engine` результирующей таблицы, столбец `Support` может принимать три значения:

- `YES` — данный тип таблиц включен в состав сервера и может использоваться в операторах `CREATE TABLE` и `ALTER TABLE`;
- `NO` — данный тип таблиц не включен в состав сервера и не может использоваться в операторах `CREATE TABLE` и `ALTER TABLE`;
- `DEFAULT` — данный тип таблиц включен в состав сервера и выбран в качестве типа по умолчанию. Это означает, что если при создании таблицы ее тип не будет указан при помощи ключевого слова `ENGINE` или `TYPE`, данный тип будет автоматически назначен таблице.

Последний столбец `Comment` содержит краткий комментарий для каждой из представленных таблиц.

Замечание

Подробнее познакомиться с типами таблиц можно в *главе 11*.

30.8. Оператор *SHOW ENGINE*

Оператор `SHOW ENGINE` выводит статусную информацию, журнальную информацию обращения к таблице *`engine_name`* и имеет следующий синтаксис:

```
SHOW ENGINE engine_name {LOGS | STATUS}
```

Данный оператор поддерживает не все типы таблиц и в настоящий момент может применяться только в следующих двух формах:

```
SHOW ENGINE BDB LOGS
SHOW ENGINE INNODB STATUS
```

> **Замечание**
>
> В остальных комбинациях оператор `SHOW ENGINE` возвращает ошибку.

> **Замечание**
>
> Оператор `SHOW ENGINE` появился в СУБД MySQL, начиная с версии 4.1.2. Данный оператор заменяет собой устаревшие операторы `SHOW [BDB] LOGS` и `SHOW INNODB STATUS`, которые в настоящий момент исключены из СУБД MySQL.

Оператор `SHOW ENGINE BDB LOGS` возвращает информацию о журнальных файлах (log-файлах) для BDB-таблиц. Результирующая таблица содержит следующие столбцы:

- `File` — полный путь к файлу журнала (log-файлу);
- `Type` — тип файла журнала (log-файла);
- `Status` — статус, который может принимать два состояния: `FREE`, если файл может быть удален, и `IN USE`, если файл занят.

Оператор `SHOW ENGINE INNODB STATUS` отображает подробную статистическую информацию об InnoDB-таблицах.

30.9. Оператор *SHOW ERRORS*

Оператор `SHOW ERRORS` выводит список ошибок, которые возникли при выполнении последнего SQL-запроса, и имеет следующий синтаксис:

```
SHOW ERRORS [LIMIT [offset,] num]
```

> **Замечание**
>
> Оператор `SHOW ERRORS` является урезанной версией `SHOW WARNINGS`, который, кроме ошибок, выводит также предупреждения и замечания. Синтаксис оператора `SHOW WARNINGS` рассмотрен в *разд. 30.19*.

> **Замечание**
>
> Оператор `SHOW ERRORS` введен в СУБД MySQL, начиная с версии 4.1.0.

Возвращаясь к учебной базе данных `shop` электронного магазина, попробуем осуществить вставку новых записей в таблицу `catalogs`, используя для этого уже существующие записи таблицы (листинг 30.12).

Листинг 30.12. Ошибка при вставке новых записей в таблицу `catalogs`

```
mysql> INSERT INTO catalogs SELECT * FROM catalogs;
ERROR 1062: Duplicate entry '1' for key 1
```

Как видно из листинга, запрос завершился ошибкой, заключающейся в том, что СУБД MySQL не может добавить в таблицу запись с первичным ключом, равным 1, т. к. такая запись уже существует.

В листинге 30.13 выводится список ошибок при помощи оператора `SHOW ERRORS` — это может быть удобно, если номера и описания ошибок необходимо узнать из прикладной программы, обращающейся к серверу MySQL. Как видно из листинга, оператор возвращает сведения о двух ошибках. Узнать число ошибок можно при помощи запроса, представленного в листинге 30.14.

Листинг 30.13. Использование оператора `SHOW ERRORS`

```
mysql> SHOW ERRORS;
+-------+------+-----------------------------+
| Level | Code | Message                     |
+-------+------+-----------------------------+
| Error | 1062 | Duplicate entry '1' for key 1 |
| Error | 1105 | Unknown error               |
+-------+------+-----------------------------+
```

Листинг 30.14. Текущее число ошибок

```
mysql> SHOW COUNT(*) ERRORS;
+----------------------+
| @@session.error_count |
+----------------------+
|                    2 |
+----------------------+
```

Как видно из листинга 30.14, имя столбца результирующей таблицы является сеансовой переменной `@@session.error_count`, поэтому вернуть текущее число ошибок можно также при помощи запроса, представленного в листинге 30.15.

Листинг 30.15. Обращение к сеансовой переменной `@@session.error_count`

```
mysql> SELECT @@session.error_count;
+----------------------+
| @@session.error_count |
+----------------------+
|                    2 |
+----------------------+
```

Конструкция `LIMIT` имеет тот же синтаксис, что и в операторе `SELECT` *(см. разд. 7.4)*.

30.10. Оператор *SHOW GRANTS*

Оператор `SHOW GRANTS` выводит информацию о привилегиях (листинг 30.16), предоставленных учетной записи *`user`*, и имеет следующий синтаксис:

```
SHOW GRANTS FOR user
```

Учетная запись *`user`* состоит из имени пользователя *`username`* и хоста *`host`*, с которого пользователю разрешено обращаться к серверу MySQL — *`'username'@'host'`*.

> ***Замечание***
>
> Подробнее с учетными записями MySQL и их управлением можно ознакомиться в *главе 27*.

Листинг 30.16. Использование оператора `SHOW GRANTS`

```
mysql> SHOW GRANTS FOR 'wet'@'localhost'\G;
*************************** 1. row ***************************
Grants for wet@localhost: GRANT USAGE ON *.* TO 'wet'@'localhost'
IDENTIFIED BY PASSWORD '*196BDEDE2AE4F84CA44C47D54D78478C7E2BD7B7'
WITH MAX_QUERIES_PER_HOUR 1000 MAX_UPDATES_PER_HOUR 10
MAX_CONNECTIONS_PER_HOUR 10 MAX_USER_CONNECTIONS 3
*************************** 2. row ***************************
Grants for wet@localhost: GRANT ALL PRIVILEGES ON `shop`.* TO
'wet'@'localhost'
```

Как видно из листинга 30.16, для учетной записи `'wet'@'localhost'` были выполнены два оператора `GRANT`. Первый оператор предоставляет доступ ко всем базам данных, но не снабжает пользователя никакими полномочиями, а второй предоставляет доступ только к базе данных `shop`, но зато разрешает пользователю выполнять с этой базой любые операции. Таким образом, пользователь с учетной записью `'wet'@'localhost'` получает возможность просматривать список баз данных на сервере и работать с базой данных `shop`.

Начиная с версии 4.1.2, оператор `SHOW GRANTS` допускает упрощенный синтаксис для вывода информации о привилегиях текущего пользователя (листинг 30.17). Для решения этой задачи можно воспользоваться одним из представленных ниже запросов:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

Листинг 30.17. Использование упрощенной версии оператора `SHOW GRANTS`

```
mysql> SHOW GRANTS;
+-------------------------------------------------------------------+
| Grants for root@localhost                                         |
```

```
+-------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+-------------------------------------------------------------------+
```

До версии 4.1.2 для того чтобы определить учетную запись, с использованием которой было установлено соединение с сервером MySQL, необходимо было воспользоваться функцией `CURRENT_USER()` *(см. разд. 22.2.3)*. Результат функции затем необходимо было подставлять в оператор `SHOW GRANTS`.

30.11. Оператор *SHOW INDEX*

Оператор `SHOW INDEX` выдает информацию об индексах таблицы *`tbl`* и имеет следующий синтаксис:

```
SHOW INDEX FROM tbl [FROM dbase]
```

Необязательное дополнительное ключевое слово `FROM` в конце оператора позволяет уточнить имя базы данных, в которой располагается таблица *`tbl`*. В листинге 30.18 демонстрируется использование оператора `SHOW INDEX` применительно к таблице `products`.

> ***Замечание***
>
> Оператор `SHOW INDEX` имеет синоним `SHOW KEYS`.

> ***Замечание***
>
> Получить информацию по индексам и индексируемым столбцам можно при помощи представлений информационной схемы `STATISTICS` *(см. разд. 36.10)* и `KEY_COLUMN_USAGE` *(см. разд. 36.6)*, соответственно.

Листинг 30.18. Использование оператора `SHOW INDEX`

```
mysql> SHOW INDEX FROM products\G;
*************************** 1. row ***************************
       Table: products
  Non_unique: 0
    Key_name: PRIMARY
Seq_in_index: 1
 Column_name: id_product
   Collation: A
 Cardinality: 31
    Sub_part: NULL
      Packed: NULL
```

```
        Null:
  Index_type: BTREE
     Comment:
*************************** 2. row ***************************
       Table: products
  Non_unique: 1
    Key_name: id_catalog
Seq_in_index: 1
 Column_name: id_catalog
   Collation: A
 Cardinality: 5
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: BTREE
     Comment:
*************************** 3. row ***************************
       Table: products
  Non_unique: 1
    Key_name: search
Seq_in_index: 1
 Column_name: name
   Collation: NULL
 Cardinality: 1
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: FULLTEXT
     Comment:
*************************** 4. row ***************************
       Table: products
  Non_unique: 1
    Key_name: search
Seq_in_index: 2
 Column_name: description
   Collation: NULL
 Cardinality: 1
    Sub_part: NULL
```

```
      Packed: NULL
        Null: YES
  Index_type: FULLTEXT
     Comment:
```

Результирующая таблица, возвращаемая оператором `SHOW INDEX`, содержит 12 столбцов:

- `Table` — имя таблицы;
- `Non_unique` — поле принимает значение 0, если индекс является уникальным и не может иметь дублированных значений, в противном случае поле принимает значение 1;
- `Key_name` — имя индекса, первичный ключ не имеет имени, поэтому поле `Key_name` принимает значение `PRIMARY` (листинг 30.18);
- `Seq_in_index` — если индекс построен на нескольких столбцах, это поле содержит порядковый номер столбца в индексе. В листинге 30.18 для полнотекстового (`FULLTEXT`) индекса `search` выводятся две строки, поле `Seq_in_index` одной из которых содержит значение 1, а второй — 2;
- `Column_name` — имя индексированного столбца;
- `Collation` — поле информирует о том, как столбец сортируется в индексе. Данное поле может принимать три значения: `A` — восходящая сортировка, `D` — нисходящая сортировка, `NULL` — отсутствие сортировки;
- `Cardinality` — количество уникальных значений в индексе. Это значение обновляется при помощи оператора `ANALYZE TABLE`;
- `Sub_part` — длина префикса. Если индексируется только часть строки, например, первые 5 символов, значение этого поля будет равно 5. Если индексируется весь столбец без ограничений, поле `Sub_part` содержит значение `NULL`;
- `Packed` — поле информирует о способе упаковки индекса. Если индекс не упакован, возвращается значение `NULL`;
- `Null` — поле содержит значение `YES`, если столбец допускает в качестве значения `NULL`, иначе возвращается пустая строка;
- `Index_type` — поле информирует об использованном методе индексации (`BTREE`, `FULLTEXT`, `HASH`, `RTREE`);
- `Comment` — комментарий для столбца.

Вместо дополнительного ключевого слова `FROM` допускается использование расширенного имени таблицы. Так, представленные в листинге 30.19 запросы являются эквивалентными.

Листинг 30.19. Использование ключевого слова `FROM`

```
mysql> SHOW INDEX FROM catalogs FROM shop;
mysql> SHOW INDEX FROM shop.catalogs;
```

30.12. Оператор *SHOW PLUGIN*

Оператор `SHOW PLUGIN` отображает информацию об используемых в текущей версии MySQL плагинах (листинг 30.20).

> ***Замечание***
>
> Оператор `SHOW PLUGIN` был введен в СУБД MySQL, начиная с версии 5.1.5.

Листинг 30.20. Использование оператора `SHOW PLUGIN`

```
mysql> SHOW PLUGIN;
+------------+--------+----------------+---------+
| Name       | Status | Type           | Library |
+------------+--------+----------------+---------+
| MEMORY     | ACTIVE | STORAGE ENGINE | NULL    |
| MyISAM     | ACTIVE | STORAGE ENGINE | NULL    |
| InnoDB     | ACTIVE | STORAGE ENGINE | NULL    |
| ARCHIVE    | ACTIVE | STORAGE ENGINE | NULL    |
| CSV        | ACTIVE | STORAGE ENGINE | NULL    |
| BLACKHOLE  | ACTIVE | STORAGE ENGINE | NULL    |
| FEDERATED  | ACTIVE | STORAGE ENGINE | NULL    |
| MRG_MYISAM | ACTIVE | STORAGE ENGINE | NULL    |
+------------+--------+----------------+---------+
```

30.13. Оператор *SHOW PRIVILEGES*

Оператор `SHOW PRIVILEGES` (листинг 30.21) выводит список привилегий, поддерживаемых текущей версией СУБД MySQL.

> ***Замечание***
>
> Оператор введен в СУБД MySQL, начиная с версии 4.1.0.

Листинг 30.21. Использование оператора `SHOW PRIVILEGES`

```
mysql> SHOW PRIVILEGES\G;
*************************** 1. row ***************************
Privilege: Alter
  Context: Tables
  Comment: To alter the table
```

```
*************************** 2. row ***************************
Privilege: Alter routine
  Context: Functions,Procedures
  Comment: To alter or drop stored functions/procedures
*************************** 3. row ***************************
Privilege: Create
  Context: Databases,Tables,Indexes
  Comment: To create new databases and tables
*************************** 4. row ***************************
Privilege: Create routine
  Context: Functions,Procedures
  Comment: To use CREATE FUNCTION/PROCEDURE
*************************** 5. row ***************************
Privilege: Create temporary tables
  Context: Databases
  Comment: To use CREATE TEMPORARY TABLE
*************************** 6. row ***************************
Privilege: Create view
  Context: Tables
  Comment: To create new views
*************************** 7. row ***************************
Privilege: Create user
  Context: Server Admin
  Comment: To create new users
*************************** 8. row ***************************
Privilege: Delete
  Context: Tables
  Comment: To delete existing rows
*************************** 9. row ***************************
Privilege: Drop
  Context: Databases,Tables
  Comment: To drop databases, tables, and views
*************************** 10. row ***************************
Privilege: Execute
  Context: Functions,Procedures
  Comment: To execute stored routines
*************************** 11. row ***************************
Privilege: File
  Context: File access on server
```

```
  Comment: To read and write files on the server
*************************** 12. row ***************************
Privilege: Grant option
  Context: Databases,Tables,Functions,Procedures
  Comment: To give to other users those privileges you possess
*************************** 13. row ***************************
Privilege: Index
  Context: Tables
  Comment: To create or drop indexes
*************************** 14. row ***************************
Privilege: Insert
  Context: Tables
  Comment: To insert data into tables
*************************** 15. row ***************************
Privilege: Lock tables
  Context: Databases
  Comment: To use LOCK TABLES (together with SELECT privilege)
*************************** 16. row ***************************
Privilege: Process
  Context: Server Admin
  Comment: To view the plain text of currently executing querie
*************************** 17. row ***************************
Privilege: References
  Context: Databases,Tables
  Comment: To have references on tables
*************************** 18. row ***************************
Privilege: Reload
  Context: Server Admin
  Comment: To reload or refresh tables, logs and privileges
*************************** 19. row ***************************
Privilege: Replication client
  Context: Server Admin
  Comment: To ask where the slave or master servers are
*************************** 20. row ***************************
Privilege: Replication slave
  Context: Server Admin
  Comment: To read binary log events from the master
*************************** 21. row ***************************
Privilege: Select
```

```
  Context: Tables
  Comment: To retrieve rows from table
*************************** 22. row ***************************
Privilege: Show databases
  Context: Server Admin
  Comment: To see all databases with SHOW DATABASES
*************************** 23. row ***************************
Privilege: Show view
  Context: Tables
  Comment: To see views with SHOW CREATE VIEW
*************************** 24. row ***************************
Privilege: Shutdown
  Context: Server Admin
  Comment: To shut down the server
*************************** 25. row ***************************
Privilege: Super
  Context: Server Admin
  Comment: To use KILL thread, SET GLOBAL, CHANGE MASTER, etc.
*************************** 26. row ***************************
Privilege: Update
  Context: Tables
  Comment: To update existing rows
*************************** 27. row ***************************
Privilege: Usage
  Context: Server Admin
  Comment: No privileges - allow connect only
```

Результирующая таблица, возвращаемая оператором `SHOW PRIVILEGES`, имеет три столбца:

- `Privilege` — имя привилегии;
- `Context` — область действия привилегии, которая может принимать следующие значения: `Server Admin` — глобальный уровень (весь сервер), `Databases` — уровень базы данных, `Tables` — уровень таблицы, `Functions`, `Procedures` — уровень хранимых процедур и пользовательских функций, `Indexes` — уровень столбца;
- `Comment` — краткое описание привилегии.

Замечание

Полное описание системных привилегий приводится в табл. 27.1.

30.14. Оператор *SHOW PROCESSLIST*

Оператор `SHOW PROCESSLIST` выводит список работающих потоков сервера и имеет следующий синтаксис:

```
SHOW [FULL] PROCESSLIST
```

Только наличие привилегии `SUPER` позволяет видеть все потоки, в противном случае выводятся только те потоки, которые ассоциированы с учетной записью пользователя MySQL.

> ***Замечание***
>
> Необязательно ключевое слово `FULL` позволяет выводить больше информации в поле `Info` результирующей таблицы, в противном случае выводятся только первые 100 символов.

Этот оператор удобен при решении проблемы "слишком большого количества соединений", позволяя выяснить, где происходит "утечка" соединений (листинг 30.22).

Листинг 30.22. Использование оператора `SHOW PROCESSLIST`

```
mysql> SHOW PROCESSLIST;
+----+-------+----------------+------+---------+------+-------+---------+
| Id | User  | Host           | db   | Command | Time | State | Info    |
+----+-------+----------------+------+---------+------+-------+---------+
|  9 | root  | localhost:1489 | shop | Query   |    0 | NULL  | SHOW
                                                                PROCESSLIST |
| 11 | root  | localhost:1524 | NULL | Sleep   |  113 |       | NULL    |
| 13 | hello | localhost:1532 | test | Sleep   |    3 |       | NULL    |
| 14 | wet   | PHOTON:3027    | NULL | Sleep   |   23 |       | NULL    |
| 15 | wet   | DRAGON:3046    | NULL | Sleep   |   46 |       | NULL    |
| 16 | wet   | ELF:3110       | forum| Sleep   |   11 |       | NULL    |
+----+-------+----------------+------+---------+------+-------+---------+
```

Каждая строка результирующей таблицы, возвращаемой оператором `SHOW PROCESSLIST`, соответствует одному соединению с сервером. Как видно из листинга 30.22, на момент выполнения команды с сервером установлено шесть соединений: два с использованием учетной записи `root@localhost`, одно с использованием учетной записи `hello@localhost`, три с использованием учетной записи `'wet'@'192.168.200.%'`. Причем каждое из соединений установлено с трех различных машин локальной сети: `PHOTON`, `DRAGON` и `ELF`.

Результирующая таблица содержит 8 полей, имеющих следующие значения:

- `Id` — идентификационный номер потока; зная этот номер, можно уничтожить данный поток при помощи команды `KILL` *(см. разд. 28.7)*;

- `User` — имя пользователя;
- `Host` — хост и порт клиента, с которого установлено соединение с сервером MySQL;
- `db` — база данных, с которой в настоящий момент работает клиент в этом соединении, если ни одна из баз данных не выбрана — возвращается `NULL`;
- `Command` — статус соединения, например, статус `Query` сообщает, что в настоящий момент в данном соединении выполняется запрос, статус `Sleep` означает, что пользователь установил, но не использует соединение. Значения, которые может принимать поле `Command`, обсуждаются далее;
- `Time` — время бездействия клиента в секундах;
- `State` — информация об операциях, которые MySQL выполняет в процессе обработки оператора SQL;
- `Info` — исполняемый запрос, если в запросе `SHOW PROCESSLIST` не указывается ключевое слово `FULL`, выводятся только первые 100 символов.

Поле `State` может принимать следующие значения:

- `Checking table` — поток осуществляет автоматическую проверку таблицы;
- `Closing tables` — означает, что поток записывает измененные данные таблиц на диск и закрывает использующиеся таблицы. Выполнение этой операции должно произойти быстро. Если на нее уходит значительное время, следует убедиться, что диск не переполнен или не используется слишком интенсивно;
- `Connect Out` — подчиненный сервер репликации, подсоединяется к головному серверу;
- `Copying to tmp table on disk` — набор временных результатов превысил `tmp_table_size`, и поток переносит временную таблицу на диск для экономии памяти;
- `Creating tmp table` — поток создает временную таблицу, чтобы хранить часть результата запроса;
- `Deleting from main table` — поток выполняет первую часть операции многотабличного удаления, удаляя данные из первой таблицы;
- `Deleting from reference tables` — поток выполняет вторую часть операции многотабличного удаления, удаляя соответствующие строки из других таблиц;
- `Flushing tables` — поток запускает команду `FLUSH TABLES` и ожидает, пока все потоки закроют свои таблицы;
- `Killed` — кто-то направил команду на закрытие потока, и поток будет закрыт при следующей проверке флага закрытия. Флаг проверяется при каждом основном цикле в MySQL, но в некоторых случаях закрытие потока может занять некоторое время. Если поток заблокирован другим потоком, закрытие будет произведено сразу после того, как другой поток снимет блокировку;
- `Query` — поток выполняет запрос;

- `Sending data` — поток обрабатывает строки для оператора `SELECT`, а также направляет данные клиенту;
- `Sorting for group` — поток осуществляет сортировку в соответствии с `GROUP BY`;
- `Sorting for order` — поток осуществляет сортировку в соответствии с `ORDER BY`;
- `Opening tables` — это просто означает, что поток пытается открыть таблицу. Такая процедура осуществляется довольно быстро, если что-либо не мешает открытию. Например, команды `ALTER TABLE` или `LOCK TABLE` могут помешать открытию таблицы, пока выполнение команды не будет завершено;
- `Removing duplicates` — запрос использовал команду `SELECT DISTINCT` таким образом, что MySQL-сервер не смог произвести оптимизацию на начальном этапе. Поэтому MySQL перед отправкой результатов клиенту должен выполнить дополнительное удаление всех дублирующихся строк;
- `Reopen table` — поток блокирует таблицу, но после этого получает извещение, что после блокировки структура таблицы изменилась. Он освобождает блокировку, закрывает таблицу, а затем вновь пытается открыть ее;
- `Repair by sorting` — код восстановления использует сортировку для создания индексов;
- `Repair with keycache` — код восстановления использует создание ключей один за другим, через кэш ключей. Это намного медленнее, чем `Repair by sorting`;
- `Searching rows for update` — поток осуществляет первую фазу — производит поиск всех совпадающих строк, чтобы затем обновить их. Это действие необходимо выполнить, если команда `UPDATE` изменяет индекс, который используется для поиска указанных строк;
- `Sleep` — поток ожидает, когда клиент направит ему новую команду;
- `System lock` — поток ожидает получения внешней системной блокировки таблицы. Если не используется несколько серверов MySQL, которые получают доступ к одним и тем же таблицам, системную блокировку можно отключить при помощи параметра `--skip-external-locking`;
- `Upgrading lock` — обработчик `INSERT DELAYED` пытается заблокировать таблицу, чтобы вставить строки;
- `Updating` — поток производит поиск строк, которые необходимо обновить, и обновляет их;
- `User Lock` — поток ожидает возврата `GET_LOCK()`;
- `Waiting for tables` — поток получил уведомление, что структура таблицы изменилась, и ему необходимо повторно открыть таблицу, чтобы получить новую структуру. Чтобы повторно открыть таблицу, он вынужден ожидать, пока ее не закроют все остальные потоки. Это уведомление выдается, если другой поток вос-

пользовался командой FLUSH TABLES или к таблице была применена одна из следующих команд: FLUSH TABLES, ALTER TABLE, RENAME TABLE, REPAIR TABLE, ANALYZE TABLE или OPTIMIZE TABLE;

- Waiting for handler insert — обработчик INSERT DELAYED завершил работу со всеми вставками и ожидает новых.

Большинство состояний соответствует очень быстрым операциям. Если поток останавливается в одном из этих состояний на длительное время, это может говорить о проблеме, на которой необходимо заострить внимание.

30.15. Оператор *SHOW STATUS*

Оператор SHOW STATUS выводит информацию о состоянии сервера и имеет следующий синтаксис:

```
SHOW [GLOBAL | SESSION] STATUS [LIKE 'pattern']
```

Частичный вывод оператора SHOW STATUS представлен в листинге 30.23. Необязательные ключевые слова GLOBAL и SESSION задают тип отображаемых переменных. Если указано ключевое слово GLOBAL, оператор отображает глобальные системные переменные для всего сервера; если задается ключевое слово SESSION, то отображаются переменные текущей сессии, которые каждый пользователь может переопределить при помощи оператора SET *(см. главу 29)*.

> ***Замечание***
>
> Необязательные ключевые слова GLOBAL и SESSION введены в синтаксис оператора SHOW STATUS, начиная с версии 5.0.2.

Листинг 30.23. Использование оператора SHOW STATUS

```
mysql> SHOW STATUS;
+--------------------------+----------+
| Variable_name            | Value    |
+--------------------------+----------+
| Aborted_clients          | 0        |
| Aborted_connects         | 0        |
| Bytes_received           | 83193    |
| Bytes_sent               | 419667   |
...
| Created_tmp_disk_tables  | 4        |
| Created_tmp_tables       | 13       |
| Created_tmp_files        | 3        |
...
```

```
| Threads_created           | 18       |
| Threads_connected         | 1        |
| Threads_running           | 1        |
| Uptime                    | 16043    |
+---------------------------+----------+
```

С конструкцией `LIKE` оператор `SHOW STATUS` выводит только те переменные, имена которых соответствуют шаблону (листинг 30.24).

Листинг 30.24. Использование ключевого слова `LIKE`

```
mysql> SHOW STATUS LIKE 'Key%';
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| Key_blocks_used    | 125   |
| Key_read_requests  | 16010 |
| Key_reads          | 125   |
| Key_write_requests | 0     |
| Key_writes         | 0     |
+--------------------+-------+
```

Замечание

Большинству переменных состояния присваивается значение 0.

Сервер MySQL поддерживает большое число переменных состояния, которые могут принимать следующие значения:

- `Aborted_clients` — количество соединений, прерванных по причине отключения клиента без корректного закрытия соединения;
- `Aborted_connects` — количество неудачных попыток соединения с сервером MySQL;
- `Binlog_chache_use` — количество транзакций, использовавших временный кэш бинарного журнала. Переменная состояния добавлена в MySQL, начиная с версии 4.1.2;
- `Binlog_chache_disk_use` — количество транзакций, использовавших временный кэш бинарного журнала, но превысивших значение `binlog_cache_size` и создавших временный файл для сохранения операторов из транзакций. Переменная состояния добавлена в MySQL, начиная с версии 4.1.2;
- `Bytes_received` — количество байтов, полученных от всех клиентов;
- `Bytes_sent` — количество байтов, отправленных всем клиентам;

- `Com_xxx` — количество запусков каждой команды *xxx*. Для каждого типа оператора существует собственная переменная. Например, `Com_delete` и `Com_insert` подсчитывают количество выполнений операторов `DELETE` и `INSERT`;
- `Connections` — количество попыток (удачных и неудачных) соединения с сервером MySQL;
- `Created_tmp_disk_tables` — количество неявных временных таблиц на диске, созданных во время выполнения операторов;
- `Created_tmp_files` — количество созданных временных файлов сервером MySQL;
- `Created_tmp_tables` — количество неявных временных таблиц в памяти, созданных во время выполнения операторов;
- `Delayed_insert_threads` — количество используемых потоков вставки данных в режиме `INSERT DELAYED`;
- `Delayed_writes` — количество строк, вставленных при помощи команды `INSERT DELAYED`;
- `Delayed_errors` — количество записанных при помощи команды `INSERT DELAYED` строк, в которых произошли какие-либо ошибки (возможно, `duplicate key`);
- `Flush_commands` — количество выполненных команд `FLUSH`;
- `Handler_commit` — количество внутренних команд `COMMIT`;
- `Handler_delete` — количество удалений строки из таблицы;
- `Handler_read_first` — количество считываний из индекса первой записи. Если это значение велико, то, по всей вероятности, сервер осуществляет много полных индексных сканирований, например, `SELECT col1 FROM foo`, предполагая, что столбец *col1* проиндексирован;
- `Handler_read_key` — количество запросов на чтение строки по ключу. Высокое значение переменной говорит о том, что запросы и таблицы проиндексированы надлежащим образом;
- `Handler_read_next` — количество запросов на чтение следующей строки в порядке расположения ключей. Это значение будет увеличиваться, если запрашивается индексный столбец с ограничениями по диапазону или если проводится индексное сканирование;
- `Handler_read_prev` — количество запросов на чтение предыдущей строки в порядке расположения ключей. В большинстве случаев используется для оптимизации `ORDER BY...DESC`;
- `Handler_read_rnd` — количество запросов на чтение строки, основанных на фиксированной позиции. Значение будет высоким, если выполняется много запросов, требующих сортировки результатов;

- `Handler_read_rnd_next` — количество запросов на чтение следующей строки из файла данных. Это значение будет высоким, если производится много сканирований таблиц. Обычно это означает, что ваши таблицы не проиндексированы надлежащим образом или ваши запросы не используют преимущества индексов;
- `Handler_rollback` — количество внутренних команд `ROLLBACK`;
- `Handler_update` — количество запросов на обновление строки в таблице;
- `Handler_write` — количество запросов на вставку строки в таблицу;
- `Key_blocks_used` — количество используемых блоков в кэше ключей;
- `Key_read_requests` — количество запросов на чтение блока ключей из кэша;
- `Key_reads` — количество физических считываний блока ключей с диска;
- `Key_write_requests` — количество запросов на запись блока ключей в кэш;
- `Key_writes` — количество физических записей блоков ключей на диск;
- `Max_used_connections` — максимальное количество одновременно используемых соединений;
- `Not_flushed_key_blocks` — блоки ключей в кэше ключей, которые были изменены, но еще не записаны на диск;
- `Not_flushed_delayed_rows` — количество строк, стоящих в очереди на запись в запросах `INSERT DELAY`;
- `Open_tables` — количество открытых таблиц;
- `Open_files` — количество открытых файлов;
- `Open_streams` — количество открытых потоков (в основном используется для журналирования);
- `Opened_tables` — количество открывавшихся таблиц;
- `Qcache_free_blocks` — количество свободных блоков памяти в кэше запросов;
- `Qcache_free_memory` — объем свободной для кэша запросов памяти;
- `Qcache_inserts` — количество добавленных в кэш запросов;
- `Qcache_lowmem_prunes` — количество запросов, удаленных из кэша по причине недостаточного объема памяти;
- `Qcache_not_cached` — количество запросов, не помещенных в кэш;
- `Qcache_queries_in_cache` — количество записанных в кэш запросов;
- `Questions` — количество отправленных на сервер запросов;
- `Rpl_status` — статус отказобезопасной репликации (пока не используется);
- `Select_full_join` — количество соединений (таблиц), в которых индексы не используются. Если данное значение не равно 0, следует тщательно проверить индексы ваших таблиц;

- `Select_full_range_join` — количество соединений, где был использован поиск по диапазону в справочной таблице;
- `Select_range` — количество соединений, для которых использовались диапазоны в первой таблице (обычно это значение не критично, даже если оно велико);
- `Select_scan` — количество соединений, в которых проводилось первое сканирование первой таблицы;
- `Select_range_check` — количество соединений без ключей, в которых проверка использования ключей производится после каждой строки (если это значение равно 0, необходимо внимательно проверить индексы своих таблиц);
- `Slave_open_temp_tables` — количество временных таблиц, открытых в настоящий момент потоком подчиненного компьютера;
- `Slave_running` — содержит значение `ON`, если это подчиненный компьютер, подключенный к головному компьютеру;
- `Slow_launch_threads` — количество потоков, создание которых заняло больше времени, чем указано в `slow_launch_time`;
- `Slow_queries` — количество запросов, обработка которых заняла больше времени, чем `long_query_time`;
- `Sort_merge_passes` — количество объединений, осуществленных алгоритмом сортировки. Если это значение велико, следует увеличить `sort_buffer_size`;
- `Sort_range` — количество сортировок, которые осуществлялись в списках;
- `Sort_rows` — количество отсортированных строк;
- `Sort_scan` — количество сортировок, осуществленных путем сканирования таблицы;
- `Ssl_xxx` — переменные, используемые SSL;
- `Table_locks_immediate` — количество запросов на немедленную блокировку таблицы;
- `Table_locks_waited` — количество запросов, когда немедленная блокировка не могла быть осуществлена, и требовалось время на ожидание. Если это значение велико и у вас есть проблемы с производительностью, сначала необходимо оптимизировать свои запросы, а затем либо разделить таблицы, либо использовать репликацию;
- `Threads_cached` — количество потоков в кэше потоков;
- `Threads_connected` — количество открытых в настоящий момент соединений;
- `Threads_created` — количество потоков, созданных для управления соединениями;
- `Threads_running` — количество не простаивающих потоков;
- `Uptime` — время в секундах, в течение которого сервер находится в работе.

30.16. Оператор *SHOW TABLE STATUS*

Оператор `SHOW TABLE STATUS` выводит информацию о таблицах базы данных *dbase* и имеет следующий синтаксис:

```
SHOW TABLE STATUS [FROM dbase] [LIKE 'pattern']
```

Если для указания базы данных не используется необязательное ключевое слово `FROM`, выводится информация о текущей базе данных, выбранной ранее при помощи оператора `USE`. В листинге 30.25 выводится информация о таблицах учебной базы данных `shop`.

> ***Замечание***
>
> Ключевое слово `LIKE` позволяет вывести информацию только для тех таблиц базы данных *dbase*, которые удовлетворяют шаблону *pattern*.

> ***Замечание***
>
> Информацию обо всех таблицах, доступных для текущего пользователя на сервере, можно получить из представления `TABLES` информационной схемы *(см. разд. 36.13)*.

Листинг 30.25. Использование оператора `SHOW TABLES STATUS`

```
mysql> SHOW TABLE STATUS\G;
*************************** 1. row ***************************
           Name: catalogs
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 12
 Avg_row_length: 1365
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 20
    Create_time: 2005-05-30 01:20:35
    Update_time: NULL
     Check_time: NULL
      Collation: cp1251_general_ci
       Checksum: NULL
 Create_options: checksum=1
        Comment: InnoDB free: 4096 kB
```

```
*************************** 2. row ***************************
           Name: orders
         Engine: MyISAM
        Version: 7
     Row_format: Fixed
           Rows: 2
 Avg_row_length: 25
    Data_length: 50
Max_data_length: 7036874417766399
   Index_length: 4096
      Data_free: 0
 Auto_increment: 21
    Create_time: 2005-07-08 23:47:57
    Update_time: 2005-07-08 23:47:57
     Check_time: 2005-07-08 23:47:57
      Collation: cp1251_general_ci
       Checksum: NULL
 Create_options:
        Comment:
*************************** 3. row ***************************
           Name: products
         Engine: MyISAM
        Version: 10
     Row_format: Dynamic
           Rows: 31
 Avg_row_length: 123
    Data_length: 3836
Max_data_length: 281474976710655
   Index_length: 7168
      Data_free: 0
 Auto_increment: 22
    Create_time: 2005-07-08 23:47:57
    Update_time: 2005-07-08 23:47:57
     Check_time: 2005-07-08 23:47:57
      Collation: cp1251_general_ci
       Checksum: 1706424754
 Create_options: checksum=1
        Comment:
```

```
*************************** 4. row ***************************
           Name: users
         Engine: MyISAM
        Version: 10
     Row_format: Dynamic
           Rows: 6
 Avg_row_length: 69
    Data_length: 416
Max_data_length: 281474976710655
   Index_length: 2048
      Data_free: 0
 Auto_increment: 23
    Create_time: 2005-07-08 23:47:57
    Update_time: 2005-07-08 23:47:57
     Check_time: 2005-07-08 23:47:57
      Collation: cp1251_general_ci
       Checksum: 2410629935
 Create_options: checksum=1
        Comment:
```

Каждая строка результирующей таблицы оператора `SHOW TABLE STATUS` соответствует одной таблице базы данных. Результирующая таблица содержит 18 столбцов:

- `Name` — имя таблицы;
- `Engine` — тип таблицы *(см. главу 11)*;
- `Version` — номер версии файла с расширением frm;
- `Row_format` — формат хранения строки (`Fixed`, `Dynamic` или `Compressed`);
- `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` — время создания таблицы;

- `Update_time` — время обновления файла данных;
- `Check_time` — время, когда таблица проверялась в последний раз;
- `Collation` — сортировка, которая используется в таблице;
- `Checksum` — контрольная сумма (если включено автоматическое ее вычисление);
- `Create_options` — дополнительные опции, использованные при создании таблицы оператором `CREATE TABLE`;
- `Comment` — комментарий, введенный при создании таблицы.

В поле `Comment` для таблиц типа InnoDB сообщается о свободном месте в табличном пространстве таблиц InnoDB. В листинге 30.25 в комментарии к таблице `catalogs` сообщается, что в табличном пространстве осталось 4 Мбайт.

Для таблиц типа MEMORY (HEAP) значение `Data_length`, `Max_data_lenght` и `Index_lenght` принимают приближенное значение объема выделенной памяти, т. к. алгоритм распределения памяти резервирует память большими кусками, чтобы уменьшить количество операций.

30.17. Оператор *SHOW TABLES*

Оператор `SHOW TABLES` выводит список всех таблиц базы данных *dbase* и имеет следующий синтаксис:

```
SHOW [FULL|OPEN] TABLES [FROM dbase] [LIKE 'pattern']
```

> ***Замечание***
>
> Оператор `SHOW TABLES` отображает только постоянные таблицы, в список не попадают временные таблицы.

> ***Замечание***
>
> Ключевое слово `LIKE` позволяет вывести информацию только для тех таблиц базы данных *dbase*, которые удовлетворяют шаблону *pattern*.

Если для указания базы данных не используется необязательное ключевое слово `FROM`, выводится информация о текущей базе данных, выбранной ранее при помощи оператора `USE`. В листинге 30.26 выводится информация о таблицах учебной базы данных `shop`.

Листинг 30.26. Использование оператора `SHOW TABLES`

```
mysql> SHOW TABLES;
+---------------+
| Tables_in_shop |
```

```
+----------------+
| catalogs       |
| orders         |
| products       |
| users          |
+----------------+
```

Оператор может использоваться в форме `SHOW OPEN TABLES` и отображать список открытых таблиц, зарегистрированных в кэше таблиц.

> ***Замечание***
>
> Ключевые слова `FROM` и `LIKE` при указании ключевого слова `OPEN` игнорируются.

Начиная с версии MySQL 5.0.1, допускается использование ключевого слова `FULL`, которое приводит к выводу дополнительного столбца в результирующей таблице (листинг 30.27), принимающего два значения:

- `BASE TABLE` — обычная таблица;
- `VIEW` — представление.

> ***Замечание***
>
> Подробнее с представлениями можно познакомиться в *главе 35*.

Листинг 30.27. Использование оператора `SHOW FULL TABLES`

```
mysql> SHOW FULL TABLES;
+----------------+------------+
| Tables_in_shop | Table_type |
+----------------+------------+
| cat            | VIEW       |
| catalogs       | BASE TABLE |
| letter_catalogs | BASE TABLE |
| orders         | BASE TABLE |
| products       | BASE TABLE |
| users          | BASE TABLE |
+----------------+------------+
```

> ***Замечание***
>
> Информацию обо всех таблицах, доступных текущему пользователю на сервере, можно получить из представления `TABLES` информационной схемы *(см. разд. 36.13)*.

30.18. Оператор *SHOW VARIABLES*

Оператор `SHOW VARIABLES` выводит значения системных переменных MySQL и имеет следующий синтаксис:

```
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']
```

Опция `GLOBAL` позволяет получить значения глобальных переменных, которые будут использоваться каждым новым соединением с сервером MySQL. Опция `SESSION` обеспечивает получение значений сеансовых переменных.

> ***Замечание***
>
> Ключевое слово `SESSION` имеет синоним `LOCAL`.

Изменить значения системных переменных можно при помощи оператора `SET`, синтаксис которого описан в *главе 29*. В листинге 30.28 приводится частичный вывод оператора `SHOW VARIABLES`.

Листинг 30.28. Использование оператора `SHOW VARIABLES`

```
mysql> 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                     |
+---------------------------------+---------------------------+
```

С конструкцией `LIKE` оператор выводит только те переменные, имена которых соответствуют шаблону (листинг 30.29).

Листинг 30.29. Использование ключевого слова LIKE

```
mysql> SHOW VARIABLES LIKE 'version%';
+-------------------------+----------------------+
| Variable_name           | Value                |
+-------------------------+----------------------+
| version                 | 5.0.6-beta-nt        |
| version_comment         | Official MySQL binary |
| version_compile_machine | ia32                 |
| version_compile_os      | Win32                |
+-------------------------+----------------------+
```

30.19. Оператор *SHOW WARNINGS*

Оператор `SHOW WARNINGS` выводит ошибки, предупреждения и другие замечания, которые появляются в результате выполнения последнего запроса. Оператор имеет следующий синтаксис:

```
SHOW WARNINGS [LIMIT [offset,] num]
```

> ***Замечание***
>
> Оператор `SHOW WARNINGS` является расширенной версией оператора `SHOW ERRORS`, который выводит только ошибки. Синтаксис оператора `SHOW ERRORS` рассмотрен в *разд. 30.9*.

> ***Замечание***
>
> Оператор `SHOW WARNINGS` введен в СУБД MySQL, начиная с версии 4.1.0.

Осуществим попытку удаления несуществующей таблицы `no_table` из базы данных `shop` (листинг 30.30).

Листинг 30.30. Удаление несуществующей таблицы

```
mysql> DROP TABLE IF EXISTS no_table;
mysql> SHOW WARNINGS;
+-------+------+-------------------------+
| Level | Code | Message                 |
+-------+------+-------------------------+
| Note  | 1051 | Unknown table 'no_table' |
+-------+------+-------------------------+
```

Результирующая таблица, возвращаемая оператором SHOW WARNINGS, состоит из трех столбцов:

- Level — уровень ошибки или предупреждения (замечание — Note, предупреждение — Warning, ошибка — Error);
- Code — код ошибки или предупреждения;
- Message — расшифровывающее сообщение.

Конструкция LIMIT имеет тот же синтаксис, что и в операторе SELECT *(см. разд. 7.4)*.

Создадим таблицу tbl при помощи запроса, представленного в листинге 30.31.

Листинг 30.31. Создание таблицы tbl

```
mysql> CREATE TABLE tbl (name TINYINT NOT NULL, cod CHAR(4)) TYPE=MyISAM;
mysql> SHOW WARNINGS\G;
*************************** 1. row ***************************
  Level: Warning
   Code: 1287
Message: 'TYPE=storage_engine' is deprecated; use 'ENGINE=storage_engine'
instead
```

Как видно из листинга 30.31, оператор CREATE TABLE генерирует одно предупреждение (Warning), сообщающее, что ключевое слово TYPE для указания типа таблиц устарело, и в текущей версии рекомендуется использовать для этих целей ключевое слово ENGINE.

Узнать число ошибок, предупреждений и замечаний, которые вернул последний оператор, можно при помощи запроса, представленного в листинге 30.32.

Листинг 30.32. Текущее число ошибок, предупреждений и замечаний

```
mysql> SELECT COUNT(*) WARNINGS;
+-------------------------+
| @@session.warning_count |
+-------------------------+
|                       1 |
+-------------------------+
```

Как видно из листинга 30.32, имя столбца результирующей таблицы является сеансовой переменной @@session.warning_count, поэтому вернуть текущее число ошибок, предупреждений и замечаний можно также при помощи запроса, представленного в листинге 30.33.

Листинг 30.33. Обращение к сеансовой переменной `@@session.warning_count`

```
mysql> SELECT @@session.warning_count;
+------------------------+
| @@session.warning_count |
+------------------------+
|                      1 |
+------------------------+
```

Теперь осуществим вставку новых записей в только что созданную таблицу `tbl` (листинг 30.34).

Листинг 30.34. Вставка новых записей в только что созданную таблицу `tbl`

```
mysql> INSERT INTO tbl VALUES(10, 'mysql'), (NULL, 'test'),
    -> (300, 'open source');
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 4
mysql> SHOW WARNINGS\G;
*************************** 1. row ***************************
  Level: Warning
   Code: 1265
Message: Data truncated for column 'cod' at row 1
*************************** 2. row ***************************
  Level: Warning
   Code: 1263
Message: Column set to default value; NULL supplied to NOT NULL column
'name' at row 2
*************************** 3. row ***************************
  Level: Warning
   Code: 1264
Message: Out of range value adjusted for column 'name' at row 3
*************************** 4. row ***************************
  Level: Warning
   Code: 1265
Message: Data truncated for column 'cod' at row 3
```

Записи намеренно выбраны не совсем корректные, в результате многострочный оператор `INSERT` генерирует целых четыре предупреждения. При вставке большого объема данных, особенно в пакетном режиме, может генерироваться большое число пред-

упреждений. Максимальное количество ошибок, сообщений и замечаний, которые сохраняются между вызовами операторов, задается системной переменной `max_error_count` и равно по умолчанию 64. В запросе, приведенном в листинге 30.35, оператор `ALTER TABLE` генерирует три предупреждения (по числу записей в таблице), но сохраняется только одно, т. к. системной переменной `max_error_count` присваивается значение 1.

Листинг 30.35. Ограничение системной переменной `max_error_count`

```
mysql> SHOW VARIABLES LIKE 'max_error_count';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_error_count | 64    |
+-----------------+-------+
mysql> SET max_error_count = 1;
mysql> ALTER TABLE tbl MODIFY cod CHAR;
Query OK, 3 rows affected (0.13 sec)
Records: 3  Duplicates: 0  Warnings: 3
mysql> SELECT @@warning_count;
+-----------------+
| @@warning_count |
+-----------------+
|               3 |
+-----------------+
mysql> SHOW WARNINGS;
+---------+------+-----------------------------------------+
| Level   | Code | Message                                 |
+---------+------+-----------------------------------------+
| Warning | 1265 | Data truncated for column 'cod' at row 1 |
+---------+------+-----------------------------------------+
```

Чтобы отключить сообщения вообще, следует присвоить системной переменной `max_error_count` значение 0. В этом случае `warning_count` по-прежнему показывает, сколько предупреждений было сгенерировано, но ни одно из них не сохраняется.

30.20. Утилита *mysqlshow*

Утилита `mysqlshow` предоставляет средства для быстрого просмотра имеющихся на сервере баз данных, их таблиц, столбцов таблиц и индексов.

Синтаксис вызова утилиты `mysqlshow` выглядит следующим образом:

```
mysqlshow [parameters] [db_name [tbl_name [col_name]]
```

Сразу после имени утилиты перечисляются параметры функции *parameters*, после чего указывается имя базы данных *db_name*, таблица *tbl_name* и столбец *col_name*. Если не указана ни одна база данных, отображаются все базы данных, просмотр которых разрешен текущему пользователю. Если не указана ни одна из таблиц, отображаются все таблицы базы данных *db_name*. Если не указан ни один столбец, отображаются все столбцы таблицы *col_name*.

В именах баз данных, таблиц и столбцов можно использовать шаблоны (*, ?, % или _). Символы * и ? преобразуются в символы % и _, соответственно. Точно так же, как и в операторе LIKE, символ % соответствует любому числу символов, а символ _ — одному произвольному символу.

> ***Замечание***
>
> Если имя базы данных содержит знак подчеркивания, то он должен быть экранирован обратным слешем.

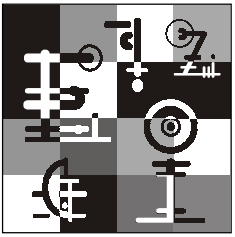
Утилита mysqlshow поддерживает параметры, перечисленные в табл. 30.1.

Таблица 30.1. *Параметры утилиты* mysqlshow

Параметр	Описание
--help, -?	Выводит страницу справки и завершает работу
--character-sets-dir=*path*	Позволяет передать путь к каталогу *path*, где установлены кодировки
--compess, -C	При использовании данного параметра вся информация, передаваемая между клиентом и сервером, подвергается сжатию
--debug[=*debug_options*], -# [*debug_options*]	Данный параметр включает вывод отладочной информации. Строка *debug_options* задает формат строки вывода, обычно принимает значение 'd:t:o,file_name'
--default-character-set=*charset*	В параметре задается кодировка *charset* по умолчанию
--host=*host_name*, -h *host_name*	В параметре указывается сетевое имя или IP-адрес компьютера *host_name*, на котором установлен MySQL-сервер
--keys, -k	При использовании данного параметра в отчете выводятся индексы таблиц
--password[=*password*], -p[*password*]	В данном параметре указывается пароль *password* для доступа к MySQL-серверу
--port=*port_num*, -P *port_num*	Параметр указывает номер порта *port_num*, по которому MySQL-сервер ожидает соединения с клиентом (по умолчанию 3306, если данный параметр не указан, производится попытка установить соединение именно по этому порту)

Таблица 30.1 *(окончание)*

Параметр	Описание
`--protocol={TCP|SOCKET|PIPE|MEMORY}`	Позволяет задать протокол подключения к серверу. Параметр введен, начиная с версии MySQL 4.1
`--socket=`*`path`*, `-S` *`path`*	Позволяет указать путь к файлу сокета, который будет использоваться для подключения к MySQL-серверу
`--status`, `-i`	При использовании данного параметра выводится дополнительная информация о каждой таблице
`--user=`*`user_name`*, `-u` *`user_name`*	Параметр указывает пользователя *`user_name`*, от имени которого осуществляется соединение с MySQL-сервером
`--verbose`, `-v`	При использовании данного параметра утилита работает в режиме расширенных сообщений и генерирует более подробные отчеты
`--version`, `-V`	При использовании данного параметра, утилита `mysqladmin` выводит информацию о версии сервера и завершает работу



Глава 31

Предотвращение катастроф и восстановление

СУБД является сложным образованием и требует постоянной поддержки и сопровождения. Поэтому проблеме восстановления данных следует уделить повышенное внимание. Для облегчения этой задачи база данных СУБД MySQL предоставляет развитые средства для анализа, восстановления и резервного копирования баз данных, которые будут рассмотрены в этой главе.

31.1. Оператор *CHECK TABLE*

Оператор `CHECK TABLE` проверяет таблицу или группу таблиц на наличие ошибок. Для таблиц MyISAM кроме всего прочего обновляется статистическая информация о ключах. Оператор `CHECK TABLE` работает только с таблицами MyISAM и BDB.

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

Пример запроса с использованием оператора `CHECK TABLE` представлен в листинге 31.1.

Листинг 31.1. Использование оператора `CHECK TABLE`

```
mysql> CHECK TABLE orders, users, products, catalogs;
+--------------+-------+----------+----------------------------------+
| Table        | Op    | Msg_type | Msg_text                         |
+--------------+-------+----------+----------------------------------+
| shop.orders  | check | warning  | 1 client is using or hasn't closed
                                    the table properly               |
| shop.orders  | check | status   | OK
|
| shop.users   | check | warning  | 1 client is using or hasn't closed
                                    the table properly               |
```

```
| shop.users    | check | status   | OK                                 |
| shop.products | check | warning  | Table is marked as crashed         |
| shop.products | check | status   | OK                                 |
| shop.catalogs | check | status   | OK                                 |
+---------------+-------+----------+------------------------------------+
```

Как видно из листинга 31.1, результирующая таблица оператора `CHECK TABLE` состоит из четырех столбцов. Столбец `Table` содержит расширенное имя таблицы, над которой была произведена операция.

> ***Замечание***
>
> Операторы `ANALYZE TABLE`, `OPTIMIZE TABLE`, `REPAIR TABLE` и `BACKUP TABLE` возвращают результат своей работы точно в таком же формате. Второй столбец результирующей таблицы `Op` сообщает, какой из операторов был использован.

Столбец `Op` отражает тип операции над таблицей и может принимать одно из шести значений:

- `check` — сообщает о том, что над таблицей проведена операция проверки при помощи оператора `CHECK TABLE`;
- `analyze` — таблица проверена при помощи оператора `ANALYZE TABLE`;
- `optimize` — таблица была оптимизирована при помощи оператора `OPTIMIZE TABLE`;
- `repair` — была произведена операция восстановления поврежденных таблиц при помощи оператора `REPAIR TABLE`;
- `backup` — была произведена операция резервного копирования таблицы при помощи оператора `BACKUP TABLE`;
- `restore` — была произведена операция восстановления из резервной копии при помощи оператора `RESTORE TABLE`.

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.

Для каждой из таблиц может быть выведено несколько сообщений. Так в листинге 31.1, помимо того, что выводится статусное сообщение (`status`), для таблиц `orders`

и `users` выводится предупреждение (`warning`) о том, что один из клиентов использует таблицы или они не были корректно закрыты. При открытии таблицы СУБД MySQL помечает ее специальным флагом, который сбрасывается при корректном завершении работы сервера. Несброшенный флаг сообщает серверу о необходимости проверки таблицы, именно на такой флаг и реагирует оператор `CHECK TABLE`.

Для таблицы `products` выводится предупреждение, что таблица может быть повреждена. Если оператор `CHECK TABLE` не находит никаких повреждений, он снимает эти отметки (листинг 31.2), однако если найдено существенное повреждение, оператор `CHECK TABLE` сообщает о повреждении и при повторном использовании. В этом случае следует попытаться восстановить таблицу при помощи оператора `REPAIR TABLE`.

> ***Замечание***
>
> В большинстве случаев повреждение таблицы связано с повреждениями индексов, а не данных — такого рода повреждения легко устраняются.

Листинг 31.2. Повторный вызов оператора `CHECK TABLE`

```
mysql> CHECK TABLE orders, users, products, catalogs;
+---------------+-------+----------+---------------------------+
| Table         | Op    | Msg_type | Msg_text                  |
+---------------+-------+----------+---------------------------+
| shop.orders   | check | status   | OK                        |
| shop.users    | check | status   | OK                        |
| shop.products | check | warning  | Table is marked as crashed |
| shop.products | check | status   | OK                        |
| shop.catalogs | check | status   | OK                        |
+---------------+-------+----------+---------------------------+
```

В конце оператора `CHECK TABLE` может быть указана одна из следующих опций 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 31.3 приводится пример использования опции QUICK совместно с оператором CHECK TABLE.

Листинг 31.3. Использование опций в операторе CHECK TABLE

```
mysql> CHECK TABLE orders, users, products, catalogs 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 31.4.

Листинг 31.4. Использование комбинации опций

```
mysql> CHECK TABLE orders, users, products, catalogs FAST QUICK;
```

Запрос в листинге 31.4 выполняет быструю проверку таблицы на предмет того, была ли она закрыта правильно.

Опции FAST и CHANGED наиболее подходят для применения в сценариях, например, запускаемых по расписанию при помощи UNIX-демона cron.

31.2. Оператор *ANALYZE TABLE*

Оператор ANALYZE TABLE анализирует и сохраняет распределение индексов таблицы (листинг 31.5). Это позволяет MySQL использовать в дальнейшем сохраненные данные для того, чтобы принимать решения относительно порядка объединения таблиц в многотабличных запросах.

> ***Замечание***
>
> Данный оператор может применяться к таблицам типа MyISAM, BDB и InnoDB. На время выполнения оператора ANALYZE TABLE доступ к таблице блокируется.

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

Листинг 31.5. Использование оператора ANALYZE TABLE

```
mysql> ANALYZE TABLES catalogs, products;
+---------------+---------+----------+--------------------------+
| Table         | Op      | Msg_type | Msg_text                 |
+---------------+---------+----------+--------------------------+
| shop.catalogs | analyze | status   | OK                       |
| shop.products | analyze | status   | Table is already up to date |
+---------------+---------+----------+--------------------------+
```

Формат результирующей таблицы оператора `ANALYZE TABLE` совпадает с описанным в предыдущем разделе форматом для оператора `CHECK TABLE`. В случае успешного обновления индексов в четвертом столбце `Msg_text` появляется сообщение `OK`. Если с момента предыдущего применения оператора `ANALYZE TABLE` таблица не изменилась, то в столбце `Msg_text` будет присутствовать сообщение `Table is already up to date`.

31.3. Оператор *CHECKSUM TABLE*

Оператор `CHECKSUM TABLE` (листинг 31.6) возвращает контрольную сумму таблицы (или таблиц).

> ***Замечание***
>
> Оператор `CHECKSUM TABLE` добавлен в MySQL, начиная с версии 4.1.1.

Листинг 31.6. Использование оператора `CHECKSUM TABLE`

```
mysql> CHECKSUM TABLE catalogs;
+--------------+-----------+
| Table        | Checksum  |
+--------------+-----------+
| shop.catalogs | 219379133 |
+--------------+-----------+
mysql> INSERT INTO catalogs VALUES(NULL, 'Периферия');
mysql> CHECKSUM TABLE catalogs;
+--------------+------------+
| Table        | Checksum   |
+--------------+------------+
| shop.catalogs | 1559532263 |
+--------------+------------+
mysql> UPDATE catalogs SET id_catalog = 10 WHERE id_catalog = 6;
mysql> CHECKSUM TABLE catalogs;
+--------------+------------+
| Table        | Checksum   |
+--------------+------------+
| shop.catalogs | 1559532263 |
+--------------+------------+
mysql> UPDATE catalogs SET name = '' WHERE name = 'Периферия';
mysql> CHECKSUM TABLE catalogs;
+--------------+------------+
| Table        | Checksum   |
+--------------+------------+
| shop.catalogs | 1895537962 |
+--------------+------------+
```

Изменения, проводимые над таблицей `catalogs`, такие как вставка новых записей, редактирование или удаление уже существующих, приводят к изменению контрольной суммы таблицы, поэтому, делая отметки, всегда можно контролировать, подвергалась таблица изменению или нет. Следует заметить, что изменение числового значения с 6 на 10 не приводит к изменению контрольной суммы таблицы. Оператор `CHECKSUM TABLE` может подсчитывать контрольную сумму сразу для нескольких таблиц — для этого их необходимо перечислить через запятую сразу после предложения `CHECKSUM TABLE` (листинг 31.7).

Листинг 31.7. Подсчет контрольной суммы для нескольких таблиц

```
mysql> CHECKSUM TABLE catalogs, products, users;
+--------------+------------+
| Table        | Checksum   |
+--------------+------------+
| shop.catalogs | 1895537962 |
| shop.products | 1706424754 |
| shop.users    | 1587572300 |
+--------------+------------+
```

Оператор `CHECKSUM TABLE` имеет две взаимоисключающие опции — `QUICK` и `EXTENDED`, которые располагаются сразу после списка таблиц. При создании таблицы при помощи оператора `CREATE TABLE` или в результате ее изменения при помощи `ALTER TABLE`, можно потребовать от СУБД MySQL автоматически подсчитывать контрольную сумму, выставив опцию таблицы `CHECKSUM` в 1. Опция `QUICK` оператора `CHECKSUM TABLE` (листинг 31.8) позволяет извлекать контрольную сумму из счетчика, который ведется для таблицы — если учет суммы не ведется, оператор вернет `NULL`. Опция `EXNTENDED` игнорирует счетчик контрольной суммы и требует от оператора `CHECKSUM TABLE` провести полное сканирование таблицы.

Замечание

Использование включенной опции `CHECKSUM` в таблице приводит к замедлению операций вставок, обновления и удаления записей. Кроме того, в настоящий момент автоматический счетчик контрольной суммы поддерживает только один тип таблиц — MyISAM.

Листинг 31.8. Использование опции `QUICK`

```
mysql> CHECKSUM TABLE catalogs, products, users QUICK;
+--------------+----------+
| Table        | Checksum |
+--------------+----------+
| shop.catalogs |     NULL |
```

```
| shop.products |     NULL |
| shop.users    |     NULL |
+---------------+----------+
mysql> ALTER TABLE catalogs CHECKSUM = 1;
mysql> ALTER TABLE products CHECKSUM = 1;
mysql> ALTER TABLE users CHECKSUM = 1;
mysql> CHECKSUM TABLE catalogs, products, users QUICK;
+---------------+------------+
| Table         | Checksum   |
+---------------+------------+
| shop.catalogs | 1895537962 |
| shop.products | 1706424754 |
| shop.users    | 2410629935 |
+---------------+------------+
```

Как видно из листинга 31.8, 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.

31.4. Оператор *OPTIMIZE TABLE*

Оператор `OPTIMIZE TABLE` дефрагментирует таблицу, освобождая неиспользуемое свободное пространство (листинг 31.9). Такое свободное пространство может появляться в результате выполнения операторов `DELETE`, `REPLACE` и `UPDATE`, особенно, если в таблице используются столбцы переменной длины.

> ***Замечание***
>
> Данный оператор может применяться только к таблицам типа MyISAM и BDB. На время выполнения оператора `OPTIMIZE TABLE` доступ к таблице блокируется.

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

Листинг 31.9. Использование оператора `OPTIMIZE TABLE`

```
mysql> OPTIMIZE TABLE catalogs, products, users, orders;
+---------------+----------+----------+----------+
| Table         | Op       | Msg_type | Msg_text |
```

```
+--------------+----------+----------+----------+
| shop.catalogs | optimize | status   | OK       |
| shop.products | optimize | status   | OK       |
| shop.users    | optimize | status   | OK       |
| shop.orders   | optimize | status   | OK       |
+--------------+----------+----------+----------+
```

Оператор `OPTIMIZE TABLE` выполняет следующие действия:

- дефрагментирует таблицы для уменьшения неиспользуемого пространства и уменьшения размера таблиц;
- объединяет содержимое строк переменной длины, которые стали фрагментированными (разбиты на несколько частей);
- по мере необходимости сортирует индексные страницы;
- обновляет внутреннюю табличную статистику.

В большинстве случаев вообще нет необходимости вызывать оператор `OPTIMIZE TABLE`. Даже если в таблицу постоянно вносятся изменения, а сама таблица содержит строки переменной длины, не рекомендуется проводить оптимизацию таблиц чаще, чем раз в неделю или месяц.

31.5. Оператор *REPAIR TABLE*

Оператор `REPAIR TABLE` (листинг 31.10) предназначен для восстановления поврежденных таблиц.

> ***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 31.10. Использование оператора `REPAIR TABLE`

```
mysql> REPAIR TABLE catalogs, products, users, orders;
+--------------+--------+----------+--------------------------------+
| Table        | Op     | Msg_type | Msg_text                       |
+--------------+--------+----------+--------------------------------+
| shop.catalogs | repair | note     | The storage engine for the table
```

```
                                        doesn't support repair           |
| shop.products | repair | status   | OK                                 |
| shop.users    | repair | status   | OK                                 |
| shop.orders   | repair | status   | OK                                 |
+---------------+--------+----------+------------------------------------+
```

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 опции:

- `QUICK` — быстрое восстановление, при котором восстанавливается только дерево индекса;
- `EXTENDED` — полное восстановление, в котором оператор просматривает таблицу строку за строкой;
- `USE_FRM` — используется (листинг 31.11), 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 31.11. Использование опции `USE_FRM`

```
mysql> 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. В худшем случае можно получить пустой индексный файл, и следующая операция может переписать файл данных. Это маловероятно, но все же возможно.

31.6. Оператор *BACKUP TABLE*

Оператор `BACKUP TABLE` копирует в резервный каталог минимальный набор файлов таблицы, необходимый для ее последующего восстановления (листинг 31.12). Копируются только файлы данных с расширением myd и определения структуры таблицы с расширением frm. Индексный myi-файл может быть построен на основе первых двух.

Замечание

Данный оператор может 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 31.12. Использование оператора `BACKUP TABLE`

```
mysql> 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 31.13).

Листинг 31.13. Резервный каталог `'d:\tmp'` должен существовать

```
mysql> 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 выполнением оператора `BACKUP TABLE` вызвать блокировку таблиц по чтению (листинг 31.14). 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.

Замечание

Подробнее синтаксис оператора `LOCK TABLES` рассматривается в *главе 26*.

Листинг 31.14. Предварительная блокировка таблиц

```
mysql> LOCK TABLES products READ, users READ, orders READ;
mysql> 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       |
+--------------+--------+----------+----------+
```

31.7. Оператор *RESTORE TABLE*

Оператор `RESTORE TABLE` (листинг 31.15) восстанавливает таблицу из резервной копии, созданной при помощи оператора `BACKUP TABLE`.

Замечание

Данный оператор может применяться только к таблицам типа MyISAM.

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 31.15. Использование оператора `RESTORE TABLE`

```
mysql> 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 31.16.

Листинг 31.16. Восстанавливаемых таблиц в базе данных не должно быть

```
mysql> 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 myd-файлов данных и frm-файла структуры таблицы. 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 оператора `BACKUP TABLE`, т. к. требуется воссоздание индексов. Чем больше у таблицы индексов, тем больше требуется времени на восстановление.

31.8. Резервное копирование

Помимо операторов `BACKUP TABLE` и `RESTORE TABLE` существует еще несколько альтернатив при создании резервных копий. Как уже упоминалось в *главах 3* и *4*, таблица базы данных представляется в виде нескольких файлов, поэтому резервное копирование можно осуществлять просто их копированием. Единственной сложностью тут является поддержание целостности данных, копирование файлов занимает определенный промежуток времени, за который может быть осуществлена вставка нескольких записей. Чтобы предотвратить это, можно заблокировать таблицы при помощи оператора `LOCK TABLES`, синтаксис которого подробно рассматривается в *главе 26*. Сразу после оператора `LOCK TABLES` необходимо выполнить оператор `FLUSH TABLES`, который гарантирует, что все активные страницы индексов будут записаны на диск до начала резервного копирования.

Осуществление этих процедур вручную — весьма утомительное занятие, кроме того, при ручном управлении достаточно просто совершить ошибку. Поэтому в состав дистрибутива MySQL входит сценарий на Perl — `mysqlhotcopy`, который соединяется с сервером и отсылает операторы блокировки и обновления для каждой копируемой таблицы.

> ***Замечание***
>
> Скрипт `mysqlhotcopy` работает только UNIX-подобных операционных системах и требует наличие в системе Perl и DBI-интерфейса. Кроме того, при помощи `mysqlhotcopy` можно резервировать только таблицы типа MyISAM и ARCHIVE.

Утилита `mysqlhotcopy` имеет следующий синтаксис запуска:

`mysqlhotcopy [`*`parameters`*`] `*`db_name`*` [`*`/path/to/new_directory`*`]`

Утилита может принимать параметры *`parameters`* и копировать базу данных в *`db_name`* в каталог *`/path/to/new_directory`*.

В листинге 31.17 приводится пример использования утилиты `mysqlhotcopy` — каталог учебной базы данных `shop` копируется в каталог /tmp/shop.

Листинг 31.17. Использование утилиты `mysqlhotcopy`

```
mysqlhotcopy -u root shop /tmp/copy
```

Как видно из листинга 31.17, утилита принимает параметр `-u`, через который сообщается, что пользователь действует из-под учетной записи `root`. Утилита `mysqlhotcopy` может принимать стандартные параметры, представленные в табл. 3.2, а также ряд специфических параметров, которые описываются в табл. 31.1.

Таблица 31.1. *Параметры утилиты* `mysqlhotcopy`

Параметр	Описание
`--addtodest`	При использовании этого параметра не переименовывается каталог назначения, если он уже существует — в него просто добавляются резервируемые файлы
`--allowold`	При использовании данного параметра работа функции не прерывается, если в каталоге назначения существует файл с именем резервируемого файла. Старый файл переименовывается с добавлением суффикса `_old`
`--checkpoint=`*`db_name.tbl_name`*	Вставляет контрольные точки в таблице *`tbl_name`* базы данных *`db_name`*
`--chroot=`*`path`*	Путь к `chroot`-каталогу, этот параметр необходимо использовать, если сервер `mysqld` запускается в `chroot`-окружении
`--debug`	Данный параметр включает вывод отладочной информации
`--dryrun`, `-n`	При использовании данного параметра утилита сообщает о совершаемых действиях без их реального выполнения. Этот режим может быть удобен для проверки функциональных возможностей `mysqlhotcopy`
`--flushlog`	При использовании данного параметра таблицы обновляются при помощи оператора `FLUSH TABLES` сразу после того, как они закрываются при помощи `LOCK TABLES`
`--keepold`	При использовании параметра `--allowold` существующий файл не удаляется, а переименовывается в новый файл с суффиксом `_old`. Использование параметра `--keepold` требует от утилиты `mysqlhotcopy`, чтобы переименованный файл не удалялся после работы
`--method=`*`command`*	Данный параметр может принимать два значения (`method`) — `cp` и `scp`. По умолчанию для копирования файлов используется UNIX-утилита `cp`, которая действует только локально, но можно указать и утилиту `scp`, которая копирует файлы по сети с помощью протокола SSH
`--noindices`	При использовании данного параметра индексные `myi`-файлы не резервируются. Данный параметр можно использовать для ускорения процесса резервирования. Восстановить индексы можно при помощи утилиты `myisamchk`
`--quiet`, `-q`	Отключить весь вывод в стандартный поток кроме сообщений об ошибках

Таблица 31.1 *(окончание)*

Параметр	Описание
`--regexp=`*`expr`*	Скопировать базы данных, имена которых совпадают с регулярным выражением *`expr`*
`--suffix=`*`str`*	При использовании данного параметра каждому файлу будет добавлен суффикс *`str`*. Данный параметр используется при копировании базы данных непосредственно в каталог баз данных
`--tmpdir=`*`path`*	Путь к каталогу, в котором создаются временные файлы. Если параметр не указывается, используется каталог, заданный в переменной окружения `TMPDIR`

При использовании параметра `--checkpoint=`*`db_name`*`.`*`tbl_name`*, для создания контрольной точки необходимо в базе данных *`db_name`* создать таблицу *`tbl_name`*, структура которой представлена в листинге 31.18.

Листинг 31.18. Таблица для создания контрольных точек

```
CREATE TABLE tbl_name
(
  time_stamp TIMESTAMP NOT NULL,
  src VARCHAR(32),
  dest VARCHAR(60),
  msg VARCHAR(255)
);
```

Здесь `src` и `dest` являются именами исходной и полученной баз данных, а в поле `msg` записывается сообщение со статусом операции копирования (успешно или неуспешно).

31.9. Утилита *mysqlcheck*

Утилита `mysqlcheck` проверяет и восстанавливает таблицы типа MyISAM. Для утилиты `mysqlcheck` имеется альтернативная утилита `myisamchk`. Основное отличие этих двух функций заключается в том, что утилита `mysqlcheck` должна использоваться при запущенном сервере MySQL, в то время как `myisamchk` — нет.

Утилита `mysqlcheck` является интерфейсом для операторов `CHECK TABLE`, `REPAIR TABLE`, `ANALYZE TABLE` и `OPTIMIZE TABLE`.

Существуют три основных способа запуска утилиты `mysqlcheck`:

```
shell> mysqlcheck [parameter] db_name [tables]
shell> mysqlcheck [parameter] --databases db_name1 [db_name2 db_name3...]
shell> mysqlcheck [parameter] --all-databases
```

Если не указываются никакие таблицы *tables* или используется параметр --databases или --all-databases, будут проверены все базы данных. Порядок вызова утилиты mysqlcheck несколько отличается от других. Поведение утилиты mysqlcheck по умолчанию (проверка таблицы) может быть изменено путем переименования программы. Так, для восстановления таблицы утилиту можно просто переименовать в mysqlrepair (можно также создать символическую ссылку). После этого запуск утилиты будет приводить не к проверке таблиц, а к их восстановлению. В табл. 31.2 приводятся имена утилиты и соответствующее им поведение по умолчанию.

Таблица 31.2. *Имена утилиты* mysqlcheck

Имя	Поведение по умолчанию
mysqlcheck	По умолчанию используется параметр --check
mysqlrepair	По умолчанию используется параметр --repair
mysqlanalyze	По умолчанию используется параметр --analyze
mysqloptimize	По умолчанию используется параметр --optimize

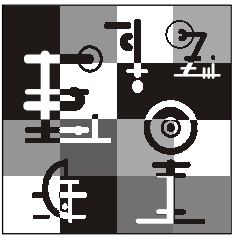
Утилита mysqlcheck может принимать стандартные параметры, представленные в табл. 3.2, а также ряд специфических параметров, которые описываются в табл. 31.3.

Таблица 31.3. *Параметры утилиты* mysqlhotcopy

Параметр	Описание
--all-databases, -A	При использовании этого параметра проверяются все таблицы во всех базах данных
--all-in-1, -1	При использовании данного параметра вместо того, чтобы выполнять оператор для каждой таблицы базы данных, выполняется один оператор для всей базы данных
--analyze, -a	При использовании данного оператора осуществляется анализ таблиц
--auto-repair	Если проверяемая таблица повреждена, автоматически осуществляется попытка ее восстановления. Все операции по восстановлению выполняются после того, как все таблицы будут проверены
--check, -c	При использовании данного оператора осуществляется проверка таблиц
--check-only-changed, -C	При использовании данного параметра проверяются только те таблицы, которые изменились с момента последней проверки или не были корректно закрыты
--check-upgrade, -g	При использовании данного параметра проверка при помощи оператора CHECK TABLE запускается с ключевым словом FOR UPGRADE. Параметр --check-upgrade добавлен, начиная с версии MySQL 5.1.7

Таблица 31.3 *(окончание)*

Параметр	Описание
`--databases`, `-B`	Данный параметр позволяет указать базы данных, таблицы в которых необходимо проверить
`--debug`	Данный параметр включает вывод отладочной информации
`--extended`, `-e`	Если при проверке таблиц используется параметр `--extended`, это гарантирует, что они будут на 100% исправны и непротиворечивы, но проверка потребует длительного времени Если параметр используется при восстановлении таблиц, будет выполняться расширенное восстановление, которое не только потребует длительного времени, но и может сгенерировать большое количество "мусорных" строк
`--fast`, `-F`	При использовании данного параметра проверяются только те таблицы, которые не были корректно закрыты
`--fix-db-names`	При использовании данного параметра имена баз данных конвертируются в формат версии MySQL 5.1. Параметр добавлен, начиная с версии MySQL 5.1.7
`--fix-table-names`	При использовании данного параметра имена таблиц конвертируются в формат версии MySQL 5.1. Параметр добавлен, начиная с версии MySQL 5.1.7
`--force`, `-f`	При использовании данного параметра утилита продолжает работу, даже если происходит ошибка
`--medium-check`, `-m`	При использовании данного параметра осуществляется проверка, которая быстрее, чем при операции `--extended`. При этом находятся 99,99% всех ошибок
`--optimize`, `-o`	При использовании данного оператора осуществляется оптимизация таблиц
`--quick`, `-q`	При проверке таблиц использование данного параметра предотвращает сканирование строк для поиска неправильных ссылок, что ускоряет работу утилиты
`--repair`, `-r`	При использовании данного оператора осуществляется восстановление таблиц
`--silent`, `-s`	Режим минимального количества сообщений. В выходной поток выдаются только сообщения об ошибках
`--tables`	Параметр используется совместно с параметром `--databases` или `-B`. Все аргументы после параметра `--tables` трактуются как имена таблиц
`--verbose`, `-v`	Режим расширенных сообщений. При использовании данного параметра выводится дополнительная информация



Глава 32

Репликация в MySQL

Репликация позволяет дублировать данные основного сервера на одном или более подчиненных серверов. Репликация может осуществляться как в режиме online, так и время от времени — подчиненный сервер может загружаться только для того, чтобы загрузить обновления. В любом случае резервное копирование осуществляется автоматически без вмешательства администратора. Однако если интервал обновления мал (по умолчанию 60 секунд), то все изменения (в том числе и случайные удаления баз данных) будут тут же отражаться на подчиненном сервере. Главное назначение серверов репликации — поддержка точной копии, чтобы предотвратить повреждение баз данных в результате ошибки или после выхода из строя сервера по аппаратной причине временно заменить его. Проводя аналогию с жесткими дисками, можно сказать, что репликационный сервер — это своеобразный RAID 1 для базы данных, т. е. зеркальная копия данных.

Замечание

Репликации могут подвергаться как все базы данных сервера, так и их часть.

32.1. Введение в репликацию

В схеме репликации один север выступает в качестве главного (master) сервера, в то время как один или более серверов являются подчиненными (slave). Главный сервер регистрирует все изменения в бинарном журнале *(см. разд. 28.4.4)*, который затем рассылается подчиненным серверам. Когда подчиненный сервер подключается к главному серверу, он извлекает все изменения в бинарном журнале, которые накопились с момента последнего обращения. Репликация — это, в отличие от кластерных технологий, всегда односторонний процесс, главный сервер не получает бинарные журналы подчиненных серверов. Даже если происходит сбой, то восстановление с использованием информации с подчиненного сервера выполняется вручную. Поэтому все изменения базы данных производятся, как правило, только на главном сервере, на подчиненном сервере SQL-запросы не выполняются для того, чтобы избежать возможных конфликтов при обновлении. Например, если на подчиненном

сервере удаляется таблица, а на главном сервере она остается и фигурирует в бинарном журнале — возникнет конфликт.

> ***Замечание***
>
> Для реализации репликации на главном сервере необходимо активировать бинарный журнал при помощи параметра `--log-bin` *(см. разд. 28.1)* 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 изменится и имя бинарного журнала, что в свою очередь может привести к сбою процесса репликации.

Подчиненный сервер может выступать также и в качестве главного сервера для других подчиненных серверов. Таким образом, могут выстраиваться длинные цепочки серверов, которые последовательно получают обновления друг от друга.

Получая обновления, подчиненный сервер осуществляет операции для изменения базы данных и сам, в свою очередь, отмечает изменения в своем бинарном журнале. Записи в бинарном журнале начинаются со строго определенной даты, когда был активирован бинарный журнал, или с момента последнего удаления старых журнальных файлов. Поэтому на подчиненных серверах перед запуском репликации должна быть копия базы данных с главного сервера, в противном случае не исключена возможность аварийного завершения работы подчиненного сервера.

Один из способов скопировать данные главного сервера на подчиненный предполагает использование оператора `LOAD DATA FROM MASTER` *(см. разд. 32.7.2)*. Однако данный оператор обладает рядом ограничений, в частности, он поддерживает только таблицы типа MyISAM, к тому же он осуществляет глобальную блокировку по чтению, поэтому пока он не завершит работу, никакие операции на главном сервере не возможны. Как правило, оператор `LOAD DATA FROM MASTER` применяется только для небольших таблиц.

После того как подчиненный сервер получит обновления с главного сервера, он не отключается от него и ждет обновлений, которые необходимо обработать. Если главный сервер отключается или прерывается связь, подчиненный сервер периодически пытается подключиться к главному серверу. Интервал между попытками определяется параметром `--master-connect-retry` *(см. разд. 32.5)*, который по умолчанию принимает значение 60 секунд.

Каждый подчиненный сервер запоминает, на каком этапе он утратил связь. Главный сервер не имеет понятия о том, сколько подчиненных серверов к нему подключено и какие из них содержат актуальную информацию.

32.2. Детали реализации процесса репликации

Для поддержки репликации на главном сервере создается один поток, а на подчиненном — два. При запуске подчиненного сервера, создается поток ввода/вывода, который осуществляет соединение с главным сервером и запрашивает обновления

из бинарного журнала с момента последнего соединения. Главный сервер, в свою очередь, создает поток для отправки содержимого бинарного журнала подчиненному серверу. Если на главном сервере выполнить оператор `SHOW PROCESSLIST` *(см. разд. 30.14)*, то этот процесс будет помечен как `Binlog Dupm`. Процесс ввода/вывода подчиненного сервера получает информацию, отсылаемую процессом `Binlog Dupm`, и копирует ее в локальные файлы каталога данных. Такие файлы называются *ретрансляционными журналами* (relay logs). После этого подчиненный сервер создает второй поток для чтения ретрансляционных журналов и выполнения хранящихся в них обновлений.

> ***Замечание***
>
> Главный сервер создает отдельный поток для каждого из подчиненных серверов.

Использование двух потоков на подчиненном сервере требуется для увеличения скорости загрузки обновлений. Например, если подчиненный сервер долгое время не запускался, при старте его потока ввода/вывода очень быстро получает обновления бинарного журнала от главного сервера, даже если поток обработки будет отставать, и ему потребуются многие часы для того, чтобы обработать все обновления. Это позволит главному серверу не дожидаться конца обновления и быстро завершить соединение.

Оператор `SHOW PROCESSLIST` позволяет контролировать процессы как на главном сервере, так и на подчиненном. На главном сервере результирующая таблица оператора `SHOW PROCESSLIST` может выглядеть так, как это представлено в листинге 32.1.

> ***Замечание***
>
> Для вывода результирующей таблицы в вертикальном формате часто применяют модификатор `\G` *(см. разд. 3.1)*.

Листинг 32.1. Процесс главного сервера

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 2
   User: root
   Host: localhost:32931
     db: NULL
Command: Binlog Dump
   Time: 94
  State: Has sent all binlog to slave; waiting for binlog to
         be updated
   Info: NULL
```

Замечание

Если в отчете оператора `SHOW PROCESSLIST` не наблюдаются какие-либо потоки `Binlog Dump`, это означает, что к главному серверу не подключен ни один подчиненный сервер.

Здесь поток с идентификатором 2 — это поток репликации, созданный для подключения подчиненного сервера. В поле `State` сообщается текущий статус, из которого следует, что все необработанные изменения данных отправлены подчиненному серверу, и главный сервер ожидает поступления новых изменений.

На подчиненном сервере результат применения оператора `SHOW PROCESSLIST` может выглядеть так, как это представлено в листинге 32.2.

Листинг 32.2. Процессы подчиненного сервера

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 10
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Waiting for master to send event
   Info: NULL
*************************** 2. row ***************************
     Id: 11
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Has read all relay log; waiting for the slave I/O
         thread to update it
   Info: NULL
```

Здесь поток с идентификатором 10 — это поток ввода/вывода, который загружает обновления с главного сервера, а поток с идентификатором 11 — поток, обрабатывающий ретрансляционные журналы. Как следует из комментария в поле `State`, оба потока простаивают, ожидая будущих изменений.

Столбец `Time` показывает, как давно подчиненный сервер не связывался с главным.

32.2.1. Состояние потока репликации главного сервера

В списке, представленном далее, приводятся состояния потока `Binlog Dump` главного сервера, которые могут фигурировать в столбце `State` результирующей таблицы оператора `SHOW PROCESSLIST`.

- `Sending binlog event to slave` (Событие отправки бинарного журнала подчинено серверу). Это состояние возникает, если бинарный журнал содержит события, которые представляют собой обычный оператор обновления, плюс некоторую добавочную информацию. Поток считывает события из бинарного журнала и отправляет их подчиненному серверу.
- `Finished reading one binlog; switching to next binlog` (Завершение чтения очередного бинарного журнала; переключение на следующий бинарный журнал). Состояние возникает, если поток завершил чтение файла бинарного журнала и открывает следующий такой файл, чтобы отправить его подчиненному серверу.
- `Has sent all binlog to slave; waiting for binlog to be updated` (Все бинарные журналы отправлены подчиненному серверу; ожидание бинарного журнала для обновления). Состояние возникает, если поток прочитал все отложенные изменения из бинарных журналов и отправил их подчиненному серверу. Это состояние простоя, с ожиданием появления новых событий в бинарном журнале, появляющихся в результате выполнения операторов изменения данных.
- `Waiting to finalize termination` (Ожидание завершения). Данное кратковременное состояние возникает при остановке потока.

32.2.2. Состояние потока ввода/вывода подчиненного сервера

В списке, приведенном далее, перечислены состояния потока ввода/вывода подчиненного сервера. Данное состояние можно контролировать в столбце `State` результирующей таблицы оператора `SHOW PROCESSLIST`. Начиная с версии 4.1.1, это состояние может присутствовать и в столбце `Slave_IO_State` результирующей таблицы оператора `SHOW SLAVE STATUS` *(см. разд. 32.7.7)*.

- `Connecting to master` (Соединение с главным сервером). Состояние возникает, если поток пытается подключиться к главному серверу.
- `Checking master version` (Проверка версии главного сервера). Данное кратковременное состояние возникает, когда поток пребывает в состоянии непосредственно после того, как подключение установлено.
- `Registering slave on master` (Регистрация подчиненного сервера на главном сервере). Также очень кратковременное состояние после установления соединения.
- `Requesting binlog dump` (Запрос дампа бинарного журнала). Это кратковременное состояние возникает после установки соединения с главным сервером.

Поток посылает главному серверу запрос на извлечение содержимого бинарного журнала, начиная с последней строки обновления.

- `Waiting to reconnect after a failed binlog dump request` (Ожидание повторного соединения после сбоя запроса дампа бинарного журнала). Если запрос выдал сбой (например, из-за разрыва соединения), поток переходит в это состояние, при котором находится в спящем режиме, периодически пытаясь восстановить соединение. Интервал между попытками определяется при помощи параметра `--master-connect-retry` *(см. разд. 32.5)*, который по умолчанию принимает значение 60 секунд.
- `Reconnecting after a failed binlog dump request` (Повторное соединение после сбоя запроса дампа бинарного журнала). Данное состояние возникает, если поток пытается восстановить соединение с главным сервером.
- `Waiting for master to send event` (Ожидание событий от главного сервера). Состояние возникает, если поток подключается к главному серверу и ожидает появления событий из бинарного журнала. В этом состоянии он может находиться длительное время, если главный сервер простаивает. Если ожидание длится более `slave_read_timeout` секунд, подчиненный сервер считает, что изменений нет, и разрывает соединение на `master-connect-retry` секунд (по умолчанию 60).
- `Queueing master event to the relay log` (Запрос события главного сервера для журнала ретрансляции). Данное состояние возникает, если поток прочитал событие и копирует его в журнал ретрансляции, чтобы поток обработки ретрансляционных журналов мог его обработать.
- `Waiting to reconnect after a failed master event read` (Ожидание повторного соединения после сбоя в чтении события главного сервера). Состояние возникает, если в результате разрыва соединения произошла ошибка чтения информации с главного сервера. Поток переходит в режим ожидания на `master-connect-retry` секунд и затем повторяет попытку подключения.
- `Reconnecting after a failed master event read` (Повторное соединение после разрыва соединения, произошедшего во время чтения информации с главного сервера). Состояние возникает, если поток пытается восстановить соединение с главным сервером. Когда соединение устанавливается, поток переходит в состояние `Waiting for master to send event`.
- `Waiting for the slave SQL thread to free enough relay log space` (Ожидание, пока поток подчиненного сервера освободит достаточно места в журнале ретрансляции). Возникновение данного состояния означает, что используется ненулевое значение переменной `relay_log_space_limit` и журналы ретрансляции выросли настолько, что их общий размер превысил это значение. Поток ввода/вывода ожидает, пока поток обработки ретрансляционных файлов не освободит достаточно места, обработав накопившиеся в журналах ретрансляции события, чтобы иметь возможность удалить часть из них.
- `Waiting for slave mutex on exit` (Ожидание семафора завершения от подчиненного сервера). Данное кратковременное состояние возникает при остановке потока.

32.2.3. Состояние потока обработки ретрансляционных журналов подчиненного сервера

Далее приводятся состояния потока обработки ретрансляционных журналов подчиненного сервера. Данное состояние можно контролировать в столбце `State` результирующей таблицы оператора `SHOW PROCESSLIST`.

- `Reading event from the relay log` (Чтение события из журнала ретрансляции). Событие возникает, если поток читает событие из журнала ретрансляции перед началом его обработки.
- `Has read all relay log; waiting for the slave I/O thread to update it` (Прочитан весь журнал ретрансляции; ожидание, когда поток ввода/вывода не обновит журнал). Состояние возникает, если поток обработал все события из файлов журнала ретрансляции и ждет, когда поток ввода/вывода запишет туда новые события.
- `Waiting for slave mutex on exit` (Ожидание семафора завершения от подчиненного сервера). Данное кратковременное состояние возникает при остановке потока.

Столбец `State` результирующей таблицы оператора `SHOW PROCESSLIST` может также содержать текст оператора, если из журнала ретрансляции извлечен оператор и запущен на выполнение.

32.2.4. Журнал ретрансляции и файлы состояния

По умолчанию журналы ретрансляции используют имена *host*-relay-bin.*nnnnnn*, где *host* — имя подчиненного сервера, а *nnnnnn* — последовательный номер, который начинается с 000001. Подчиненный сервер учитывает текущие используемые журналы в файле индекса. Имя файла индекса по умолчанию формируется как *host*-relay-bin.index, где *host* — имя подчиненно сервера. По умолчанию все эти файлы создаются в каталоге данных подчиненного сервера. Имена файлов по умолчанию могут быть изменены с помощью параметров сервера `--relay-log` и `--relay-log-index` *(см. разд. 32.5)*.

> ***Замечание***
>
> Файлы ретрансляционных журналов имеют тот же формат, что и бинарные журналы, поэтому для их чтения можно использовать утилиту `mysqlbinlog` *(см. разд. 28.4.5)*.

Журналы ретрансляции удаляются сразу после того, как поток обработки ретрансляционных файлов выполнит все записи из журнала. Не существует никакого явного механизма удаления журналов ретрансляции, т. к. поток обработки журнала сам заботится об этом.

Подчиненный сервер репликации помимо журналов ретрансляции создает два файла состояния в каталоге данных: master.info и relay-log.info. Эти файлы содержат информацию, подобную той, что выводит оператор `SHOW SLAVE STATUS` *(см. разд. 32.7.7)*. В случае остановки сервера в этих файлах сохраняется информация о том, на каком месте остановилось чтение бинарных журналов главного сервера (master.info) и на каком журнале ретрансляции (relay-log.info).

Файл master.info обновляется потоком ввода/вывода. До MySQL 4.1 соответствие между строками в файле и столбцами результирующей таблицы оператора SHOW SLAVE STATUS определялось табл. 32.1.

Таблица 32.1. *Соответствие между строками файла master.info и результирующей таблицей оператора* `SHOW SLAVE STATUS` *до MySQL 4.1*

Строка файла	Описание
1	Master_Log_File
2	Read_Master_Log_Pos
3	Master_Host
4	Master_User
5	Пароль (не выводится оператором SHOW SLAVE STATUS)
6	Master_Port
7	Connect_Retry

Начиная с версии MySQL 4.1, файл включает количество строк и информацию о параметрах SSL (табл. 32.2).

Таблица 32.2. *Соответствие между строками файла master.info и результирующей таблицей оператора* `SHOW SLAVE STATUS` *после MySQL 4.1*

Строка файла	Описание
1	Количество строк в файле
2	Master_Log_File
3	Read_Master_Log_Pos
4	Master_Host
5	Master_User
6	Пароль (не выводится оператором SHOW SLAVE STATUS)
7	Master_Port
8	Connect_Retry
9	Master_SSL_Allowed
10	Master_SSL_CA_File
11	Master_SSL_CA_Path
12	Master_SSL_Cert
13	Master_SSL_Cipher
14	Master_SSL_Key

Файл relay-log.info обновляется потоком обработки ретрансляционных журналов. Соответствие между строками файла и столбцами результирующей таблицы оператора SHOW SLAVE STATUS выглядит так, как это представлено в табл. 32.3.

***Таблица 32.3.** Соответствие между строками файла relay-log.info и результирующей таблицей оператора SHOW SLAVE STATUS*

Строка файла	Описание
1	Relay_Log_File
2	Relay_Log_Pos
3	Relay_Master_Log_File
4	Exec_Master_Log_Pos

32.3. Настройка репликации

Все процедуры по настройке репликации выполняются от имени пользователя, который имеет привилегию SUPER.

Для настройки репликации обычно создают отдельную учетную запись на главном сервере (с привилегией REPLICATION SLAVE, однако не лишним будет снабдить ее привилегиями SELECT, RELOAD, SUPER), которую сможет использовать подчиненный сервер для подключения. Предположим, что и главный, и подчиненные серверы находятся в домене domain.ru. Тогда для репликационных серверов можно создать учетную запись с именем repl и паролем slavepass так, как это представлено в листинге 32.3. Данная учетная запись может использоваться всеми серверами в доменной зоне domain.ru.

Листинг 32.3. Создание учетной записи для репликации

```
mysql> GRANT REPLICATION SLAVE ON *.*
    -> TO 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass';
```

Если планируется использовать операторы LOAD TABLE FROM MASTER и LOAD DATA FROM MASTER с хоста подчиненного сервера, то учетную запись необходимо снабдить дополнительными привилегиями:

- глобальными привилегиями SUPER и RELOAD;
- привилегией SELECT на все таблицы, которые необходимо загружать. Любые таблицы главного сервера, для которых учетная запись не может выполнить оператор SELECT, будут проигнорированы оператором LOAD DATA FROM MASTER.

Перед запуском подчиненного сервера, на нем необходимо расположить точную копию данных главного сервера. Если используются только таблицы типа MyISAM, необходимо сбросить все таблицы и установить блокировку на запись, выполнив оператор FLUSH TABLES WITH READ LOCK (листинг 32.4).

Листинг 32.4. Использование оператора FLUSH TABLES WITH READ LOCK

```
mysql> 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 31)*.

> ***Замечание***
>
> Если нет необходимости реплицировать базу данных mysql, ее можно не включать в SQL-дамп, который разворачивается на подчиненном сервере.

Пока действует блокировка по чтению, установленная командой FLUSH TABLES WITH READ LOCK, необходимо прочитать значение имени текущего бинарного журнала и смещения на главном сервере (листинг 32.5).

Листинг 32.5. Использование оператора SHOW MASTER STATUS

```
mysql > 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 зафиксировано имя журнала и смещение, можно снять блокировку и разрешить проведение изменений при помощи оператора UNLOCK TABLES (листинг 32.6).

Листинг 32.6. Разблокировка таблиц

```
mysql> UNLOCK TALBES;
```

Следует убедиться, что секция [mysqld] в файле my.cnf на хосте главного сервера включает параметр log-bin, который активирует бинарный журнал *(см. разд. 28.4.4)*. Помимо этого необходимо указать идентификатор главного сервера при помощи параметра server-id, который может принимать положительное значение в интервале

от 1 до $2^{32} - 1$. Если не указать параметр `server-id` на одном из серверов, процесс репликации работать не будет. Каждый сервер в схеме репликации должен иметь уникальный номер, по которому серверы отличаются друг от друга. Секция `[mysqld]` в конфигурационном файле `my.cnf` может выглядеть так, как это представлено в листинге 32.7.

> ***Замечание***
>
> После редактирование конфигурационного файла сервер необходимо перезапустить.

Листинг 32.7. Настройка главного сервера

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

Затем можно переходить к настройке подчиненного репликационного сервера. Для этого его необходимо остановить и добавить в секцию `[mysqld]` конфигурационного файла my.cnf параметр `server-id` со значением, отличным от значения этого параметра на главном сервере и других подчиненных серверах (если они имеются). Секция `[mysqld]` в конфигурационном файле my.cnf на подчиненном сервере может выглядеть так, как это представлено в листинге 32.8.

Листинг 32.8. 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.

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 и запустить сервер. 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 32.9).

Листинг 32.9. Подготовка подчиненного сервера к репликации

```
mysql> 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, ранее мы назвали такого пользователя `repl` (см. листинг 32.3), `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.

> ***Замечание***
>
> Параметр `MASTER_HOST` принимает максимальное значение 60 символов, `MASTER_USER` — 16, `MASTER_PASSWORD` — 32, а `MASTER_LOG_FILE` — 255.

После того как все приготовления выполнены, необходимо запустить репликацию на подчиненном сервере при помощи оператора `START SLAVE`.

32.4. Совместимость репликации между версиями MySQL

Бинарные журналы и репликация впервые появились в версии MySQL 3.23. Затем формат бинарного журнала изменился в версии MySQL 4.0 и версии MySQL 5.0. В связи с этим, на версии главного и подчиненных серверов накладываются определенные ограничения, которые описываются в табл. 32.4.

Таблица 32.4. *Совместимость репликации между серверами различных версий*

Подчиненный сервер	Главный сервер		
	версии 3.23.33 и выше	версии 4.0.3 и выше	версии 5.0.0 и выше
Версии 3.23.33 и выше	Да	Нет	Нет
Версии 4.0.3 и выше	Да	Да	Нет
Версии 5.0.0 и выше	Да	Да	Да

32.5. Параметры запуска репликации

В данном разделе описываются параметры сервера `mysqld`, имеющие отношение к репликации. Некоторые параметры репликации подчиненного сервера обрабатываются специальным образом: они игнорируются, если на момент старта сервера уже существует файл master.info. К таким параметрам относятся:

- `--master-host`;
- `--master-user`;

- `--master-password`;
- `--master-port`;
- `--master-connect-retry`.

Начиная с версии MySQL 4.1.1, к приведенному выше списку добавились параметры:

- `--master-ssl`;
- `--master-ssl-ca`;
- `--master-ssl-capath`;
- `--master-ssl-cert`;
- `--master-ssl-cipher`;
- `--master-ssl-key`.

Если файл master.info не существует на момент запуска подчиненного сервера, для этих параметров используются те значения, которые указываются в командной строке или конфигурационном файле. Такая ситуация возникает, когда сервер запускается первый раз в качестве подчиненного репликационного сервера или после применения оператора `RESET SLAVE` *(см. разд. 32.7.5)*. Если файл master.info существует на момент запуска MySQL-сервера, приведенные выше параметры игнорируются, а вместо них используются значения из master.info.

Далее описываются параметры MySQL-сервера, используемые для настройки репликации.

- `--log-slave-updates`. Обычно изменения, полученные подчиненным сервером от главного, не записываются в его бинарный журнал. Параметр `--log-slave-updates` требует от подчиненного сервера регистрировать изменения в бинарном журнале. Для того чтобы MySQL-сервер не проигнорировал этот параметр, необходимо, чтобы сервер был запущен также с параметром `--bin-log`, который активирует бинарную активацию. Параметр `--log-slave-updates` применяется в тех случаях, когда необходимо выстроить цепочку репликационных серверов `A -> B -> C`. Для того чтобы репликационный сервер `C` получал обновления, на сервере `B` должен поддерживаться актуальный бинарный журнал.
- `--log-warnings[=`*`level`*`]`. Использование данного параметра заставляет подчиненный сервер печатать больше сообщений о том, что он делает. Например, сервер сообщит о разрыве соединения и его успешном восстановлении, как стартовал каждый из потоков, обслуживающих репликацию. По умолчанию параметр включен, но может быть отключен при помощи параметра `--skip-log-warnings`. О разрыве соединения сообщается лишь в том случае, если значение параметра *`level`* принимает значение больше единицы.
- `--master-connect-retry=`*`seconds`*. Параметр определяет интервал (в секундах), в течение которого сервер ожидает перед повторной попыткой восстановить соединение с главным сервером. Значение в файле master.info имеет приоритет перед параметром `--master-connect-retry`, если, конечно, файл может быть

прочитан. Если параметр не установлен, то по умолчанию принимается значение, равное 60 секундам.

- `--master-host=`*`host_name`*. Параметр определяет имя хоста или IP-адрес главного сервера репликации. Если параметр не указан, репликация на подчиненном сервере не запускается. Значение в файле master.info имеет приоритет перед параметром `--master-host`, если, конечно, файл может быть прочитан.
- `--master-info-file=`*`file_name`*. Параметр определяет имя файла, в который подчиненный сервер записывает информацию о главном сервере. Значение по умолчанию — mysql.info; это файл, размещенный в каталоге данных.
- `--master-password=`*`password`*. Параметр определяет пароль учетной записи, которую поток подчиненного сервера использует при подключении к главному серверу. Значение в файле master.info имеет приоритет перед параметром `--master-password`, если, конечно, файл может быть прочитан.
- `--master-port=`*`port_number`*. Параметр определяет номер порта TCP/IP-соединения, который используется главным сервером для прослушивания соединений. Значение, указанное в master.info, имеет приоритет, если может быть прочитано. Если номер не задается при помощи параметра `--master-port`, принимается значение по умолчанию, равное 3306.
- `--master-retry-count=`*`count`*. Данный параметр задает количество попыток, которые осуществляет подчиненный сервер для соединения с главным сервером перед тем, как перейти в спящий режим.
- `--master-`*`ssl`*, `--master-ssl-ca=`*`file_name`*, `--master-ssl-capath=`*`directory_name`*, `--master-ssl-cert=`*`file_name`*, `--master-ssl-cipher=`*`cipher_list`*, `--master-ssl-key=`*`file_name`*. Данные параметры используются для настройки защищенного канала между подчиненным и главным сервером. Аналогичные значения в файле master.info имеют приоритет перед параметрами, если, конечно, файл может быть прочитан. Параметры добавлены в версии MySQL 4.1.1.
- `--master-user=`*`user_name`*. Параметр определяет имя учетной записи, которую подчиненный сервер использует для подключения к главному серверу. Данная учетная запись должна иметь привилегию `REPLICATION SLAVE`. Значение в файле master.info имеет приоритет перед параметром `--master-user`, если, конечно, файл может быть прочитан. Если имя пользователя не указано, по умолчанию используется `test`.
- `--max-relay-log-size=`*`size`*. Данный параметр предназначен для автоматической ротации журналов ретрансляции.
- `--read-only`. Параметр запрещает какие-либо изменения данных кроме тех, что выполняет поток репликации или пользователь с привилегией `SUPER`. Обычно применяется, если необходимо гарантировать, чтобы подчиненный сервер не допускал никаких обновлений, которые могли бы конфликтовать с обновлениями, поступающими с главного сервера.

- `--relay-log=`*`file_name`*. Параметр задает имя журнала ретрансляции *`file_name`*. Если параметр не задан, имя файла журнала формируется по схеме *host*-relay-bin.*nnn*, где *host* — имя хоста подчиненного сервера, а *nnn* — последовательный номер журнала. Используя параметр `--relay-log`, можно задать имя журнала, не зависящее от имени хоста, или переместить его из каталога данных в другой каталог.
- `--relay-log-index=`*`file_name`*. Параметр задает местоположение и имя файла индекса для журнала ретрансляции. Если параметр не задан, имя индекса журнала формируется по схеме *host*-relay-bin.index, где *host* — имя хоста подчиненного сервера.
- `--relay-log-info-file=`*`file_name`*. Параметр определяет имя файла, в котором подчиненный сервер записывает информацию о журналах ретрансляции. Если параметр не задан, в качестве такого имени используется relay-log.info.
- `--relay-log-purge={0|1}`. Параметр включает (1) или отключает (0) автоматическую очистку журналов ретрансляции, как только они становятся не нужны. Если параметр не указан, считается, что автоматическая очистка журналов включена (1). Параметр влияет на глобальную переменную `relay_log_purge`, изменить значение которой можно при помощи запроса `SET relay_log_purge` *(см. главу 29)*. Параметр введен в MySQL, начиная с версии 4.1.1.
- `--relay-log-space-limit=`*`size`*. Параметр устанавливает верхний предел общего размера всех журналов ретрансляции на подчиненном сервере в байтах (0 означает отсутствие каких-либо ограничений). Параметр применяется на хостах с ограниченным дисковым пространством. Когда предел достигнут, поток ввода/вывода приостанавливает чтение событий из бинарного журнала главного сервера до тех пор, пока обработчик ретрансляционных журналов не обработает и не удалит часть ретрансляционных журналов.
- `--replicate-do-db=`*`db_name`*. Параметр позволяет указать имя базы данных *`db_name`*, по отношению к которой следует проводить репликацию. Не указанные в параметре базы данных репликации подвергаться не будут. Для указания более чем одной базы данных параметр необходимо записать несколько раз, т. к. он может принимать в качестве значения имя только одной базы данных. Реплицироваться будут только те операторы, которые имеют отношение к базе данных *`db_name`*, которые выполняются после оператора `USE` *`db_name`*. Так если база данных не выбрана при помощи `USE`, оператор `UPDATE` *`db_name`*`.`*`tbl`* `SET` *`field`*`='`*`value`*`'` не будет подвергаться репликации. Если необходимо, чтобы реплицировались межбазовые запросы, лучше воспользоваться параметром `--replicate-wild-do-table=`*`db_name`*`.%` вместо параметра `--replicate-do-db=`*`db_name`*.
- `--replicate-do-table=`*`db_name`*`.`*`tbl_name`*. Данный параметр указывает подчиненному серверу реплицировать только таблицу *`tbl_name`* базы данных *`db_name`*. Для указания более одной таблицы необходимо применить этот параметр несколько раз. В отличие от параметра `--replicate-do-db`, параметр `--replicate-do-table` будет работать и для операторов межбазового обновления.

- `--replicate-ignore-db=`*db_name*. Параметр позволяет указать имя базы данных *db_name*, по отношению к которой не следует проводить репликацию. Для указания более чем одной базы данных, параметр необходимо привести несколько раз, т. к. он может принимать в качестве значения имя только одной базы данных. Репликации не будут подвергаться операторы, изменяющие базу данных *db_name*, которые выполняются после того, как выбрана база данных *db_name*. Так, если база данных не выбрана при помощи `USE`, оператор `UPDATE` *db_name*`.`*tbl* `SET` *field*`='`*value*`'`, будет подвергаться репликации.
- `--replicate-ignore-table=`*db_name*`.`*tbl_name*. Параметр указывает подчиненному серверу не реплицировать операторы, которые изменяют таблицу *tbl_name* базы данных *db_name*. Для указания более одной таблицы, необходимо применить этот параметр несколько раз. В отличие от параметра `--replicate-ignore-db`, параметр `--replicate-ignore-table` будет работать и для операторов межбазового обновления.
- `--replicate-rewrite-db=`*from_name*`->`*to_name*. Параметр позволяет использовать на подчиненном сервере базу данных *to_name* для репликации *from_name* главного сервера. Действие параметра не распространяется на межбазовые операторы и операторы, которые выполняются без предварительного вызова оператора `USE` *from_name*. Если параметр вызывается в командной строке, а не используется в конфигурационном файле, следует применять кавычки: `mysqld --replicate-rewrite-db="`*olddb*`->`*newdb*`"`.
- `--replicate-wild-do-table=`*db_name*`.`*tbl_name*. Параметр указывает подчиненному серверу реплицировать только те операторы, в которых используются таблицы *tbl_name* базы данных *db_name*. В отличие от предыдущих операторов, здесь выражение *db_name*`.`*tbl_name* является шаблоном, в котором можно применять символы `%` и `_`, соответствующие любому числу символов и любому одному символу. Чтобы задействовать более одного шаблона, необходимо указать параметр несколько раз, по одному разу на каждый шаблон. Параметр будет работать и для операторов межбазового обновления. Если в имени базы данных необходимо использовать символы `%` и `_`, их следует экранировать двойным обратным слешем `\\`.
- `--replicate-wild-ignore-table=`*db_name*`.`*tbl_name*. Параметр указывает подчиненному серверу не реплицировать операторы, изменяющие таблицу *tbl_name* базы данных *db_name*. В отличие от предыдущих операторов, здесь выражение *db_name*`.`*tbl_name* является шаблоном, в котором можно применять символы `%` и `_`, соответствующие любому числу символов и любому одному символу. Чтобы задействовать более одного шаблона, необходимо указать параметр несколько раз, по одному разу на каждый шаблон. Параметр будет работать и для операторов межбазового обновления. Если в имени базы данных необходимо использовать символы `%` и `_`, их следует экранировать двойным обратным слешем `\\`.
- `--report-host=`*slave_name*. Параметр определяет имя хоста или IP-адрес подчиненного сервера *slave_name*, который передается главному серверу при реги-

страции. На главном сервере это значение выводится в результирующей таблице оператора `SHOW SLAVE HOSTS` *(см. разд. 32.6.7)*.

- `--report-port=`*`slave_port_num`*. Параметр определяет порт TCP/IP-соединения подчиненного сервера *`slave_port_num`*, который передается главному серверу при регистрации.
- `--skip-slave-start`. Данный параметр требует от подчиненного сервера не начинать репликацию при старте. Это часто требуется для технического обслуживания сервера и подготовки его к процедуре репликации. Запустить сервер можно в любой момент при помощи оператора `START SLAVE` *(см. разд. 32.7.8)*.
- `--slave_compressed_protocol={0|1}`. Параметр включает (1) или отключает (0) режим сжатия протокола обмена данных между главным и подчиненным серверами при условии, что оба сервера поддерживают такой режим.
- `--slave-load-tmpdir=`*`file_name`*. Параметр определяет имя каталога *`file_name`*, в котором подчиненный сервер создает временные файлы. Если параметр не указан, то принимается значение, хранящееся в системной переменной `tmpdir`. Когда поток обработки ретрансляционных журналов подчиненного сервера реплицирует оператор `LOAD DATA INFILE`, он извлекает файл, предназначенный для загрузки, из журнала ретрансляции в каталог временных файлов *`file_name`*.
- `--slave-net-timeout=`*`seconds`*. Время, которое выжидает подчиненный сервер данных от главного сервера пред тем, как прервать чтение из-за разрыва соединения. После этого будет осуществлена попытка подключиться повторно. Интервал между попытками определяется параметром `--master-connect-retry`.
- `--slave-skip-errors=[`*`err_code1`*`,`*`err_code2`*`,...|all]`. Обычно процесс репликации останавливается, когда возникает ошибка, чтобы оператор мог вручную устранить несогласованность данных. Если используется параметр `--slave-skip-errors`, то поток обработки ретрансляционных журналов не будет останавливаться при обнаружении ошибок, перечисленных в списке оператора `--slave-skip-errors`. Если вместо списка используется значение `all` — будут игнорироваться все ошибки. Для указания кодов ошибок следует использовать номера ошибок, возвращаемые оператором `SHOW SLAVE STATUS` *(см. разд. 32.7.7)*.

32.6. Операторы управления главным сервером

Репликацией можно управлять не только при помощи параметров, но также при помощи SQL-операторов. Данный раздел посвящен операторам управления главным сервером репликации.

32.6.1. Оператор *PURGE MASTER LOGS*

Оператор `PURGE MASTER LOGS` удаляет все бинарные журналы, перечисленные в индексе журнала, которые предшествуют заданному имени *log_name* или дате *date*. Оператор имеет следующий синтаксис:

```
PURGE {MASTER | BINARY} LOGS TO 'log_name'
PURGE {MASTER | BINARY} LOGS BEFORE 'date'
```

В результате выполнения оператора удаляемые журналы также удаляются из списка в индексном файле журнала, в результате чего заданный файл журнала становится первым в списке. В листинге 32.10 демонстрируется пример использования оператора `PURGE MASTER LOGS`.

Листинг 32.10. Использование оператора `PURGE MASTER LOGS`

```
PURGE MASTER LOGS TO 'mysql-bin.010';
PURGE MASTER LOGS BEFORE '2003-04-02 22:46:26';
```

Вариант с ключевым словом `BEFORE` появился в MySQL версии 4.1, дата и время суток в этом варианте представляется в формате `DATETIME` — `YYYY-MM-DD hh:mm:ss`. Ключевые слова `MASTER` и `BINARY` являются синонимами, но `BINARY` появилось, только начиная с версии MySQL 4.1.1.

Если в момент выполнения подчиненный сервер использует журнал, то оператор `PURGE MASTER LOGS` не делает ничего и завершается ошибкой.

32.6.2. Оператор *RESET MASTER*

Оператор `RESET MASTER` удаляет все бинарные журналы, перечисленные в индексном файле, очищает сам индексный файл и создает файл бинарного журнала.

```
RESET MASTER
```

Оператор не имеет дополнительных параметров и ключевых слов.

32.6.3. Оператор *SET SQL_BIN_LOG*

Оператор `SET SQL_BIN_LOG` включает (1) и выключает (0) бинарную регистрацию текущего соединения и имеет следующий синтаксис:

```
SET SQL_BIN_LOG = {0 | 1}
```

Для выполнения данного оператора необходима привилегия `SUPER`, в противном случае будет возвращена ошибка. До MySQL 4.1 в этом случае оператор попросту игнорировался.

32.6.4. Оператор *SHOW BINLOG EVENTS*

Оператор `SHOW BINLOG EVENTS` отображает записи, занесенные в бинарный журнал, и имеет следующий синтаксис:

```
SHOW BINLOG EVENTS
   [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

Если не указывается имя журнала *log_name*, отображается первый журнал. Необязательное ключевое слово `FROM` позволяет задать позицию *pos*, начиная с которой необходимо читать записи бинарного журнала. Конструкция `LIMIT` имеет тот же синтаксис, что и в операторе `SELECT` *(см. главу 7)*.

32.6.5. Оператор *SHOW MASTER LOGS*

Оператор `SHOW MASTER LOGS` выводит список файлов бинарных журналов на главном сервере и имеет следующий синтаксис:

```
SHOW MASTER LOGS
```

> ***Замечание***
>
> Оператор `SHOW MASTER LOGS` имеет синоним `SHOW BINARY LOGS`, который был введен, начиная с версии MySQL 4.1.1.

Пример использования оператора приводится в листинге 32.11.

Листинг 32.11. Использование оператора `SHOW MASTER LOGS`

```
mysql> SHOW MASTER LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000015 |    724935 |
| binlog.000016 |    733481 |
+---------------+-----------+
```

Поле `File_size` в результирующей таблице оператора `SHOW MASTER LOGS`, в котором отображается размер файла бинарного журнала в байтах, было введено, начиная с версии MySQL 5.0.7.

32.6.6. Оператор *SHOW MASTER STATUS*

Оператор `SHOW MASTER STATUS` предоставляет информацию о состоянии файлов бинарного журнала главного сервера и имеет следующий синтаксис:

```
SHOW MASTER STATUS
```

Пример использования оператора SHOW MASTER STATUS приводится в листинге 32.12.

Листинг 32.12. Использование оператора SHOW MASTER STATUS

```
mysql > SHOW MASTER STATUS;
+--------------+----------+-------------+------------------+
| File         | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+--------------+----------+-------------+------------------+
| mysql-bin.003 | 73       | test         | manual,mysql     |
+--------------+----------+-------------+------------------+
```

32.6.7. Оператор *SHOW SLAVE HOSTS*

Оператор SHOW SLAVE HOSTS выводит список подчиненных серверов, зарегистрированных в данный момент на главном сервере репликации. Оператор имеет следующий синтаксис:

```
SHOW SLAVE HOSTS
```

Любой подчиненный сервер, который не был запущен с параметром --report-host, не будет показан в списке.

32.7. Операторы управления подчиненными серверами

В данном разделе описываются SQL-операторы, которые позволяют управлять процессом репликации на подчиненном сервере.

32.7.1. Оператор *CHANGE MASTER TO*

Оператор CHANGE MASTER TO позволяет изменять параметры, которые подчиненный сервер использует для подключения и взаимодействия с ведущим сервером. Оператор имеет следующий синтаксис:

```
CHANGE MASTER TO master_def [, master_def] ...
master_def:
    MASTER_HOST = 'host_name'
  | MASTER_USER = 'user_name'
  | MASTER_PASSWORD = 'password'
  | MASTER_PORT = port_num
  | MASTER_CONNECT_RETRY = count
  | MASTER_LOG_FILE = 'master_log_name'
```

```
| MASTER_LOG_POS = master_log_pos
| RELAY_LOG_FILE = 'relay_log_name'
| RELAY_LOG_POS = relay_log_pos
| MASTER_SSL = {0|1}
| MASTER_SSL_CA = 'ca_file_name'
| MASTER_SSL_CAPATH = 'ca_directory_name'
| MASTER_SSL_CERT = 'cert_file_name'
| MASTER_SSL_KEY = 'key_file_name'
| MASTER_SSL_CIPHER = 'cipher_list'
```

Параметры SSL, такие как MASTER_SSL, MASTER_SSL_CA, MASTER_SSL_CAPATH, MASTER_SSL_CERT, MASTER_SSL_KEY и MASTER_SSL_CIPHER доступны, только начиная с MySQL 4.1.1. Указанные параметры можно применять даже на тех подчиненных серверах, которые были скомпилированы без поддержки SSL. В этом случае, параметры сохраняются в файл master.info, но игнорируются до тех пор, пока сервер не будет заменен на версию с поддержкой SSL.

Если какой-либо параметр не указывается, его значение остается неизменным в файле master.info. Например, смена пароля для подключения к главному серверу может выглядеть так, как это представлено в листинге 32.13.

Листинг 32.13. Смена пароля доступа к главному серверу

```
STOP SLAVE;  -- если репликация была запущена
CHANGE MASTER TO MASTER_PASSWORD='new3cret';
START SLAVE; -- если необходимо перезапустить репликацию
```

Следует заметить, что все остальные параметры, которые не указываются в операторе CHANGE MASTER TO, остаются неизменными.

Параметры MASTER_HOST и MASTER_PORT позволяют задать имя хоста (или IP-адрес), на котором расположен главный сервер, и его TCP/IP-порт, соответственно. При указании любого из этих двух параметров подчиненный сервер предполагает, что адрес главного сервера изменился (даже если указываются старые значения), а старые имена файлов бинарного журнала и позиции в них рассматриваются как недействительные. Поэтому, если одновременно с параметрами MASTER_HOST и MASTER_PORT не указываются параметры MASTER_LOG_FILE и MASTER_LOG_POS, они принимают значения пустой строки и 4, соответственно.

Параметры MASTER_LOG_FILE и MASTER_LOG_POS являются своеобразными координатами, по которым поток ввода/вывода подчиненного сервера определяет место, с которого необходимо начинать чтение бинарного журнала главного сервера при следующем запуске.

Оператор `CHANGE MASTER TO` удаляет все файлы журналов ретрансляции и начинает новый журнал, если только не были указаны параметры `RELAY_LOG_FILE` и `RELAY_LOG_POS`. В этом случае журнал ретрансляций сохраняется.

Оператор `CHANGE MASTER TO` применяется для настройки подчиненного сервера репликации перед его первым запуском. Для этого необходим SQL-дамп главного сервера, имя файла бинарного журнала и смещение в нем. Последние получают на момент получения SQL-дампа. В листинге 32.14 приводится типичный пример использования оператора `CHANGE MASTER TO` для настройки подчиненного сервера перед пуском.

Листинг 32.14. Подготовка подчиненного сервера для старта

```
CHANGE MASTER TO
  MASTER_HOST='master2.mycompany.com',
  MASTER_USER='replication',
  MASTER_PASSWORD='bigs3cret',
  MASTER_PORT=3306,
  MASTER_LOG_FILE='master2-bin.001',
  MASTER_LOG_POS=4,
  MASTER_CONNECT_RETRY=10;
```

Оператор, приведенный в листинге 32.15, применяется в том случае, если по каким-либо причинам необходимо повторно выполнить восстановление из ретрансляционных журналов.

Листинг 32.15. Повторное чтение журналов ретрансляции

```
CHANGE MASTER TO
  RELAY_LOG_FILE='slave-relay-bin.006',
  RELAY_LOG_POS=4025;
```

32.7.2. Оператор *LOAD DATA FROM MASTER*

Оператор `LOAD DATA FROM MASTER` извлекает информацию с главного сервера и загружает ее на подчиненный сервер (кроме базы данных `mysql` и баз данных, репликация которых запрещена при помощи параметров `--replicate-ignore-*`). Оператор имеет следующий синтаксис:

```
LOAD DATA FROM MASTER
```

В результате запуска оператора также обновляются значения `MASTER_LOG_FILE` и `MASTER_LOG_POS` таким образом, чтобы репликация начиналась с правильной позиции.

Оператор работает только с таблицами типа MyISAM и требует глобальной блокировки чтения на главном сервере в процессе загрузки данных.

> ***Замечание***
>
> При загрузке больших таблиц может понадобиться увеличить значения системных переменных `net_read_timeout` и `net_write_timeout` как на главном, так и на подчиненном серверах.

Оператор `LOAD DATA FROM MASTER` требует наличия отдельной учетной записи с привилегиями `RELOAD` и `SUPER`, а также привилегию `SELECT` на все таблицы главного сервера, которые необходимо загружать. Если привилегия `SELECT` отсутствует, такая база данных будет проигнорирована оператором `LOAD DATA FROM MASTER`. Последнее связано с тем, что оператор `LOAD DATA FROM MASTER` использует оператор `SHOW DATABASE` для получения списка баз данных для загрузки, а `SHOW DATABASE` отображает только те базы данных, на которые у пользователя имеются хоть какие-то привилегии *(см. главу 27)*. Со стороны подчиненного сервера пользователь, выполняющий оператор `LOAD DATA FROM MASTER`, должен иметь привилегии на удаление и создание баз данных и таблиц, которые подлежат копированию.

32.7.3. Оператор *LOAD TABLE FROM MASTER*

Оператор `LOAD TABLE FROM MASTER` предназначен для переноса копии таблицы с главного сервера на подчиненный и имеет следующий синтаксис:

```
LOAD TABLE tbl FROM MASTER
```

> ***Замечание***
>
> Оператор `LOAD TABLE FROM MASTER` работает только с таблицами MyISAM.

Здесь *`tbl`* — имя переносимой таблицы. Оператор создан главным образом для отладки оператора `LOAD DATA FROM MASTER` и требует привилегий `RELOAD` и `SUPER`, а также привилегию `SELECT` на загружаемую таблицу. Со стороны подчиненного сервера пользователь, выполняющий оператор `LOAD TABLE FROM MASTER`, должен иметь привилегии на удаление и создание баз данных и таблиц, которые подлежат копированию.

32.7.4. Функция *MASTER_POS_WAIT()*

Функция `MASTER_POS_WAIT()` имеет следующий синтаксис:

```
SELECT MASTER_POS_WAIT('master_log_file', master_log_pos)
```

Функция применяется для того, чтобы подчиненный сервер прочитал и выполнил все операторы из бинарного журнала *`master_log_file`* главного сервера до позиции *`master_log_pos`*.

32.7.5. Оператор *RESET SLAVE*

Оператор `RESET SLAVE` предписывает подчиненному серверу "забыть" позицию репликации в бинарном журнале главного сервера. Оператор имеет следующий синтаксис:

```
RESET SLAVE
```

В результате выполнения оператора удаляются файлы master.info и relay-log.info и все старые журналы ретрансляций.

> ***Замечание***
>
> Все журналы ретрансляций удаляются, даже если они не были проанализированы потоком обработки подчиненного сервера.

Важно отметить, что в результате выполнения оператора теряется вся информация о параметрах подключения к главному серверу, которая хранится в файле master.info. Для возобновления репликации придется устанавливать параметры заново при помощи оператора `CHANGE MASTER TO` *(см. разд. 32.7.1)*.

32.7.6. Оператор *SET GLOBAL SQL_SLAVE_SKIP_COUNTER*

Оператор `SET GLOBAL SQL_SLAVE_SKIP_COUNTER` имеет следующий синтаксис:

```
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = n
```

Данный оператор, будучи запущен на подчиненном сервере, позволяет пропустить *n* записей в бинарном журнале главного сервера.

32.7.7. Оператор *SHOW SLAVE STATUS*

Оператор `SHOW SLAVE STATUS` предоставляет информацию о состоянии параметров потоков подчиненного сервера и имеет следующий синтаксис:

```
SHOW SLAVE STATUS
```

Пример использования оператора приводится в листинге 32.16.

Листинг 32.16. Использование оператора `SHOW SLAVE STATUS`

```
mysql> 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. Оператор возвращает следующие поля.

- `Slave_IO_State`. Данное поле является копией поля `State` из отчета оператора `SHOW PROCESSLIST` для потока ввода/вывода подчиненного сервера *(см. разд. 32.2)*. Поле информирует о том, что поток пытается подключиться к главному серверу, ожидает событий с главного сервера, повторно подключается и т. п. Поле `Slave_IO_State` появилось в MySQL версии 4.1.1.
- `Master_Host`. Данное поле информирует об имени (или IP-адресе) главного сервера.
- `Master_User`. Поле информирует о пользователе, от имени которого подчиненный сервер подключается к главному серверу.
- `Master_Port`. Поле информирует о номере порта, который прослушивает главный сервер.
- `Connect_Retry`. Поле информирует о текущем значении параметра `--master-connect-retry`.

- `Master_Log_File`. Поле содержит имя файла бинарного журнала на главном сервере, из которого поток ввода/вывода осуществляет чтение.
- `Read_Master_Log_Pos`. Поле содержит позицию бинарного журнала главного сервера, на которой остановился поток ввода/вывода.
- `Relay_Log_File`. Поле содержит имя журнала ретрансляции, из которого поток обработки ретрансляционного журнала осуществляет чтение и выполнение операторов.
- `Relay_Log_Pos`. Поле содержит позицию в журнале ретрансляции, до которой поток обработки ретрансляционного журнала прочитал и выполнил операторы.
- `Relay_Master_Log_File`. Поле содержит имя файла бинарного журнала главного сервера, который содержит последний оператор, выполненный потоком обработки ретрансляционных журналов.
- `Slave_IO_Running`. Поле принимает значение `YES`, если запущен поток ввода/вывода, и пустую строку в противном случае.
- `Slave_SQL_Running`. Поле принимает значение `YES`, если запущен поток обработки ретрансляционных журналов, и пустую строку в противном случае.
- `Replicate_Do_DB`. Поле содержит список баз данных, которые были указаны в параметре `--replication-do-db`.
- `Replicate_Ignore_DB`. Поле содержит список баз данных, которые были указаны в параметре `--replication-ignore-db`.
- `Replicate_Do_Table`. Поле содержит список таблиц, которые были указаны в параметре `--replication-do-table`.
- `Replicate_Ignore_Table`. Поле содержит список таблиц, которые были указаны в параметре `--replication-ignore-table`.
- `Replicate_Wild_Do_Table`. Поле содержит список таблиц, которые были указаны в параметре `--replication-wild-do-table`.
- `Replicate_Wild_Ignore_Table`. Поле содержит список таблиц, которые были указаны в параметре `--replication-wild-ignore-table`.
- `Last_Errno`, `Last_Error`. Поля содержат номер и сообщение об ошибке, соответственно. Если номер ошибки `Last_Errno` равен 0, а сообщение `Last_Error` равно пустой строке, то ошибки не происходили. Если поле `Last_Error` не пустое, то сообщение из этого поля также появляется в журнале ошибок подчиненного сервера.
- `Skip_Counter`. Поле содержит последнее использованное значение `SQL_SLAVE_SKIP_COUNTER`.
- `Exec_Master_Log_Pos`. Поле содержит позицию последнего исполненного потоком обработки ретрансляционного журнала оператора в бинарном журнале главного сервера.
- `Relay_Log_Space`. Суммарный размер всех существующих журналов ретрансляции.

- `Until_Condition`, `Until_Log_File`, `Until_Log_Pos`. Поля содержат значения, указанные в конструкции `UNTIL` оператора `START SLAVE` *(см. разд. 32.7.8)*. Эти поля были добавлены, начиная с версии MySQL 4.1.1. Поля `Until_Log_File`, `Until_Log_Pos` обозначают имя файла журнала и позицию в нем, определяющие точку, в которой поток обработки ретрансляционных журналов прекратит свое выполнение. Поле `Until_Condition` может принимать следующие значения:
  - `None`, если конструкция `UNTIL` не указывалась;
  - `Master`, если подчиненный сервер читает до заданной позиции в бинарных журналах главного сервера;
  - `Relay`, если подчиненный сервер читает до заданной позиции в своих журналах ретрансляции.
- `Master_SSL_Allowed`, `Master_SSL_CA_File`, `Master_SSL_CA_Path`, `Master_SSL_Cert`, `Master_SSL_Cipher`, `Master_SSL_Key`. В этих полях отображаются параметры SSL, используемые подчиненным сервером для подключения к главному серверу. Эти поля были добавлены, начиная с версии MySQL 4.1.1. Поле `Master_SSL_Allowed` принимает следующие значения:
  - `Yes`, если SSL-подключение к главному серверу разрешено;
  - `No`, если SSL-подключение к главному серверу не разрешено;
  - `Ignored`, если SSL-подключение к главному серверу подключено, но подчиненный сервер не поддерживает SSL.
- `Seconds_Behind_Master`. Поле содержит количество секунд, прошедших от временной метки с момента выполнения потоком обработки ретрансляционных журналов последнего события с главного сервера. Поле принимает значение `NULL`, если еще не выполнялся ни один из операторов, либо сразу после выполнения операторов `CHANGE MASTER` или `RESET MASTER`. Это поле может применяться для оценки того, насколько "опаздывает" сервер. Поле было добавлено, начиная с версии MySQL 4.1.1.

32.7.8. Оператор *START SLAVE*

Оператор `START SLAVE` предназначен для запуска подчиненного сервера репликации и имеет следующий синтаксис:

```
START SLAVE [thread_type [, thread_type] ... ]
START SLAVE [SQL_THREAD] UNTIL
    MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos
START SLAVE [SQL_THREAD] UNTIL
    RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos
thread_type: IO_THREAD | SQL_THREAD
```

> ***Замечание***
>
> Запуск оператора `START SLAVE` требует привилегии `SUPER`.

Оператор START SLAVE без параметров запускает репликацию на подчиненном сервере. При этом запускаются оба потока: как поток ввода/вывода, так и поток обработки журналов ретрансляции. При помощи параметра *thread_type* можно уточнить, какой из потоков нужно запустить: поток ввода/вывода IO_THREAD или поток обработки журналов SQL_THREAD.

Начиная с версии MySQL 4.1.1, можно задавать конструкцию UNTIL для указания того, что поток обработки ретрансляционных журналов должен работать до тех пор, пока не будет достигнута заданная точка в бинарном журнале или журнале ретрансляции. Точка в бинарном журнале главного сервера задается при помощи параметров MASTER_LOG_FILE и MASTER_LOG_POS, которые определяют название файла и позицию от начала файла, соответственно. Для ретрансляционного журнала точка задается при помощи параметров RELAY_LOG_FILE и RELAY_LOG_POS, которые также определяют имя журнала и позицию от начала файла.

Конструкция UNTIL сбрасывается последующим вызовом оператора SLAVE STOP или START SLAVE без конструкции UNTIL, либо перезапуском сервера MySQL. Конструкция UNTIL может оказаться удобной для отладки репликации либо для выполнения репликации только до определенной точки, в которой нужно избежать репликации отдельного оператора.

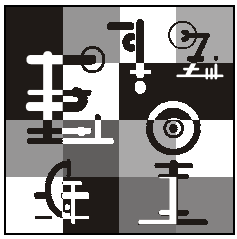
32.7.9. Оператор *STOP SLAVE*

Оператор STOP SLAVE останавливает подчиненный сервер и имеет следующий синтаксис:

```
STOP SLAVE [thread_type [, thread_type] ... ]
thread_type: IO_THREAD | SQL_THREAD
```

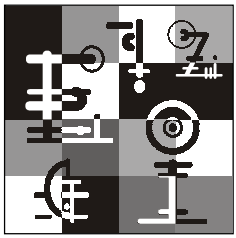
> ***Замечание***
>
> Запуск оператора START SLAVE требует привилегии SUPER.

При помощи параметра *thread_type* можно уточнить, какой из потоков репликации следует остановить: поток ввода/вывода IO_THREAD или поток обработки журналов SQL_THREAD.



ЧАСТЬ IV

НОВОВВЕДЕНИЯ MYSQL 5.0



Глава 33

Хранимые процедуры

Часто при выполнении рутинных операций требуется производить одинаковые запросы. *Хранимые процедуры* позволяют объединить последовательность таких запросов и сохранить их на сервере. После этого клиентам не придется посылать серверу последовательность запросов, достаточно отправить один запрос на выполнение хранимой процедуры.

> ***Замечание***
>
> Хранимые процедуры появились в СУБД MySQL, начиная с версии 5.0.0.

Хранимые процедуры обладают рядом преимуществ.

- *Повторное использование кода.* После того как хранимая процедура создана, ее можно вызывать из любых приложений и SQL-запросов, больше не требуется каждый раз набирать одни и те же операторы, благодаря чему снижается риск проникновения в приложения программных ошибок и уменьшается время разработки программы.
- *Сокращение сетевого трафика.* При использовании хранимых процедур вместо того, чтобы отсылать по сети каждый запрос и получать на каждый из запросов ответ (в том числе и промежуточные результирующие таблицы), гораздо экономнее послать серверу запрос на выполнение хранимой процедуры и сразу получить окончательный ответ.
- *Безопасность.* Хранимые процедуры используются для всех стандартных банковских операций. Процедура гарантирует, что последовательность действий будет согласованной и не приведет к нарушению целостности данных из-за того, что один оператор не будет выполнен. Кроме того, для выполнения хранимой процедуры пользователь должен иметь соответствующую привилегию. При этом права доступа к таблицам иметь вовсе не обязательно. Таким образом, администратор базы данных получает более широкие возможности в плане защиты данных и управления доступом пользователей к объектам базы данных.
- *Простота доступа.* Основным императивом разработки программного обеспечения провозглашено уменьшение сложности кода. Хранимые процедуры позволяют

инкапсулировать сложный код и оформить его в виде простого вызова с осмысленным именем. При создании нового каталога гораздо проще оперировать процедурой `CREATE_NEW_CATALOG()`, чем несколькими операторами непонятного назначения. При регистрации сделки, требующей поиска в таблицах `catalogs`, `products`, `users` и редактирования таблиц `orders` и `users`, гораздо проще оформить последовательность SQL-операторов для оформления сделки в хранимую процедуру `ORDERING()`. Даже если последовательность операторов хорошо знакома программисту, применение хранимых процедур позволяет перейти на более высокий уровень абстракции и оперировать не таблицами, а сущностями реального мира, такими как сделка, каталог продукции и др.

- *Выполнение деловой логики.* Хранимые процедуры могут выполнять проверку условий выполнения заказа, например, подсчитывая число товарных позиций на складе, и отклонять заказ в случае, если его недостаточно. Это позволяет перенести код сохранения целостности базы данных из прикладной программы на сервер базы данных. Так как прикладная программа может выполняться на сотне машин сети, а сервер, как правило, один, устойчивость системы при перемещении логики на сервер резко возрастает. Кроме того, в этом случае легче сменить язык разработки внешнего приложения, т. к. большая часть логики оформлена в виде хранимых процедур и не зависит от языка разработки внешнего приложения.

33.1. Хранимые процедуры и привилегии

Начиная с версии 5.0.3, СУБД MySQL требует от пользователей наличия следующих привилегий при работе с хранимыми процедурами:

- для создания хранимых процедур необходимо наличие привилегии `CREATE ROUTINE`;
- для редактирования и удаления хранимой процедуры необходимо наличие привилегии `ALTER ROUTINE`, пользователь, создающий хранимую процедуру, автоматически наделяется этой привилегией;
- для вызова хранимой процедуры необходимо наличие привилегии `EXECUTE`, эта привилегия также автоматически передается пользователю, создавшему хранимую процедуру.

> ***Замечание***
>
> Более подробно с системой привилегий СУБД MySQL можно ознакомиться в *главе 27*.

33.2. Создание хранимой процедуры

Для создания хранимой процедуры существуют два оператора: `CREATE PROCEDURE` и `CREATE FUNCTION`, которые имеют следующий синтаксис:

```
CREATE PROCEDURE sp_name ([parameter[, ...]])
    [characteristic ...] routine_body
```

```
CREATE FUNCTION sp_name ([parameter[, ...]])
    RETURNS type
    [characteristic ...] routine_body
```

Здесь *sp_name* — имя хранимой процедуры. 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` — значение этого параметра как принимается во внимание внутри процедуры, так и сохраняет свое значение по выходу из нее.

> ***Замечание***
>
> Список аргументов, заключенных в круглые скобки, должен присутствовать всегда. Если аргументы отсутствуют, следует использовать пустой список аргументов `()`.

После имени параметра *param_name* указывается его тип, который может быть одним из типов, рассмотренных в *главе 4*. По умолчанию, если ни один из модификаторов не указан, считается, что параметр объявлен с ключевым словом `IN`.

Оператор `CREATE FUNCTION` позволяет задать пользовательскую функцию, т. е. такой вид хранимой процедуры, который 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'
```

Подробнее эти ключевые слова рассматриваются в *разд. 33.3*.

33.2.1. Тело процедуры

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*.

Оператор BEGIN...END может выглядеть так, как это представлено в листинге 33.1.

Листинг 33.1. Использование составного оператора BEGIN...END

```
BEGIN
  UPDATE tbl1 SET col1 = '1234.56';
  UPDATE tbl2 SET col2 = '1234.56';
END
```

В качестве одного из операторов внутри составного оператора BEGIN...END может выступать другой составной оператор (листинг 33.2).

Листинг 33.2. Вложенный составной оператор BEGIN...END

```
BEGIN
  UPDATE tbl1 SET col1 = '1234.56';
  inner: BEGIN
    UPDATE tbl2 SET col2 = '1234.56';
    UPDATE tbl3 SET col3 = '1234.56';
  END inner
END
```

Как видно из листинга 33.2, во внутреннем составном операторе BEGIN...END перед оператором BEGIN используется метка inner, которая обязательно упоминается и в операторе END. Следует отметить, что если хранимая процедура содержит только один запрос, то можно не использовать составной оператор BEGIN...END.

> ***Замечание***
>
> После оператора END можно как помещать, так и не помещать точку с запятой. Здесь и далее точка с запятой после END помещаться не будет.

Основная трудность, которая возникает при работе с хранимыми процедурами, заключается в том, что обязательный символ точки с запятой (`;`) в конце каждого запроса воспринимается консольным клиентом 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 33.3). В этом случае для обозначения окончания ввода вместо точки с запятой необходимо будет использовать последовательность `//`.

> ***Замечание***
>
> Подробнее консольный клиент `mysql` и его параметры рассматриваются в *главе 3*.

Листинг 33.3. 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 33.4).

Листинг 33.4. Смена разделителя запросов при помощи команды `DELIMITER`

```
mysql> DELIMITER //
mysql> SELECT VERSION()//
+-----------------+
| VERSION()       |
+-----------------+
| 5.0.18-standard |
+-----------------+
mysql> DELIMITER ;
mysql> SELECT VERSION();
+-----------------+
| VERSION()       |
+-----------------+
| 5.0.18-standard |
+-----------------+
```

Имя хранимой процедуры не может превышать 64 символа и не зависит от регистра, т. е. имена `numcatalogs()`, `Numcatalogs()` и `NUMCATALOGS()` являются эквивалентными. Пример создания простейшей хранимой процедуры демонстрируется в листинге 33.5.

Листинг 33.5. Создание хранимой процедуры `my_version()`

```
mysql> CREATE PROCEDURE my_version ()
    -> BEGIN
    ->   SELECT VERSION();
    -> END
    -> //
```

Если тело процедуры содержит единственный запрос, можно не использовать составной оператор `BEGIN...END`. Определения хранимой процедуры `my_version()` в листингах 33.5 и 33.6 эквивалентны.

Листинг 33.6. Альтернативное определение хранимой процедуры `my_version()`

```
mysql> CREATE PROCEDURE my_version ()
    ->   SELECT VERSION();
    -> //
```

Функция `my_version()` ничего не делает, кроме того, что выводит версию сервера MySQL. Каким образом можно воспользоваться результатами запроса? Для того чтобы вызвать хранимую процедуру, необходимо применить оператор `CALL`, после которого помещается имя процедуры и ее параметры в круглых скобках (листинг 33.7).

Листинг 33.7. Вызов хранимой процедуры при помощи ключевого слова `CALL`

```
mysql> CALL my_version ();
+-----------------+
| VERSION()       |
+-----------------+
| 5.0.18-standard |
+-----------------+
```

При вызове хранимой процедуры, в отличие от встроенных функций, между именем функции и круглыми скобками допускается пробел. В листинге 33.8 оба запроса являются эквивалентными.

Листинг 33.8. Пробел после имени функции допустим

```
mysql> CALL my_version();
mysql> CALL my_version ();
```

При именовании функций следует избегать названий, совпадающих с именами внутренних функций MySQL (листинг 33.9).

Листинг 33.9. Хранимая процедура `pi ()`

```
mysql> CREATE PROCEDURE pi ()
    -> BEGIN
    ->   SELECT VERSION();
    -> END
    -> //
```

В листинге 33.9 объявляется хранимая процедура `pi ()`, которая выводит текущую версию сервера MySQL. Следует обратить внимание, что между именем процедуры и круглыми скобками помещен пробел — в противном случае СУБД MySQL интерпретирует последовательность `pi()` как вызов встроенной функции, возвращающей число π. При вызове хранимой процедуры использование пробела в этом случае также является обязательным (листинг 33.10). Перегрузка функций в MySQL не предусмотрена.

Листинг 33.10. Вызов хранимой процедуры `pi ()`

```
mysql> CALL pi();
ERROR 1064 (42000): 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 'pi()' at line 1
mysql> CALL pi ();
+-----------------+
| VERSION()       |
+-----------------+
| 5.0.18-standard |
+-----------------+
```

Во избежание подобных накладок рекомендуется избегать использования названий хранимых процедур, совпадающих с именами встроенных функций MySQL. Если все же это необходимо, то и в определении функции, и при ее вызове следует использовать пробел между именем хранимой процедуры и круглыми скобками.

Какие операторы допустимы в теле хранимых процедур? Любые, включая `INSERT`, `UPDATE`, `DELETE`, `SELECT`, `DROP`, `REPLACE` и др. (листинг 33.11).

Листинг 33.11. Примеры определения хранимых процедур

```
mysql> CREATE PROCEDURE p () DELETE FROM t; //
mysql> CREATE PROCEDURE p () SET @x = 5; //
mysql> CREATE PROCEDURE p () DROP TABLE t; //
mysql> CREATE PROCEDURE p () SELECT 'A' ; //
```

В теле хранимой процедуры можно использовать многострочный комментарий в стиле языка C, который начинается с последовательности `/*` и заканчивается последовательностью `*/` (листинг 33.12).

Листинг 33.12. Использование комментария в теле функции

```
mysql> CREATE PROCEDURE my_version ()
    -> BEGIN
    ->  /* Многострочный комментарий внутри
    ->     функции my_version()
    ->  */
    ->  SELECT VERSION(); /* Вызов единственного оператора */
    -> END
    -> //
```

33.2.2. Параметры процедуры

В предыдущем разделе хранимые процедуры не содержали параметров. Как упоминалось ранее, каждый параметр может быть снабжен одним из модификаторов `IN`, `OUT` или `INOUT`. В листинге 33.13 приводится пример функции, которая присваивает пользовательской переменной `@x` новое значение.

Листинг 33.13. Использование ключевого слова `IN`

```
mysql> USE test//
Database changed
mysql> CREATE PROCEDURE set_x (IN value INT)
    -> BEGIN
    ->   SET @x = value;
    -> END
    -> //
mysql> CALL set_x(123456)//
```

```
mysql> SELECT @x//
+--------+
| @x     |
+--------+
| 123456 |
+--------+
```

Как видно из листинга 33.13, через параметр `value` функции передается числовое значение `123456`, которое она присваивает пользовательской переменной `@x`. Модификатор `IN` сообщает СУБД MySQL, что при помощи параметра `value` пользователи передают данные внутрь функции.

> ***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.

Использование ключевого слова `IN` не является обязательным — если ни один из модификаторов не указан, СУБД MySQL считает, что параметр объявлен с модификатором `IN` (листинг 33.14).

Листинг 33.14. Использование ключевого слова `IN`

```
mysql> USE test//
Database changed
mysql> CREATE PROCEDURE set_y (value INT)
    -> BEGIN
    ->   SET value = 7;
    ->   SET @x = value;
    -> END
    -> //
mysql> SET @val = 123456//
mysql> CALL set_y(@val)//
mysql> SELECT @x, @val//
+------+--------+
| @x   | @val   |
+------+--------+
| 7    | 123456 |
+------+--------+
```

> ***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.

Замечание

Подробнее пользовательские переменные обсуждаются в *главе 23*.

Хранимая процедура set_y() принимает единственный IN-параметр value, при помощи оператора SET значение параметра изменяется внутри функции. Однако, как видно из листинга 33.14, после выполнения хранимой процедуры значение пользовательской переменной @val, переданной функции в качестве параметра, не изменяется. Если требуется, чтобы значение переменной подвергалось изменению, необходимо объявить параметр процедуры с модификатором OUT (листинг 33.15).

Листинг 33.15. Использование ключевого слова OUT

```
mysql> USE test//
Database changed
mysql> CREATE PROCEDURE set_y (OUT value INT)
    -> BEGIN
    ->   SET @x = value;
    ->   SET value = 7;
    -> END
    -> //
mysql> SET @val = 123456//
mysql> CALL set_y(@val)//
mysql> SELECT @x, @val//
+------+------+
| @x   | @val |
+------+------+
| NULL | 7    |
+------+------+
```

Как видно из листинга 33.15, 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 переменные не инициируются при помощи оператора SET или ключевого слова DEFAULT, они получают значение NULL.

Для того чтобы через параметр можно было и передать значение внутрь процедуры, и получить значение, которое попадает в параметр в результате вычислений внутри процедуры, его следует объявить с модификатором `INOUT` (листинг 33.16).

Листинг 33.16. Использование ключевого слова `INOUT`

```
mysql> USE test//
Database changed
mysql> CREATE PROCEDURE set_y (INOUT value INT)
    -> BEGIN
    ->   SET @x = value;
    ->   SET value = 7;
    -> END
    -> //
mysql> SET @val = 123456//
mysql> CALL set_y(@val)//
mysql> SELECT @x, @val//
+--------+------+
| @x     | @val |
+--------+------+
| 123456 | 7    |
+--------+------+
```

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.

33.2.3. Работа с таблицами базы данных

Все процедуры, рассмотренные выше, не использовали в своей работе таблицы. Далее рассмотрим несколько процедур, осуществляющих запросы к таблицам базы данных.

Создадим функцию `numcatalogs()`, которая подсчитывает количество записей в таблице `catalogs` учебной базы данных `shop` (листинг 33.17).

Листинг 33.17. Создание хранимой процедуры `numcatalogs()`

```
mysql> USE shop
Database changed
mysql> CREATE PROCEDURE numcatalogs (OUT total INT)
```

```
    -> BEGIN
    ->   SELECT COUNT(*) INTO total FROM catalogs;
    -> END
    -> //
```

Хранимая процедура `numcatalogs()` имеет один целочисленный (`INT`) параметр `total`, в который сохраняется число записей в таблице `catalogs`. Осуществляется это при помощи оператора `SELECT...INTO...FROM`, который позволяет сохранять результаты непосредственно в выходном параметре `total` функции `numcatalogs`. Данный оператор позволяет оперировать сразу несколькими столбцами (листинг 33.18).

Листинг 33.18. Использование оператора `SELECT...INTO...FROM`

```
SELECT id, data INTO x, y FROM test LIMIT 1;
```

Таким образом, параметр `total` после выполнения хранимой процедуры содержит число записей в таблице `catalogs` (листинг 33.19).

Листинг 33.19. Вызов хранимой процедуры `numcatalogs()`

```
mysql> USE shop;
mysql> CALL numcatalogs(@a);
mysql> SELECT @a;
+------+
| @a   |
+------+
| 5    |
+------+
```

Как видно из листинга 33.19, в качестве параметра функции `numcatalogs()` передается пользовательская переменная `@a`.

СУБД поддерживает контекст вызова хранимой процедуры для базы данных по умолчанию. Это означает, что если хранимая процедура, созданная в базе данных `shop`, будет вызвана в тот момент, когда текущей базой данных является база `test` — СУБД MySQL вернет ошибку (листинг 33.20).

Листинг 33.20. Вызов хранимой процедуры `numcatalogs()` в контексте базы `test`

```
mysql> USE test;
mysql> CALL numcatalogs(@a);
ERROR 1305: PROCEDURE test.numcatalogs does not exist
```

Хранимая процедура наследует базу данных по умолчанию от вызывающего оператора, поэтому при обращении к таблицам других баз данных необходимо использовать расширенные имена.

> ***Замечание***
>
> Использование оператора `USE` в хранимых процедурах запрещено.

Создадим хранимую процедуру `catalogname()`, которая будет возвращать по первичному ключу `id_catalog` название каталога `name` (листинг 33.21). Для этого потребуется определить параметр `id_catalog` с атрибутом `IN`, а `name` с атрибутом `OUT`.

Листинг 33.21. Создание хранимой процедуры `catalogname()`

```
mysql> CREATE PROCEDURE catalogname (IN id INT, OUT catalog TINYTEXT)
    -> BEGIN
    ->   SELECT name INTO catalog FROM catalogs
    ->   WHERE id_catalog = id;
    -> END
    -> //
mysql> SET @id := 5//
mysql> CALL catalogname(@id, @name)//
mysql> SELECT @id, @name//
+------+--------------------+
| @id  | @name              |
+------+--------------------+
| 5    | Оперативная память |
+------+--------------------+
mysql> CALL catalogname(1, @name)//
mysql> SELECT @name//
+------------+
| @name      |
+------------+
| Процессоры |
+------------+
```

Примером функции, использующей параметр типа `INOUT`, может послужить функция `count_by_id()`, которая, принимая в качестве параметра первичный ключ каталога из таблицы `catalogs`, возвращает число товарных позиций в данном каталоге (листинг 33.22).

Листинг 33.22. Создание хранимой процедуры `count_by_id()`

```
mysql> CREATE PROCEDURE count_by_id (INOUT id INT)
    -> BEGIN
    ->   SELECT COUNT(*) INTO id FROM products
    ->   WHERE id_catalog = id;
    -> END
    -> //
mysql> SET @id = 3//
mysql> CALL count_by_id(@id)//
mysql> SELECT @id//
+------+
| @id  |
+------+
| 4    |
+------+
```

Однако применения `INOUT`-параметров следует всячески избегать, т. к. при их использовании возникает соблазн нарушить правила хорошего стиля программирования, как в листинге 33.22, где пользовательская переменная `@id` сначала содержала первичный ключ каталога, а после вызова содержит уже число товарных позиций в данном каталоге. Использование переменных в этом духе непременно приводит к возникновению ошибок. Лучше вообще отказаться от использования `INOUT`-переменных даже в том случае, когда функция принимает и возвращает однотипные данные (например, первичный ключ), лучше создать два параметра: один для входного значения, другой — для выходного.

Оператор `SELECT...INTO...FROM` не возвращает результат запроса клиенту непосредственно, в отличие от обычного оператора `SELECT...FROM`, который отправляет результат клиенту, даже если вызывается внутри хранимой процедуры (листинг 33.23).

Листинг 33.23. Использование обычного оператора `SELECT`

```
mysql> CREATE PROCEDURE numcatalogsview ()
    -> BEGIN
    ->   SELECT COUNT(*) FROM catalogs;
    -> END
    -> //
mysql> CALL numcatalogsview ()//
```

```
+----------+
| COUNT(*) |
+----------+
|        5 |
+----------+
```

Как видно из листинга 33.23, функция `numcatalogsview()` позволяет добиться тех же результатов, что и `numcatalogs()` (см. листинг 33.17) без использования параметров и пользовательских переменных. Однако использование параметров оправдано, т. к. позволяет создавать более читабельный код и использовать результат функции при вызове ее из другой функции. Создадим функцию `catalog_by_product()`, которая по имени товарной позиции возвращает имя элемента каталога, к которому относится данная товарная позиция. В ходе создания новой функции будем опираться на ранее созданную функцию `catalogname()`, которая возвращает имя элемента каталога по его первичному ключу `id_catalog` (листинг 33.21). Код новой функции представлен в листинге 33.24.

Листинг 33.24. Создание хранимой процедуры `catalog_by_product()`

```
mysql> CREATE PROCEDURE catalog_by_product (IN product TINYTEXT,
    ->                                      OUT catalog TINYTEXT)
    -> BEGIN
    ->   DECLARE id INT;
    ->   SELECT id_catalog INTO id FROM products
    ->   WHERE name = product LIMIT 1;
    ->   CALL catalogname(id, catalog);
    -> END
    -> //
mysql> CALL catalog_by_product('Celeron 1.8', @catalogname)//
mysql> SELECT @catalogname//
+--------------+
| @catalogname |
+--------------+
| Процессоры   |
+--------------+
```

Как видно из листинга 33.24, функция `catalog_by_product()` имеет два параметра:

- `product` — входной параметр, через который передается название товарной позиции;
- `catalog` — выходной параметр, через который можно получить результат работы функции — название элемента каталога, в который входит товарная позиция.

По названию товарной позиции product при помощи SELECT-запроса определяется первичный ключ каталога. Полученное значение помещается во временную переменную id, которая передается в качестве первого аргумента функции catalogname(), возвращающей название элемента каталога и помещающей результат работы в переменную catalog, являющуюся выходным параметром функции catalog_by_product().

В функции catalog_by_product() потребовалась временная переменная id. Для использования любой переменной в функции требуется ее объявление 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 33.25).

Листинг 33.25. Использование оператора DECLARE

```
mysql> CREATE PROCEDURE declare_var ()
    -> BEGIN
    ->   DECLARE id, num INT(11) DEFAULT 0;
    ->   DECLARE name, hello, temp TINYTEXT;
    -> END
    -> //
```

В листинге 33.25 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 (см. листинг 33.14).

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 одинаковым именем, и действовать они будут только в рамках данного блока, не пересекаясь с переменными других блоков (листинг 33.26).

Листинг 33.26. Область видимости локальных переменных с одинаковыми именами

```
mysql> 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
    -> //
mysql> CALL declare_var()//
+----------------------+
| var                  |
+----------------------+
| внутренняя переменная |
+----------------------+
+--------------------+
| var                |
+--------------------+
| внешняя переменная |
+--------------------+
```

В листинге 33.26 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 объявлено экранирующей ее переменной (листинг 33.27).

Листинг 33.27. Область видимости локальных переменных, объявленных во внешнем блоке `BEGIN...END`

```
mysql> CREATE PROCEDURE one_declare_var ()
    -> BEGIN
    ->   DECLARE var TINYTEXT DEFAULT 'внешняя переменная';
    ->   BEGIN
    ->     SELECT var;
    ->   END;
    ->   SELECT var;
    -> END
    -> //
mysql> CALL one_declare_var()//
+--------------------+
| var                |
+--------------------+
| внешняя переменная |
+--------------------+
```

```
+--------------------+
| var                |
+--------------------+
| внешняя переменная |
+--------------------+
```

Обратное не верно: переменная, объявленная во вложенном блоке, недоступна во внешнем (листинг 33.28).

Листинг 33.28. Область видимости локальных переменных, объявленных во вложенном блоке `BEGIN...END`

```
mysql> CREATE PROCEDURE inner_declare_var ()
    -> BEGIN
    ->   BEGIN
    ->     DECLARE var TINYTEXT DEFAULT 'внутренняя переменная';
    ->     SELECT var;
    ->   END;
    ->   SELECT var;
    -> END
    -> //
mysql> CALL inner_declare_var()//
+----------------------+
| var                  |
+----------------------+
| внутренняя переменная |
+----------------------+
ERROR 1054 (42S22): Unknown column 'var' in 'field list'
```

Как видно из листинга 33.28, 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 раздела следует отметить, что не допускается и повторное объявление переменной в рамках одного блока `BEGIN...END` (листинг 33.29). Это приводит к возникновению ошибки 1331: "Повторное объявление переменной".

Листинг 33.29. Повторное объявление переменной `var`

```
mysql> 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
```

33.2.4. Хранимые функции

Помимо формы `CREATE PROCEDURE`, создающей хранимую процедуру, допускается использование формы `CREATE FUNCTION`, которая создает *хранимую функцию*. Функция в отличие от процедуры может вызываться непосредственно, без использования оператора `CALL` и возвращать одно значение, которое подставляется в место вызова функции, как в случае встроенных функций MySQL.

Создадим простейшую хранимую функцию `say_hello()`, которая будет принимать единственный входной параметр с именем `name` и возвращать фразу "`Hello, name!`", где вместо подстроки *`name`* будет подставлено значение параметра `name` (листинг 33.30).

Листинг 33.30. Использование оператора `CREATE FUNCTION`

```
mysql> CREATE FUNCTION say_hello (name CHAR(20)) RETURNS CHAR(50)
    -> BEGIN
    ->  RETURN CONCAT('Hello, ', name, '!');
    -> END
    -> //
mysql> SELECT say_hello('world'), say_hello('softtime')//
+-------------------+----------------------+
| say_hello('world') | say_hello('softtime') |
+-------------------+----------------------+
| Hello, world!      | Hello, softtime!      |
+-------------------+----------------------+
```

После объявления параметров функции следует оператор `RETURNS`, который задает тип возвращаемого функцией значения. Вернуть значение из функции можно при помощи оператора `RETURN` (листинг 33.31), который может быть вызван в любой точке функции. Вызов оператора `RETURN` означает, что функция должна немедленно завершить выполнение и вернуть значение, переданное в качестве аргумента оператора `RETURN`.

Замечание

При объявлении параметров функции использование ключевых слов `IN`, `INOUT` и `OUT` недопустимо. Все параметры, передаваемые функции, являются входными.

Замечание

Функция обязательно должна содержать оператор `RETURNS`, устанавливающий тип возвращаемого значения, и хотя бы один оператор `RETURN` в теле функции, который возвращает это значение.

Листинг 33.31. Использование оператора `RETURN`

```
mysql> CREATE FUNCTION func_catalog (id INT)
    -> RETURNS TINYTEXT
    -> BEGIN
    ->   DECLARE catalog TINYTEXT;
    ->   SELECT name INTO catalog FROM catalogs
    ->   WHERE id_catalog = id LIMIT 1;
    ->   RETURN catalog;
    ->   SELECT name INTO catalog FROM catalogs
    ->   WHERE id_catalog = id + 1 LIMIT 1;
    ->   RETURN catalog;
    -> END
    -> //
mysql> SELECT func_catalog(1)//
+-----------------+
| func_catalog(1) |
+-----------------+
| Процессоры      |
+-----------------+
```

Хранимая функция `func_catalog()` принимает единственный параметр `id` — первичный ключ таблицы `catalogs`. Приняв в качестве параметра `id` значение 1, функция возвращает результат ("Процессоры"), достигнув первого оператора `RETURN`. При этом вторая пара операторов `SELECT` и `RETURN` не достигается никогда (иначе возвращалось бы значение "Оперативная память"). Это не значит, что два оператора `RETURN` в теле функции не должны встречаться. Далее будут рассмотрены условные конструкции, позволяющие в зависимости от условий выбирать вариант, который должен возвращаться — в такой ситуации использование множественного выхода из функции при помощи нескольких операторов `RETURN` не избежать.

Если последовательность операторов, которые решено оформить в виде хранимой процедуры, возвращает одно-единственное значение, гораздо удобнее оформить их в виде хранимой функции, т. к. работать с ними в выражениях гораздо удобнее. Создадим две функции `count_product_in_catalog()` и `count_product()`, которые будут возвращать общее число товарных позиций в каталоге и общее число товарных позиций в учебном электронном магазине `shop` (листинг 33.32).

Листинг 33.32. Подсчет процентного вклада товарных позиций каталога

```
mysql> CREATE FUNCTION count_product_in_catalog (id INT)
    -> RETURNS INT
    -> BEGIN
    ->   DECLARE total INT;
    ->   SELECT SUM(count) INTO total FROM products
    ->   WHERE id_catalog = id LIMIT 1;
    ->   RETURN total;
    -> END
    -> //
mysql> CREATE FUNCTION count_product ()
    -> RETURNS INT
    -> BEGIN
    ->   DECLARE total INT;
    ->   SELECT SUM(count) INTO total FROM products;
    ->   RETURN total;
    -> END
    -> //
mysql> SELECT count_product_in_catalog(1) AS total,
    -> count_product_in_catalog(1)/count_product()*100 AS persent//
+-------+---------+
| total | persent |
+-------+---------+
|    56 | 26.5403 |
+-------+---------+
```

Как видно из листинга 33.32, функции удобно использовать в выражениях, например для подсчета процентного вклада товарных позиций каталога в общее число товаров в электронном магазине `shop`.

33.3. Группа характеристик хранимых процедур

Синтаксис хранимых процедур допускает использование следующих характеристик *`characteristic`* в определении `CREATE PROCEDURE` и `CREATE FUNCTION` *(см. разд. 33.2)*:

```
LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | 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 33.33).

Листинг 33.33. Использование ключевого слова `LANGUAGE SQL`

```
mysql> 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 33.34).

Листинг 33.34. Использование ключевого слова `NOT DETERMINISTIC`

```
mysql> 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.

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, создав-

шего ее. При использовании `SQL SECURITY INVOKER` процедура вызывается с привилегиями пользователя, вызывающего процедуру оператором `CALL` (листинг 33.35).

Листинг 33.35. Использование ключевого слова `SQL SECURITY`

```
mysql> 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 33.36) 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 33.36. Использование ключевого слова `COMMENT`

```
mysql> 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
```

33.4. Операторы управления потоком данных

Хранимые процедуры — это не просто удобные контейнеры для группы запросов, они позволяют реализовать достаточно сложную логику, используя операторы ветвления и циклы.

> ***Замечание***
>
> Вне хранимых процедур описываемые в данном разделе операторы применять нельзя.

33.4.1. 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 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()`, описание синтаксиса которой можно найти в *разд. 22.1.2*.

Рассмотрим простейший пример использования оператора `IF`. Хранимая процедура `pricelist()`, представленная в листинге 33.37, выводит список товарных позиций

с каталогом, первичный ключ которых равен `id`. Вторым параметром функции является число `cur`, которое может принимать значение 0, если требуется вывести цены в рублях, и 1, если цены должны быть пересчитаны в доллары с курсом 27.0 рублей.

> ***Замечание***
>
> Для создания логических выражений можно использовать все операторы сравнения (=, >, >=, <>, <, <=), описанные в *разд. 15.1.2*. Кроме того, логические выражения можно комбинировать при помощи операторов `&&` (И) и `||` (ИЛИ).

Листинг 33.37. Использование оператора `IF`

```
mysql> CREATE PROCEDURE pricelist (id INT, cur INT)
    -> LANGUAGE SQL
    -> BEGIN
    ->   IF(cur = 0) THEN
    ->     SELECT name, price FROM products WHERE id_catalog = id;
    ->   END IF;
    ->   IF(cur = 1) THEN
    ->     SELECT name, price/27.0 FROM products WHERE id_catalog = id;
    ->   END IF;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL pricelist(1,1)//
+-----------------------+------------+
| name                  | price/27.0 |
+-----------------------+------------+
| Celeron 1.8           | 59.074074  |
| Celeron 2.0GHz        | 72.925926  |
| Celeron 2.4GHz        | 78.111111  |
| Celeron D 320 2.4GHz  | 72.666667  |
| Celeron D 325 2.53GHz | 101.740741 |
| Celeron D 315 2.26GHz | 69.629630  |
| Intel Pentium 4 3.2GHz| 268.851852 |
| Intel Pentium 4 3.0GHz| 227.666667 |
| Intel Pentium 4 3.0GHz| 210.111111 |
+-----------------------+------------+
```

Оператор IF может быть снабжен дополнительным блоком ELSE, после которого выполняются операторы, если условие оказалось ложным. Функцию pricelist() можно переписать так, как это показано в листинге 33.38.

> ***Замечание***
>
> После ключевого слова ELSE ставить точку с запятой не нужно, т. к. этим самым единый оператор IF разбивается на части — точку с запятой ставят после ключевого слова END IF.

Листинг 33.38. Использование оператора IF...ELSE

```
mysql> CREATE PROCEDURE pricelist (id INT, cur INT)
    -> LANGUAGE SQL
    -> BEGIN
    ->   IF(cur) THEN
    ->     SELECT name, price/28.8 FROM products WHERE id_catalog = id;
    ->   ELSE
    ->     SELECT name, price FROM products WHERE id_catalog = id;
    ->   END IF;
    -> END
    -> //
```

С точки зрения функциональности, хранимые процедуры из листингов 33.37 и 33.38 абсолютно одинаковы, но в последнем случае понадобился лишь один оператор IF, который принимает в качестве логического выражения параметр cur. Если cur равен 1, что является истиной, выполняется первый оператор, выводящий цены в долларах, если параметр cur равен 0, что является ложью, выполняется запрос в блоке ELSE.

Оператор IF позволяет выбрать и большее число альтернатив. Пусть необходимо помимо рублевого и долларового прайс-листа выводить цены в евро по курсу 34.5 рублей за евро. Таким образом, параметр cur может принимать следующие значения:

- 0 — рубли;
- 1 — доллары;
- 2 и выше — евро.

Листинг 33.39. Использование оператора IF...ELSEIF...ELSE

```
mysql> CREATE PROCEDURE pricelist (id INT, cur INT)
    -> LANGUAGE SQL
    -> BEGIN
    ->   IF(cur = 0) THEN
```

```
    ->     SELECT name, price FROM products WHERE id_catalog = id;
    ->   ELSEIF (cur = 1) THEN
    ->     SELECT name, price/28.8 FROM products WHERE id_catalog = id;
    ->   ELSE
    ->     SELECT name, price/34.5 FROM products WHERE id_catalog = id;
    ->   END IF;
    -> END
    -> //
```

Оператор `IF` в листинге 33.39 проверяет, не равен ли параметр `cur` нулю, если это так, срабатывает первый запрос, и процедура выходит из `IF`. Если параметр `cur` не равен нулю, проверка перемещается к блоку `ELSEIF`, где происходит сравнение параметра `cur` с единицей. Равенство `cur` единице приводит к выполнению второго оператора `SELECT`. Если оператор `cur` принимает любое другое значение, отличное от 0 и 1, выполняется третий запрос. Третий запрос будет выполнен, даже если `cur` является отрицательным значением.

Число блоков `ELSEIF` не ограничено — можно использовать любое их количество. Добавим в процедуру `pricelist()` английский фунт стерлингов по курсу 50.4 рубля за фунт — будем использовать значение параметра `cur`, равное 3 и выше, для обозначения этой валюты (листинг 33.40).

Листинг 33.40. Использование оператора `IF...ELSEIF...ELSE`

```
mysql> CREATE PROCEDURE pricelist (id INT, cur INT)
    -> LANGUAGE SQL
    -> BEGIN
    ->   IF(cur = 0) THEN
    ->     SELECT name, price FROM products WHERE id_catalog = id;
    ->   ELSEIF (cur = 1) THEN
    ->     SELECT name, price/28.8 FROM products WHERE id_catalog = id;
    ->   ELSEIF (cur = 2) THEN
    ->     SELECT name, price/34.5 FROM products WHERE id_catalog = id;
    ->   ELSE
    ->     SELECT name, price/50.4 FROM products WHERE id_catalog = id;
    ->   END IF;
    -> END
    -> //
```

Однако злоупотреблять блоками `ELSEIF` не рекомендуется, т. к. большое число блоков `ELSEIF` снижает читабельность кода.

Если в блоках IF, ELSEIF и ELSE используется два или более операторов, для наглядности можно прибегать к составному оператору BEGIN...END. В листинге 33.41 приводится вариант процедуры pricelist() с выводом прайс-листов, содержащих цены в рублях, долларах и евро. Однако для каждого списка будет выводиться общее число товарных позиций и их общая стоимость в выбранной валюте для элементов каталога с первичным ключом id.

Листинг 33.41. Использование оператора IF...ELSEIF...ELSE

```
mysql> CREATE PROCEDURE pricelist (id INT, cur INT)
    -> LANGUAGE SQL
    -> BEGIN
    ->   IF(cur = 0) THEN
    ->     BEGIN
    ->       SELECT name, price FROM products WHERE id_catalog = id;
    ->       SELECT SUM(count), SUM(price*count) FROM products
    ->       WHERE id_catalog = id;
    ->     END;
    ->   ELSEIF (cur = 1) THEN
    ->     BEGIN
    ->       SELECT name, price/28.8 FROM products WHERE id_catalog = id;
    ->       SELECT SUM(count), SUM(price*count)/28.8  FROM products
    ->       WHERE id_catalog = id;
    ->     END;
    ->   ELSE
    ->     BEGIN
    ->       SELECT name, price/34.5 FROM products WHERE id_catalog = id;
    ->       SELECT SUM(count), SUM(price*count)/34.5  FROM products
    ->       WHERE id_catalog = id;
    ->     END;
    ->   END IF;
    -> END
    -> //
```

33.4.2. 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 соответствующий 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 `pricelist()` из листинга 33.40 при помощи оператора `CASE` (листинг 33.42).

Листинг 33.42. Использование оператора `CASE` (первая форма)

```
mysql> CREATE PROCEDURE pricelist (id INT, cur INT)
    -> LANGUAGE SQL
    -> BEGIN
    ->   CASE cur
    ->     WHEN 0 THEN
    ->       SELECT name, price FROM products WHERE id_catalog = id;
    ->     WHEN 1 THEN
    ->       SELECT name, price/28.8 FROM products WHERE id_catalog = id;
    ->     WHEN 2 THEN
    ->       SELECT name, price/34.5 FROM products WHERE id_catalog = id;
    ->     WHEN 3 THEN
    ->       SELECT name, price/50.4 FROM products WHERE id_catalog = id;
    ->     ELSE
    ->       SELECT 'Ошибка в параметре cur';
    ->   END CASE;
    -> END
```

```
    -> //
mysql> CALL pricelist(1, 5)//
+------------------------+
| Ошибка в параметре cur |
+------------------------+
| Ошибка в параметре cur |
+------------------------+
```

В листинге 33.42 ситуация, когда в качестве параметра `cur` 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 33.43 представлена процедура `pricelist()`, реализованная с применением второй формы оператора `CASE`.

Листинг 33.43. Использование оператора `CASE` (вторая форма)

```
mysql> CREATE PROCEDURE pricelist (id INT, cur INT)
    -> LANGUAGE SQL
    -> BEGIN
    ->   CASE
    ->     WHEN cur = 0 THEN
    ->       SELECT name, price FROM products WHERE id_catalog = id;
    ->     WHEN cur = 1 THEN
    ->       SELECT name, price/28.8 FROM products WHERE id_catalog = id;
    ->     WHEN cur = 2 THEN
    ->       SELECT name, price/34.5 FROM products WHERE id_catalog = id;
    ->     WHEN cur = 3 THEN
    ->       SELECT name, price/50.4 FROM products WHERE id_catalog = id;
    ->     ELSE
    ->       SELECT 'Ошибка в параметре cur';
    ->   END CASE;
    -> END
    -> //
```

Если в одном блоке `WHEN` необходимо выполнить несколько запросов, следует использовать блок `BEGIN...END`.

33.4.3. 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 33.44).

Листинг 33.44. Использование оператора `WHILE` (пример 1)

```
mysql> CREATE PROCEDURE NOW5 ()
    -> LANGUAGE SQL
    -> BEGIN
    ->   DECLARE i INT DEFAULT 3;
    ->   WHILE i > 0 DO
    ->     SELECT NOW();
    ->     SET i = i - 1;
    ->   END WHILE;
    -> END
    -> //
mysql> CALL NOW5()//
+---------------------+
| 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 33.45 представлен код хранимой процедуры, которая выводит текущую дату `num` раз, где `num` — параметр, задаваемый пользователем.

Листинг 33.45. Использование оператора `WHILE` (пример 2)

```
mysql> 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
    -> //
mysql> CALL NOWN(2)//
+---------------------+
| NOW()               |
+---------------------+
| 2005-07-18 12:51:27 |
+---------------------+
+---------------------+
| NOW()               |
+---------------------+
| 2005-07-18 12:51:27 |
+---------------------+
```

Как видно из листинга 33.45, 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.

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 C-подобных языках программирования.

Хранимая процедура `NOWN()` 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 33.46 приводится пример хранимой процедуры, где число итераций ограничено двумя.

Листинг 33.46. Досрочный выход из цикла `WHILE`

```
mysql> 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
    -> //
mysql> 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 33.47).

Листинг 33.47. 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 33.48).

Листинг 33.48. 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 33.49).

Листинг 33.49. Использование оператора `ITERATE`

```
mysql> 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
    -> //
mysql> 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 нечетным (1, 3, 5, 7, 9), текущий цикл прекращается при помощи ключевого слова ITERATE, если индекс является четным (0, 2, 4, 6, 8), 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***
>
> Следует 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.

33.4.4. 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 крайней 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 33.50 представлена хранимая процедура binrand(), которая генерирует и выводит случайную бинарную последовательность из 20 символов. Для формирования бинарной последовательности применяется цикл REPEAT (листинг 33.50).

Листинг 33.50. Использование цикла REPEAT

```
mysql> 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
    -> //
mysql> CALL binrand()//
+----------------------+
| bin                  |
+----------------------+
| 10101000111100011111110 |
+----------------------+
```

33.4.5. 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 33.50, можно переписать с использованием цикла `LOOP`, как это сделано в листинге 33.51.

Листинг 33.51. Использование оператора LOOP

```
mysql> 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
    -> //
mysql> CALL binrand()//
+----------------------+
| bin                  |
+----------------------+
| 01000111011110001110 |
+----------------------+
```

33.4.6. Оператор *GOTO*

Оператор `GOTO` позволяет осуществлять безусловный переход и имеет следующий синтаксис:

```
GOTO label;
```

Оператор `GOTO` осуществляет переход к оператору, помеченному меткой *label*. Это может быть как оператор `BEGIN`, так и любой из циклов: `WHILE`, `REPEAT` и `LOOP`. Кроме того, метка может быть не привязана ни к одному из операторов процедуры, а объявлена при помощи оператора `LABEL`, который имеет следующий синтаксис:

```
LABEL label;
```

В общем случае использование операторов `GOTO` и `LABEL` может выглядеть так, как это представлено в листинге 33.52.

Листинг 33.52. Использование операторов `GOTO` и `LABEL`

```
CREATE PROCEDURE binrand ()
BEGIN
...
```

```
LABEL labelname;
...
GOTO labelname;
...
END
```

Оператор `LABEL` не выполняет никаких действий — задача этого оператора просто объявить метку *labelname*. На месте этого оператора может быть помещен любой оператор, допускающий использование в своем составе метки (`BEGIN`, `WHILE`, `REPEAT` и `LOOP`). Достигая оператора `GOTO`, процедура следует по ссылке и переходит обратно к метке *labelname*. Таким образом, в листинге 33.52 реализован бесконечный цикл. Обычно переход по метке `GOTO` используют совместно с оператором `IF` так, как это продемонстрировано в листинге 33.53, где представлена реализация хранимой процедуры `binrand()` с использованием оператора безусловного перехода `GOTO`.

Листинг 33.53. Использование операторов `GOTO` и `IF`

```
mysql> CREATE PROCEDURE binrand ()
    -> LANGUAGE SQL
    -> BEGIN
    ->   DECLARE i INT DEFAULT 0;
    ->   DECLARE bin TINYTEXT DEFAULT '';
    ->   LABEL wet;
    ->   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 GOTO wet;
    ->   END IF;
    ->   SELECT bin;
    -> END
    -> //
mysql> CALL binrand()//
+----------------------+
| bin                  |
+----------------------+
| 00001010001110111100 |
+----------------------+
```

Однако использовать оператор `GOTO` для реализации циклов не рекомендуется, т. к. обычные циклы гораздо нагляднее и проще поддаются модификации, к тому же в них сложнее допустить логическую ошибку.

Часто вообще рекомендуется не применять оператор GOTO, т. к. он подталкивает к созданию плохо структурированного кода. Это не исключает возможности создания читабельного и структурированного кода с использованием оператора GOTO, но соблазн неправильной реализации этого оператора слишком велик. Кроме того, любой код можно переписать без GOTO, некоторые современные языки программирования (например, PHP), вообще не включают его в свой состав.

33.5. Метаданные

Существует четыре способа просмотреть данные, относящиеся к хранимым процедурам или функциям:

- операторы SHOW PROCEDURE STATUS/SHOW FUNCTION STATUS;
- операторы SHOW CREATE PROCEDURE/SHOW CREATE FUNCTION;
- запрос SELECT FROM mysq.proc;
- запрос SELECT FROM *information_schema*.

Первые три рассматриваются в данном разделе, последний способ с привлечением информационной схемы представлен в *главе 36*.

33.5.1. Оператор *SHOW PROCEDURE STATUS*

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 хранимые функции. При использовании ключевого слова LIKE можно вывести информацию только о тех процедурах, имена которых удовлетворяют шаблону *pattern* (листинг 33.54).

Листинг 33.54. Использование оператора SHOW PROCEDURE STATUS

```
mysql> 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 33.54, 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 хранимых процедур (листинг 33.55). При использовании ключевого слова `LIKE` можно вывести информацию только о тех функциях, имена которых удовлетворяют шаблону *pattern*.

Листинг 33.55. Использование оператора `SHOW FUNCTION STATUS`

```
mysql> 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.

33.5.2. Оператор *SHOW CREATE*

Еще одним оператором, позволяющим получить информацию о хранимых процедурах, является оператор `SHOW CREATE PROCEDURE`, который имеет следующий синтаксис:

```
SHOW CREATE PROCEDURE procname;
```

Оператор выводит оператор `CREATE PROCEDURE`, при помощи которого была создана хранимая процедура *procname* (листинг 33.56).

Листинг 33.56. Использование оператора `SHOW CREATE PROCEDURE`

```
mysql> 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
```

Оператор `SHOW CREATE PROCEDURE` выводит информацию только для хранимых процедур, для хранимых функций необходимо воспользоваться оператором `SHOW CREATE FUNCTION`, который имеет следующий синтаксис:

```
SHOW CREATE FUNCTION funcname;
```

Оператор выводит оператор `CREATE FUNCTION`, при помощи которого была создана хранимая процедура *`funcname`*.

33.5.3. Извлечение информации из таблицы *mysql.proc*

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 33.57 приводится пример `SELECT`-запроса, извлекающего запись, соответствующую хранимой функции `count_product_in_catalog()` (см. листинг 33.32).

Листинг 33.57. Извлечение информации из таблицы `mysql.proc`

```
mysql> 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 33.57, 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` — время создания хранимой процедуры;
- `modified` — 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 33.58 формируется список всех хранимых процедур и функций, причем выводится только название и признак — процедура или функция.

Листинг 33.58. Извлечение списка хранимых процедур

```
mysql> 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 |
+--------------------------+-----------+
```

33.6. Удаление хранимых процедур

Для удаления хранимых процедур предназначен оператор `DROP PROCEDURE`, который имеет следующий синтаксис:

```
DROP PROCEDURE [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 в базе данных `test` хранимую процедуру `test()` и применим к ней оператор `DROP PROCEDURE` (листинг 33.59).

Листинг 33.59. Использование оператора `DROP PROCEDURE`

```
mysql> use test
Database changed
mysql> CREATE PROCEDURE test() SELECT VERSION();
mysql> SELECT name, type FROM mysql.proc WHERE db = 'test';
+------+-----------+
| name | type      |
+------+-----------+
| test | PROCEDURE |
+------+-----------+
mysql> DROP PROCEDURE test;
mysql> SELECT name, type FROM mysql.proc WHERE db = 'test';
Empty set (0.00 sec)
```

Однако использование оператора `DROP PROCEDURE` применительно к хранимой функции заканчивается ошибкой (листинг 33.60).

Листинг 33.60. Удаление хранимой функции

```
mysql> CREATE FUNCTION test() RETURNS TINYTEXT RETURN VERSION();
mysql> SELECT name, type FROM mysql.proc WHERE db = 'test';
+------+----------+
| name | type     |
+------+----------+
| test | FUNCTION |
+------+----------+
mysql> DROP PROCEDURE test;
```

```
ERROR 1305 (42000): PROCEDURE test.test does not exist
mysql> SELECT name, type FROM mysql.proc WHERE db = 'test';
+------+----------+
| name | type     |
+------+----------+
| test | FUNCTION |
+------+----------+
```

Для удаления хранимых функций необходимо использовать специальный оператор `DROP FUNCTION` (листинг 33.61).

Листинг 33.61. Использование оператора `DROP FUNCTION`

```
mysql> SELECT name, type FROM mysql.proc WHERE db = 'test';
+------+----------+
| name | type     |
+------+----------+
| test | FUNCTION |
+------+----------+
mysql> DROP FUNCTION test;
mysql> SELECT name, type FROM mysql.proc WHERE db = 'test';
Empty set (0.00 sec)
```

33.7. 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 для хранимой процедуры.

> ***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 33.62.

Листинг 33.62. Использование оператора `ALTER PROCEDURE`

```
mysql> CREATE PROCEDURE test() SELECT VERSION();
mysql> SELECT name, type, security_type, comment FROM mysql.proc
    -> WHERE db = 'test';
+------+-----------+---------------+---------+
| name | type      | security_type | comment |
+------+-----------+---------------+---------+
| test | PROCEDURE | DEFINER       |         |
+------+-----------+---------------+---------+
mysql> ALTER PROCEDURE test
    -> SQL SECURITY INVOKER
    -> COMMENT 'Функция возвращает версию сервера';
mysql> 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 33.62, режим выполнения и комментарий к хранимой процедуре были успешно изменены.

33.8. 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 стандартом в SQL. В СУБД MySQL данный код поддерживается, начиная с версии 4.1. Примером 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 33.63).

Листинг 33.63. Создание таблицы tbl

```
mysql> 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` два одинаковых значения (листинг 33.64).

Листинг 33.64. Использование оператора `DECLARE...HANDLER FOR`

```
mysql> 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
    -> //
mysql> SELECT @error//
+--------+
| @error |
+--------+
| NULL   |
+--------+
mysql> CALL handler_key()//
+-----------------+
| VERSION()       |
+-----------------+
| 5.0.18-standard |
+-----------------+
mysql> SELECT @error//
+--------+
| @error |
+--------+
| Ошибка |
+--------+
```

Как видно из листинга 33.64, 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 33.65).

Листинг 33.65. Использование составного оператора в обработчике

```
mysql> 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
    -> //
mysql> 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 |
+-----------------------------------------------+
mysql> SELECT @error//
+--------+
| @error |
+--------+
| Ошибка |
+--------+
```

Как видно из листинга 33.65, 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 33.66).

Листинг 33.66. Использование обработчика `EXIT`

```
mysql> 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
    -> //
mysql> CALL handler_key()//
+-----------------------------------------------+
| Ошибка при вставке нового значения в таблицу tbl |
+-----------------------------------------------+
| Ошибка при вставке нового значения в таблицу tbl |
+-----------------------------------------------+
mysql> SELECT @error//
+--------+
| @error |
+--------+
| Ошибка |
+--------+
```

Как видно из листинга 33.66, если используется обработчик `EXIT`, то при возникновении первой же ошибки функция прекращает свою работу.

При указании кода ошибки можно использовать не только целочисленные коды, 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. Примером 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 33.67.

Листинг 33.67. 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 33.68 демонстрируется использование именованных условий, при этом следует обратить внимание, что название именованного условия заключается в необязательные обратные кавычки.

Листинг 33.68. Использование именованных условий

```
mysql> 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 33.68 хранимая процедура выполняет транзакцию. При возникновении ошибки дублирования записей состояние базы данных откатывается при помощи оператора ROLLBACK.

> ***Замечание***
>
> Транзакции подробнее описываются в *главе 26*.

33.9. 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 33.69).

Листинг 33.69. Ошибочное выполнение хранимой процедуры

```
mysql> CREATE PROCEDURE idcatalogs (OUT total INT)
    -> BEGIN
    ->   SELECT id_catalog INTO total FROM catalogs;
    -> END
    -> //
mysql> 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 — создания новой таблицы letter_catalogs с именами элементов каталога в верхнем регистре. Создадим таблицу letter_catalogs при помощи запроса, представленного в листинге 33.70.

Листинг 33.70. Создание таблицы letter_catalogs

```
mysql> 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 33.71.

Листинг 33.71. Использование курсоров

```
mysql> 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
    -> //
mysql> CALL curcatalogs()//
mysql> SELECT * FROM letter_catalogs//
+------------+--------------------+
| id_catalog | name               |
```

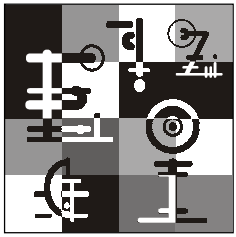
```
+------------+--------------------+
|          1 | ПРОЦЕССОРЫ         |
|          2 | МАТЕРИНСКИЕ ПЛАТЫ  |
|          3 | ВИДЕОАДАПТЕРЫ      |
|          4 | ЖЕСТКИЕ ДИСКИ      |
|          5 | ОПЕРАТИВНАЯ ПАМЯТЬ |
+------------+--------------------+
```

Как видно из листинга 33.71, 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.
```



Глава 34

Триггеры

Триггер — эта та же хранимая процедура, но 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`. Например, при оформлении нового заказа, т. е. при добавлении новой записи в таблицу `orders`, можно создать триггер, который автоматически будет вычитать число заказанных товарных позиций в таблице `products`.

> ***Замечание***
>
> Поддержка триггеров введена в СУБД MySQL, начиная с версии 5.0.2.

В данной главе рассматривается создание, удаление и использование триггеров.

34.1. Оператор *CREATE TRIGGER*

Оператор `CREATE TRIGGER` позволяет создать новый триггер и 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.

Вообще синтаксис и допустимые операторы совпадают с телом хранимых процедур *(см. главу 33)*. 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 типа 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. Подробнее с системой привилегий СУБД MySQL можно познакомиться в *главе 27*.

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 34.1).

Листинг 34.1. Создание простейшего триггера

```
mysql> CREATE TRIGGER sub_count AFTER INSERT ON orders
    -> FOR EACH ROW
    -> BEGIN
    ->   SET @tot = 1;
    -> END
    -> //
mysql> select @tot//
+------+
| @tot |
+------+
| NULL |
+------+
mysql> INSERT INTO orders VALUES (NULL,1,NOW(),1,10)//
mysql> select @tot//
+------+
| @tot |
+------+
| 1    |
+------+
```

Как видно из листинга 34.1, 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 34.2).

Листинг 34.2. Новый вариант триггера `sub_count`

```
mysql> CREATE TRIGGER sub_count AFTER INSERT ON orders
    -> FOR EACH ROW
    -> BEGIN
    ->   SET @tot = @tot + NEW.number;
    -> END
    -> //
mysql> select @tot//
+------+
| @tot |
+------+
```

```
| 1    |
+------+
mysql> INSERT INTO orders VALUES (NULL,1,NOW(),5,10)//
mysql> select @tot//
+------+
| @tot |
+------+
| 6    |
+------+
```

Как видно из листинга 34.2, 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 34.3).

Листинг 34.3. Использование ключевого слова `BEFORE`

```
mysql> CREATE TRIGGER restrict_count BEFORE INSERT ON orders
    -> FOR EACH ROW
    -> BEGIN
    ->   SET NEW.number = 1;
    -> END
    -> //
mysql> INSERT INTO orders VALUES (NULL,1,NOW(),2,10)//
mysql> 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 34.3, 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 34.4.

Листинг 34.4. Контролирующий триггер

```
mysql> 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')//
mysql> SELECT surname, patronymic, name FROM users
    -> WHERE id_user = LAST_INSERT_ID()//
+---------+------------+------+
| surname | patronymic | name |
+---------+------------+------+
| Ремизов | A          | C    |
+---------+------------+------+
```

Как видно из листинга 34.4, 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`.

34.2. Оператор *DROP TRIGGER*

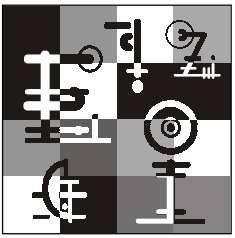
Оператор `DROP TRIGGER` позволяет удалять существующие триггеры и имеет следующий синтаксис:

```
DROP TRIGGER tbl_name.trigger_name
```

Оператор удаляет триггер с именем *trigger_name* таблицы *tbl_name*. В листинге 34.5 демонстрируется удаление триггера `restrict_user` таблицы `users`, созданного в листинге 34.4.

Листинг 34.5. Удаление триггера `restrict_user`

```
mysql> DROP TRIGGER users.restrict_user;
```



Глава 35

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.

> ***Замечание***
>
> Представления и все связанные с ними операторы доступны в СУБД MySQL, начиная с версии 5.0.1.

Когда СУБД MySQL встречает в запросе ссылку на представление, она отыскивает его определение, сохраненное в базе данных. Затем происходит преобразование пользовательского запроса с участием представления в эквивалентный запрос к исходным таблицам. После чего осуществляется выполнение запроса. Таким образом, клиент может работать с представлениями так, будто бы они независимые таблицы. Если определение представления простое, то каждая строка представления формируется "на лету". Если определение сложное, СУБД MySQL "материализует" представление в виде временной таблицы в оперативной памяти или на жестком диске.

Клиент, обращаясь к представлению, будет видеть только столбцы результирующей таблицы. При этом не имеет значения, сколько столбцов в исходной таблице и является ли `SELECT`-запрос, лежащий в основе представления, одно- или многотабличным. Кроме того, клиенту можно запретить обращаться к исходным таблицам, но снабдить привилегиями обращения к представлениям. Таким образом, на одном и том же наборе таблиц можно создать гибкие системы доступа к таблицам базы данных.

Учитывая все вышесказанное, можно выделить следующие преимущества представлений:

- безопасность — каждому пользователю можно разрешить доступ к небольшому числу представлений, содержащих только ту информацию, которую ему позволено знать;

- простота запросов — с помощью представления можно извлечь данные из нескольких таблиц и представить их как одну таблицу, заменяя запрос ко многим таблицам в однотабличный запрос к представлению;
- простота структуры — представления позволяют создать для каждого пользователя собственную "структуру" базы данных, отображая только те данные, которые ему нужны, и "не засоряя" результаты вспомогательными столбцами, которые пользователю заведомо не пригодятся;
- защита от изменений — в связи с оптимизацией скорости, таблицы и их структура могут постоянно претерпевать изменения и переименовываться. Представления позволяют создавать виртуальные таблицы со старыми именами и структурой, помогая избежать модификации внешней прикладной программы.

Помимо преимуществ, описанных выше, представления обладают рядом недостатков:

- производительность — представления создают лишь видимость существования соответствующей таблицы, и СУБД MySQL приходится преобразовывать запрос к представлению в запрос к исходным таблицам. Если представление отображает многотабличный запрос, то простой запрос к представлению становится сложным объединением и на его выполнение может потребоваться много времени;
- ограничение на обновление — когда пользователь пытается обновить строки представления, СУБД MySQL необходимо обновить строки в исходных таблицах. Это возможно только для простых представлений; сложные представления обновить нельзя, они доступны только для выборки.

Указанные недостатки означают, что не стоит без разбора применять представления вместо исходных таблиц. В каждом конкретном случае необходимо учитывать перечисленные преимущества и недостатки.

35.1. 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*.

В листинге 35.1 приводится пример создания представления `cat`, которое дублирует таблицу `catalogs` учебной базы данных `shop`.

Листинг 35.1. Создание представления `cat`

```
mysql> CREATE VIEW cat AS SELECT * FROM catalogs;
mysql> SELECT * FROM cat;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
```

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 35.2).

> ***Замечание***
>
> В зависимости от используемой версии СУБД MySQL структура файла, в котором хранится определение представления, может отличаться от приведенного в листинге 35.2.

Листинг 35.2. Структура файла cat.frm представления `cat`

```
TYPE=VIEW
query=select `shop`.`catalogs`.`id_catalog` AS
`id_catalog`,`shop`.`catalogs`.`name` AS `name` from `shop`.`catalogs`
md5=d0f2f9ade597ab03f3c876837262b4d6
updatable=1
algorithm=0
with_check_option=0
revision=1
timestamp=2005-07-14 14:51:37
create-version=1
source=CREATE VIEW cat AS SELECT * FROM catalogs
```

Представление рассматривается СУБД MySQL как полноценная таблица и может быть просмотрено в списке таблиц базы данных при помощи оператора `SHOW TABLES` (листинг 35.3).

Листинг 35.3. Просмотр представления при помощи оператора SHOW TABLES

```
mysql> SHOW TABLES;
+---------------+
| Tables_in_test |
+---------------+
| cat            |
| catalogs       |
| forums         |
| goods          |
| products       |
+---------------+
```

В листинге 35.1 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 35.4).

Листинг 35.4. Изменения названия столбцов и порядка их следования

```
mysql> CREATE VIEW cat (catalog, id)
    -> AS SELECT name, id_catalog FROM catalogs;
mysql> SELECT * FROM cat;
+--------------------+----+
| catalog            | id |
+--------------------+----+
| Процессоры         |  1 |
| Материнские платы  |  2 |
| Видеоадаптеры      |  3 |
| Жесткие диски      |  4 |
| Оперативная память |  5 |
+--------------------+----+
```

Как видно из листинга 35.4, 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. Пусть для таблицы `products` следует создать представление `goods`, содержащее имя

товарной позиции, а также ее цену в рублях, долларах и евро. Пусть курс доллара будет составлять 27.0 рублей, а евро 34.6 рублей, тогда запрос на создание представления может выглядеть так, как это представлено в листинге 35.5.

Листинг 35.5. Создание представления с вычисляемыми столбцами

```
mysql> CREATE OR REPLACE VIEW goods (name, rubl, doll, euro)
    -> AS SELECT name, price, price/27.0, price/34.6 FROM products;
mysql> SELECT * FROM goods;
+------------------------------+---------+-----------+-----------+
| name                         | rubl    | doll      | euro      |
+------------------------------+---------+-----------+-----------+
| Celeron 1.8                  | 1595.00 | 59.074074 | 46.098266 |
| Celeron 2.0GHz               | 1969.00 | 72.925926 | 56.907514 |
| Celeron 2.4GHz               | 2109.00 | 78.111111 | 60.953757 |
| Celeron D 320 2.4GHz         | 1962.00 | 72.666667 | 56.705202 |
| Celeron D 325 2.53GHz        | 2747.00 | 101.740741| 79.393064 |
| Celeron D 315 2.26GHz        | 1880.00 | 69.629630 | 54.335260 |
| Intel Pentium 4 3.2GHz       | 7259.00 | 268.851852| 209.797688|
| Intel Pentium 4 3.0GHz       | 6147.00 | 227.666667| 177.658960|
| Intel Pentium 4 3.0GHz       | 5673.00 | 210.111111| 163.959538|
| Gigabyte GA-8I848P-RS        | 1896.00 | 70.222222 | 54.797688 |
| Gigabyte GA-8IG1000          | 2420.00 | 89.629630 | 69.942197 |
| Gigabyte GA-8IPE1000G        | 2289.00 | 84.777778 | 66.156069 |
| Asustek P4C800-E Delux       | 5395.00 | 199.814815| 155.924855|
| Asustek P4P800-VM\L i865G    | 2518.00 | 93.259259 | 72.774566 |
| Epox EP-4PDA3I               | 2289.00 | 84.777778 | 66.156069 |
| ASUSTEK A9600XT/TD           | 5156.00 | 190.962963| 149.017341|
| ASUSTEK V9520X               | 1602.00 | 59.333333 | 46.300578 |
| SAPPHIRE 256MB RADEON 9550   | 2730.00 | 101.111111| 78.901734 |
| GIGABYTE AGP GV-N59X128D     | 5886.00 | 218.000000| 170.115607|
| Maxtor 6Y120P0               | 2456.00 | 90.962963 | 70.982659 |
| Maxtor 6B200P0               | 3589.00 | 132.925926| 103.728324|
| Samsung SP0812C              | 2093.00 | 77.518519 | 60.491329 |
| Seagate Barracuda ST3160023A | 3139.00 | 116.259259| 90.722543 |
| Seagate ST3120026A           | 2468.00 | 91.407407 | 71.329480 |
| DDR-400 256MB Kingston       | 1085.00 | 40.185185 | 31.358382 |
| DDR-400 256MB Hynix Original | 1179.00 | 43.666667 | 34.075145 |
```

```
| DDR-400 256MB PQI              | 899.00  | 33.296296  | 25.982659  |
| DDR-400 512MB Kingston         | 1932.00 | 71.555556  | 55.838150  |
| DDR-400 512MB PQI              | 1690.00 | 62.592593  | 48.843931  |
| DDR-400 512MB Hynix            | 1717.00 | 63.592593  | 49.624277  |
+--------------------------------+---------+------------+------------+
```

Следует отметить, что представление `goods`, кроме того, не содержит дополнительных столбцов, таких как описание каждой товарной позиции, внешний ключ для связи с таблицей `catalogs` и пр., способных затруднить работу с прайс-листом. Такие представления называются *вертикальными представлениями*. Вертикальные представления широко применяются для ограничения доступа пользователей к столбцам таблицы. Например, можно создать представление `list_user`, которое будет отображать фамилию и инициалы пользователя, скрывая поля с контактными телефонами и адресами электронной почты (листинг 35.6).

Листинг 35.6. Создание представления списка пользователей `list_user`

```
mysql> 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;
mysql> 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 покупателей, 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 35.7 выводится список пользователей, отсортированный в обратном порядке, который получен с помощью представления `list_user`.

Листинг 35.7. Список пользователей, отсортированный в обратном порядке

```
mysql> 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.

Наряду с вертикальными представлениями используются *горизонтальные представления*, которые ограничивают доступ пользователей к строкам таблиц, делая видимыми только те строки, с которыми они работают.

Допустим, в учебном электронном магазине `shop`, содержащем пять элементов каталога, работают три менеджера, ответственность которых распределяется по этим элементам каталога следующим образом:

- видеоадаптеры — первый менеджер;
- жесткие диски — второй менеджер;
- материнские платы — первый менеджер;
- оперативная память — третий менеджер;
- процессоры — третий менеджер.

Для того чтобы при работе с таблицей `products` менеджер видел только те товарные позиции, за которые он отвечает, можно создать три представления: `first`, `second` и `third` для первого, второго и третьего менеджеров соответственно (листинг 35.8). Затем учетные записи менеджеров следует лишить привилегии доступа к таблице `products` и разрешить просматривать только свои представления. Таким образом, с одной стороны, менеджерам не потребуется искать свои товарные позиции среди большого числа других товарных позиций, не имеющих к ним отношения, с другой стороны, они не смогут случайно внести изменения в товарные позиции других менеджеров.

Листинг 35.8. Создание представлений `first`, `second` и `third`

```
mysql> CREATE VIEW first
    -> AS SELECT * FROM products
    ->    WHERE id_catalog IN (SELECT id_catalog
```

```
    ->                          FROM catalogs
    ->                          WHERE name = 'Видеоадаптеры' OR
    ->                                name = 'Материнские платы')
    ->    ORDER BY name;
mysql> SELECT name, price, count FROM first;
+---------------------------+---------+-------+
| name                      | price   | count |
+---------------------------+---------+-------+
| ASUSTEK A9600XT/TD        | 5156.00 |     2 |
| Asustek P4C800-E Delux    | 5395.00 |     4 |
| Asustek P4P800-VM\L i865G | 2518.00 |     6 |
| ASUSTEK V9520X            | 1602.00 |     6 |
| Epox EP-4PDA3I            | 2289.00 |     5 |
| GIGABYTE AGP GV-N59X128D  | 5886.00 |     6 |
| Gigabyte GA-8I848P-RS     | 1896.00 |     4 |
| Gigabyte GA-8IG1000       | 2420.00 |     2 |
| Gigabyte GA-8IPE1000G     | 2289.00 |     6 |
| SAPPHIRE 256MB RADEON 9550 | 2730.00 |     3 |
+---------------------------+---------+-------+
mysql> CREATE VIEW second
    -> AS SELECT * FROM products
    ->    WHERE id_catalog IN (SELECT id_catalog
    ->                         FROM catalogs
    ->                         WHERE name = 'Жесткие диски')
    ->    ORDER BY name;
mysql> SELECT name, price, count FROM second;
+-----------------------------+---------+-------+
| name                        | price   | count |
+-----------------------------+---------+-------+
| Maxtor 6B200P0              | 3589.00 |     4 |
| Maxtor 6Y120P0              | 2456.00 |     6 |
| Samsung SP0812C             | 2093.00 |     5 |
| Seagate Barracuda ST3160023A | 3139.00 |     3 |
| Seagate ST3120026A          | 2468.00 |     8 |
+-----------------------------+---------+-------+
mysql> CREATE VIEW third
    -> AS SELECT * FROM products
    ->    WHERE id_catalog IN (SELECT id_catalog
    ->                         FROM catalogs
```

```
    ->                          WHERE name = 'Процессоры' OR
    ->                                name = 'Оперативная память')
    ->    ORDER BY name;
mysql> SELECT name, price, count FROM third;
+------------------------------+---------+-------+
| name                         | price   | count |
+------------------------------+---------+-------+
| Celeron 1.8                  | 1595.00 |    10 |
| Celeron 2.0GHz               | 1969.00 |     2 |
| Celeron 2.4GHz               | 2109.00 |     4 |
| Celeron D 315 2.26GHz        | 1880.00 |     6 |
| Celeron D 320 2.4GHz         | 1962.00 |     0 |
| Celeron D 325 2.53GHz        | 2747.00 |     6 |
| DDR-400 256MB Hynix Original | 1179.00 |    15 |
| DDR-400 256MB Kingston       | 1085.00 |    20 |
| DDR-400 256MB PQI            | 899.00  |    10 |
| DDR-400 512MB Hynix          | 1717.00 |     8 |
| DDR-400 512MB Kingston       | 1932.00 |    20 |
| DDR-400 512MB PQI            | 1690.00 |    12 |
| Intel Pentium 4 3.0GHz       | 6147.00 |     1 |
| Intel Pentium 4 3.0GHz       | 5673.00 |    12 |
| Intel Pentium 4 3.2GHz       | 7259.00 |     5 |
+------------------------------+---------+-------+
```

Использование вложенных запросов при формировании представлений не совсем рационально, т. к. на представления `first`, `second` и `third` могут накладываться внешние условия. Это приведет к тому, что скорость выполнения запроса будет невелика. При формировании запросов лучше поступиться их читабельностью и вычислить заранее первичные ключи элементов каталога, возвращаемые вложенными запросами (листинг 35.9). Это позволит не выполнять многократно дополнительный вложенный запрос для каждой строки.

Листинг 35.9. Представления, оптимизированные по скорости

```
mysql> CREATE VIEW first
    -> AS SELECT * FROM products
    ->    WHERE id_catalog IN (2, 3)
    ->    ORDER BY name;
mysql> CREATE VIEW second
    -> AS SELECT * FROM products
```

```
    ->    WHERE id_catalog IN (4)
    ->    ORDER BY name;
mysql> CREATE VIEW third
    -> AS SELECT * FROM products
    ->    WHERE id_catalog IN (1, 5)
    ->    ORDER BY name;
```

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.

Предложение ALGORITHM (листинг 35.10) 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 35.10. Использование ключевого слова ALGORITHM

```
mysql> CREATE ALGORITHM = TEMPTABLE VIEW cat AS SELECT * FROM catalogs;
```

Запрос в листинге 35.10 требует от MySQL при каждом обращении к представлению cat создавать временную таблицу. При создании представления с помощью запроса, приведенного в листинге 35.11, временная таблица для результирующей таблицы SELECT-запроса не будет создаваться никогда.

Листинг 35.11. Использование ключевого слова MERGE

```
mysql> CREATE ALGORITHM = MERGE VIEW cat AS SELECT * FROM catalogs;
```

Ключевые слова DEFINER и SQL SECURITY были добавлены, начиная с версии MySQL 5.0.13, но реально заработали, лишь начиная с версии 5.0.16.

Ключевое слово `SQL SECURITY`, так же как и в хранимых процедурах *(см. разд. 33.3)*, 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` *(см. главу 27)*.

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`.

Наиболее удобно использовать представления для формирования сгруппированных таблиц. При работе с такими таблицами СУБД MySQL самостоятельно формирует временную таблицу. Создадим представление `price` с общей стоимостью товарных позиций в каждом из элементов каталога (листинг 35.12).

Листинг 35.12. Создание представления `price`

```
mysql> CREATE VIEW price
    -> AS SELECT id_catalog, SUM(price*count) AS price
    ->    FROM products
    ->    GROUP BY id_catalog
    ->    ORDER BY price;
mysql> SELECT * FROM price;
+------------+-----------+
| id_catalog | price     |
+------------+-----------+
|          3 | 63430.00  |
|          4 | 68718.00  |
|          2 | 74291.00  |
|          5 | 121031.00 |
|          1 | 182554.00 |
+------------+-----------+
mysql> SELECT catalogs.name, price.price
    -> FROM price, catalogs
    -> WHERE price.id_catalog = catalogs.id_catalog
```

```
    -> ORDER BY catalogs.name;
+--------------------+-----------+
| name               | price     |
+--------------------+-----------+
| Видеоадаптеры      | 63430.00  |
| Жесткие диски      | 68718.00  |
| Материнские платы  | 74291.00  |
| Оперативная память | 121031.00 |
| Процессоры         | 182554.00 |
+--------------------+-----------+
mysql> SELECT MIN(price), MAX(price), SUM(price) FROM price;
+------------+------------+------------+
| MIN(price) | MAX(price) | SUM(price) |
+------------+------------+------------+
| 63430.00   | 182554.00  | 510024.00  |
+------------+------------+------------+
```

К сгруппированному представлению `price` нельзя применить операторы `UPDATE` и `DELETE`, т. к. суммарная стоимость товара для каждого из элементов каталога хранится во временной таблице, а сами операции обновления и удаления суммарной стоимости, учитывая структуру базы данных `shop`, не имеют смысла. Использование операторов `UPDATE` и `DELETE` совместно с представлениями не допускается, если в `SELECT`-запрос, лежащий в основе представления, входят следующие предложения:

- агрегатные функции (`SUM()`, `MIN()`, `MAX()`, `COUNT()` и др.);
- `DISTINCT`;
- `GROUP BY`;
- `HAVING`;
- `UNION` или `UNION ALL`;
- вложенные запросы в списке `SELECT`;
- перекрестное, левое или правое объединение;
- представления в предложении `FROM`, к которым нельзя применять операторы `UPDATE` и `DELETE`;
- вложенные запросы в предложении `WHERE`, которые ссылаются на таблицы из предложения `FROM`;
- простые литеральные значения, такие как цифра 5 или строка `'Hello'` в списке столбцов на выборку;
- `ALGORITHM = TEMPTABLE` (при явном использовании временных таблиц).

Необязательное ключевое слово `OR REPLACE` позволяет заменить представление, если представление с таким именем уже существует. Если это ключевое слово не используется, попытка создания представления поверх существующего приводит к ошибке 1050: "Таблица 'cat' уже существует" (листинг 35.13).

Листинг 35.13. Использование ключевого слова `OR REPLACE`

```
mysql> CREATE VIEW cat AS SELECT * FROM catalogs;
ERROR 1050: Table 'cat' already exists
mysql> CREATE OR REPLACE VIEW cat AS SELECT * FROM catalogs;
Query OK, 0 rows affected (0.02 sec)
```

> ***Замечание***
>
> Применение оператора `CREATE VIEW` требует привилегии `CREATE VIEW`, а также привилегий `SELECT` для каждого столбца, используемого в представлении. При указании ключевого слова `OR REPLACE` также требуется привилегия `DELETE` для представления.

При создании представления с помощью оператора `CREATE VIEW` представление сохраняется в текущей базе данных. Однако представление можно создать и в произвольной базе данных, указав имя базы данных при помощи расширенного имени таблицы *`dbase.tbl`* (листинг 35.14).

Листинг 35.14. Создание представления в базе данных `test`

```
mysql> CREATE VIEW test.cat AS SELECT * FROM catalogs;
mysql> USE test;
Database changed
mysql> SELECT * FROM cat;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
```

Теперь получить доступ к таблице `catalogs` можно и в базе данных `test`, не прибегая к расширенному имени — достаточно обратиться к представлению `cat`. Таблицы и представления разделяют пространство имен базы данных, поэтому имена пред-

ставлений и таблиц не могут совпадать и должны быть уникальными в пределах одной базы данных.

В качестве SELECT-запроса к представлению может выступать не только запрос к таблице, но и запрос к системной функции MySQL (листинг 35.15).

Листинг 35.15. Представление функции NOW()

```
mysql> CREATE VIEW time AS SELECT NOW() AS time;
mysql> SELECT * FROM time;
+---------------------+
| time                |
+---------------------+
| 2005-07-14 18:15:44 |
+---------------------+
```

Запрос к представлению time будет выдавать текущее время, а не время создания представления. Несмотря на то, что для создания представлений в качестве SELECT-запроса могут выступать практически любые запросы, имеется несколько ограничений:

- SELECT-запрос не может содержать подзапрос в предложении FROM;
- SELECT-запрос не может ссылаться на системную или пользовательскую переменную;
- любые таблицы или представления, на которые в свою очередь ссылается представление, должны существовать физически;
- внутри хранимых процедур представление не может ссылаться на параметры процедуры или локальные переменные процедуры;
- представления не могут ссылаться на временные таблицы и сами объявляться временными при помощи ключевого слова TEMPORARY;
- триггер нельзя ассоциировать с представлением.

Предложение WITH CHECK OPTION добавляется для представлений, к которым могут быть применены операторы INSERT и UPDATE. В этом случае 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 встав-

ляемых 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 35.16).

Листинг 35.16. Использование предложения WITH CHECK OPTION

```
mysql> CREATE TABLE tbl (a INT);
mysql> CREATE VIEW v1 AS SELECT * FROM tbl WHERE a < 2
    -> WITH CHECK OPTION;
mysql> CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0
    -> WITH LOCAL CHECK OPTION;
mysql> CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0
    -> WITH CASCADED CHECK OPTION;
```

Кроме того, в листинге 35.16 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 35.17).

Листинг 35.17. Вставка новых значений в представления v1, v2 и v3

```
mysql> INSERT INTO v1 VALUES (-1);
Query OK, 1 row affected (0.03 sec)
mysql> INSERT INTO v1 VALUES (2);
ERROR 1369: CHECK OPTION failed 'test.v1'
mysql> INSERT INTO v2 VALUES (-1);
ERROR 1369: CHECK OPTION failed 'test.v2'
mysql> INSERT INTO v2 VALUES (2);
Query OK, 1 row affected (0.05 sec)
mysql> INSERT INTO v3 VALUES (-1);
ERROR 1369: CHECK OPTION failed 'test.v3'
mysql> INSERT INTO v3 VALUES (2);
ERROR 1369: CHECK OPTION failed 'test.v3'
```

35.2. 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 35.18).

Листинг 35.18. Использование оператора `DROP VIEW`

```
mysql> DROP VIEW cat, tbl1, tbl2;
mysql> 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 35.19). Это особенно удобно при пакетном выполнении группы операторов, когда эта незначительная ошибка может привести к остановке выполнения всех оставшихся операторов в пакетном файле.

Листинг 35.19. Использование ключевого слова `IF EXISTS`

```
mysql> DROP VIEW cat;
ERROR 1051: Unknown table 'test.cat'
mysql> 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 35.20).

Листинг 35.20. Возвращение статуса удаления представления

```
mysql> SHOW WARNINGS;
+-------+------+-------------------------+
| Level | Code | Message                 |
+-------+------+-------------------------+
| Note  | 1051 | Unknown table 'test.cat' |
+-------+------+-------------------------+
```

35.3. Редактирование представлений

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 35.21 оба запроса являются эквивалентными.

Листинг 35.21. Использование оператора `ALTER VIEW`

```
mysql> ALTER VIEW cat AS SELECT * FROM catalogs;
mysql> CREATE OR REPLACE VIEW cat AS SELECT * FROM catalogs;
```

> ***Замечание***
>
> Использование оператора `ALTER VIEW` требует привилегии `CREATE VIEW` и `DELETE`, а также привилегий `SELECT` для каждого столбца, используемого в представлении.

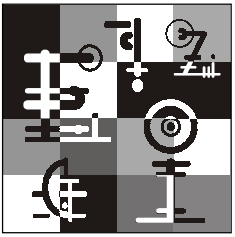
35.4. Оператор *SHOW CREATE VIEW*

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 рас-

ширенные имена (листинг 35.22). Оператор возвращает содержимое файла с расширением frm, соответствующего представлению (листинг 35.22).

Листинг 35.22. Использование оператора SHOW CREATE VIEW

```
mysql> CREATE OR REPLACE VIEW cat AS SELECT * FROM catalogs;
mysql> 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 36

Информационная схема

В *главе 35* были рассмотрены представления, позволяющие создать единый интерфейс для приложений, использующих базы данных. Реальная структура базы данных может изменяться, но при помощи представлений можно воссоздавать интерфейс к базе данных, который ожидает внешнее приложение.

С другой стороны, каждая СУБД имеет собственную системную базу данных, не совпадающую по структуре с системными базами других СУБД. В них хранится самая разнообразная информация, начиная привилегиями и заканчивая хранимыми процедурами. Недостатком системной базы данных является ее непереносимость — каждая СУБД организует системную базу данных по-своему.

> ***Замечание***
>
> В СУБД MySQL системная база данных называется `mysql`.

Устранить недостаток несовместимости призван стандартный набор представлений системной таблицы, который называют *информационной схемой*. При использовании запросов к информационной схеме можно не опасаться, что они будут несовместимыми с другой базой данных.

> ***Замечание***
>
> Представления информационной схемы поддерживаются СУБД MySQL, начиная с версии 5.0.2. В более ранних версиях для просмотра информации о СУБД и базах данных необходимо использовать оператор `SHOW`, который подробно рассматривается в *главе 30*. В то время как оператор `SHOW` поддерживается только СУБД MySQL, информационная схема является стандартной для всех реляционных СУБД.

Информационная схема оформлена в виде специального каталога `INFORMATION_SCHEMA`. В отличие от системного каталога, в СУБД MySQL отсутствует физический каталог, а также файлы для базы данных `INFORMATION_SCHEMA` и ее таблиц — база данных полностью располагается в оперативной памяти. 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`.

Кроме того, база данных информационной схемы INFORMATION_SCHEMA присутствует в результирующей таблице оператора SHOW DATABASES (листинг 36.1).

Замечание

Доступ к базе данных INFORMATION_SCHEMA не требует специальных привилегий. Каждый пользователь снабжается привилегией SELECT на каждую таблицу базы данных INFORMATION_SCHEMA.

Листинг 36.1. База данных INFORMATION_SCHEMA

```
mysql> SHOW DATABASES//
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| qwert              |
| shop               |
| test               |
| wet                |
+--------------------+
```

36.1. Представление *CHARACTER_SETS*

Представление CHARACTER_SETS информационной схемы содержит список и характеристики кодировок, доступных текущему пользователю. Информация в представлении дублирует информацию, предоставляемую оператором SHOW CHARACTER SET *(см. разд. 30.1)*. Соответствие столбцов представления CHARACTER_SETS и результирующей таблицы, возвращаемой оператором SHOW CHARACTER SET, дано в табл. 36.1. Список столбцов представления CHARACTER_SETS приведен в листинге 36.2.

Таблица 36.1. *Представление CHARACTER_SETS*

Имя столбца	SHOW-эквивалент	Замечание
CHARACTER_SET_NAME	Charset	
DEFAULT_COLLATE_NAME	Default collation	
DESCRIPION	Description	Расширение MySQL
MAXLEN	Maxlen	Расширение MySQL

Замечание

Столбцы, помеченные как "Расширение MySQL", не являются стандартными и свойственны только для СУБД MySQL.

Листинг 36.2. Список столбцов представления CHARACTER_SETS

```
mysql> DESCRIBE INFORMATION_SCHEMA.CHARACTER_SETS;
+----------------------+-------------+------+-----+---------+-------+
| Field                | Type        | Null | Key | Default | Extra |
+----------------------+-------------+------+-----+---------+-------+
| CHARACTER_SET_NAME   | varchar(64) | NO   |     |         |       |
| DEFAULT_COLLATE_NAME | varchar(64) | NO   |     |         |       |
| DESCRIPTION          | varchar(60) | NO   |     |         |       |
| MAXLEN               | bigint(3)   | NO   |     | 0       |       |
+----------------------+-------------+------+-----+---------+-------+
```

Столбцы представления CHARACTER_SETS имеют следующие значения:

- CHARACTER_SET_NAME — название кодировки, которую текущий пользователь может использовать, например, в операторе SET CHARACTER SET;
- DEFAULT_COLLATE_NAME — название сортировки, которая назначена по умолчанию для кодировки CHARACTER_SET_NAME;
- DESCRIPTION — описание кодировки;
- MAXLEN — максимальное число байтов, отводимых под один символ.

При извлечении информации из представления CHARACTER_SETS можно использовать все конструкции оператора SELECT (листинг 36.3).

Листинг 36.3. Извлечение кодировок по шаблону

```
mysql> SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
    -> WHERE CHARACTER_SET_NAME LIKE 'cp12%'\G;
*************************** 1. row ***************************
  CHARACTER_SET_NAME: cp1250
DEFAULT_COLLATE_NAME: cp1250_general_ci
         DESCRIPTION: Windows Central European
              MAXLEN: 1
*************************** 2. row ***************************
  CHARACTER_SET_NAME: cp1251
DEFAULT_COLLATE_NAME: cp1251_general_ci
         DESCRIPTION: Windows Cyrillic
              MAXLEN: 1
*************************** 3. row ***************************
  CHARACTER_SET_NAME: cp1256
```

```
DEFAULT_COLLATE_NAME: cp1256_general_ci
         DESCRIPTION: Windows Arabic
              MAXLEN: 1
*************************** 4. row ***************************
  CHARACTER_SET_NAME: cp1257
DEFAULT_COLLATE_NAME: cp1257_general_ci
         DESCRIPTION: Windows Baltic
              MAXLEN: 1
```

Замечание

Очень широкие таблицы, содержащие большое число столбцов, удобнее выводить в вертикальном представлении — добиться этого можно за счет передачи клиенту `mysql` параметра `--vertical` (в кратком представлении `-E`) или добавлением в конце запроса последовательности `\G`. В вертикальном представлении каждая строка начинается с серии звездочек, посередине которых указан номер строки, после чего друг под другом указываются имена столбцов результирующей таблицы и соответствующие им значения. Подробнее консольный клиент `mysql` и его параметры рассматриваются в *главе 3*.

Запрос, представленный в листинге 36.3, можно представить и при помощи оператора `SHOW CHARACTER SET` (листинг 36.4).

Листинг 36.4. Альтернативная выборка при помощи `SHOW CHARACTER SET`

```
mysql> SHOW CHARACTER SET LIKE 'cp12%'\G;
```

Однако при помощи оператора `SHOW CHARACTER SET` невозможно выбрать лишь часть столбцов (листинг 36.5).

Листинг 36.5. Ограничение по горизонтали

```
mysql> SELECT CHARACTER_SET_NAME, DESCRIPTION
    -> FROM INFORMATION_SCHEMA.CHARACTER_SETS
    -> WHERE CHARACTER_SET_NAME LIKE 'cp12%';
+--------------------+--------------------------+
| CHARACTER_SET_NAME | DESCRIPTION              |
+--------------------+--------------------------+
| cp1250             | Windows Central European |
| cp1251             | Windows Cyrillic         |
| cp1256             | Windows Arabic           |
| cp1257             | Windows Baltic           |
+--------------------+--------------------------+
```

36.2. Представление *COLLATIONS*

Представление COLLATIONS информационной схемы содержит список и характеристики сортировок, доступных текущему пользователю. Информация в представлении дублирует информацию, предоставляемую оператором SHOW COLLATION *(см. разд. 30.2)*. Соответствие столбцов представления COLLATIONS и результирующей таблицы, возвращаемой оператором SHOW COLLATION, показано в табл. 36.2.

> ***Замечание***
>
> Столбцы, помеченные как "Расширение MySQL", не являются стандартными и свойственны только для СУБД MySQL.

Таблица 36.2. *Представление* COLLATIONS

Имя столбца	SHOW-эквивалент	Замечание
COLLATION_NAME	Collation	
CHARACTER_SET_NAME	Charset	Расширение MySQL
ID	Id	Расширение MySQL
IS_DEFAULT	Default	Расширение MySQL
IS_COMPILED	Compiled	Расширение MySQL
SORTLEN	Sortlen	Расширение MySQL

В листинге 36.6 приведена результирующая таблица оператора DESCRIBE, примененного по отношению к представлению COLLATIONS.

Листинг 36.6. Список столбцов представления COLLATIONS

```
mysql> DESCRIBE INFORMATION_SCHEMA.COLLATIONS;
+--------------------+-------------+------+-----+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+-----+---------+-------+
| COLLATION_NAME     | varchar(64) | NO   |     |         |       |
| CHARACTER_SET_NAME | varchar(64) | NO   |     |         |       |
| ID                 | bigint(11)  | NO   |     | 0       |       |
| IS_DEFAULT         | varchar(3)  | NO   |     |         |       |
| IS_COMPILED        | varchar(3)  | NO   |     |         |       |
| SORTLEN            | bigint(3)   | NO   |     | 0       |       |
+--------------------+-------------+------+-----+---------+-------+
```

Столбцы представления COLLATIONS имеют следующие значения:

- COLLATION_NAME — название сортировки, доступной текущему пользователю;
- CHARACTER_SET_NAME — название кодировки, по отношению к которой можно задать данную сортировку;
- ID — уникальный идентификатор комбинации кодировка/сортировка;
- IS_DEFAULT — показывает, является ли данная сортировка сортировкой по умолчанию для текущей кодировки, если это так, поле принимает значение YES, в противном случае — NO;
- IS_COMPILED — показывает, скомпилирована ли текущая версия сервера с поддержкой данной кодировки, если это так, поле принимает значение YES, в противном случае — пустую строку;
- SORTLEN — сообщает о дополнительном числе байтов на символ, необходимых для сортировки текста с используемой кодировкой.

Пример запроса представлен в листинге 36.7.

Листинг 36.7. Выборка представления INFORMATION_SCHEMA.COLLATIONS

```
mysql> SELECT * FROM INFORMATION_SCHEMA.COLLATIONS
    -> WHERE CHARACTER_SET_NAME = 'cp1251'\G;
*************************** 1. row ***************************
    COLLATION_NAME: cp1251_bulgarian_ci
CHARACTER_SET_NAME: cp1251
                ID: 14
        IS_DEFAULT:
       IS_COMPILED:
           SORTLEN: 0
*************************** 2. row ***************************
    COLLATION_NAME: cp1251_ukrainian_ci
CHARACTER_SET_NAME: cp1251
                ID: 23
        IS_DEFAULT:
       IS_COMPILED:
           SORTLEN: 0
*************************** 3. row ***************************
    COLLATION_NAME: cp1251_bin
CHARACTER_SET_NAME: cp1251
                ID: 50
        IS_DEFAULT:
```

```
       IS_COMPILED:
           SORTLEN: 0
*************************** 4. row ***************************
    COLLATION_NAME: cp1251_general_ci
CHARACTER_SET_NAME: cp1251
                ID: 51
        IS_DEFAULT: Yes
       IS_COMPILED:
           SORTLEN: 0
*************************** 5. row ***************************
    COLLATION_NAME: cp1251_general_cs
CHARACTER_SET_NAME: cp1251
                ID: 52
        IS_DEFAULT:
       IS_COMPILED:
           SORTLEN: 0
```

Следует отметить, что использовать оператор `SHOW COLLATION` вместо запроса из листинга 36.7 нельзя, т. к. ключевое слово `LIKE` в нем применяется по отношению к полю `Collation`, поэтому осуществлять выборку с условием по полю `Charset` невозможно.

36.3. Представление *COLLATION_CHARACTER_SET_APPLICABILITY*

Представление `COLLATION_CHARACTER_SET_APPLICABILITY` содержит всевозможные комбинации кодировок и сортировок, доступные текущему пользователю. Столбцы представления соответствуют первым двум столбцам представления `COLLATIONS` *(см. разд. 36.2)*. Список столбцов представления `COLLATION_CHARACTER_SET_APPLICABILITY` представлен в листинге 36.8.

Листинг 36.8. Список столбцов представления

```
mysql> DESCRIBE INFORMATION_SCHEMA.COLLATION_CHARACTER_SET_APPLICABILITY;
+--------------------+-------------+------+-----+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+-----+---------+-------+
| COLLATION_NAME     | varchar(64) | NO   |     |         |       |
| CHARACTER_SET_NAME | varchar(64) | NO   |     |         |       |
+--------------------+-------------+------+-----+---------+-------+
```

Столбцы представления COLLATION_CHARACTER_SET_APPLICABILITY имеют следующие значения:

- COLLATION_NAME — название сортировки, доступной текущему пользователю;
- CHARACTER_SET_NAME — название кодировки, по отношению к которой можно задать данную сортировку.

Пример запроса представлен в листинге 36.9.

Листинг 36.9. Выборка сортировок, начинающихся с 'cp1251'

```
mysql> SELECT *
    -> FROM INFORMATION_SCHEMA.COLLATION_CHARACTER_SET_APPLICABILITY
    -> WHERE CHARACTER_SET_NAME = 'cp1251';
+--------------------+--------------------+
| COLLATION_NAME     | CHARACTER_SET_NAME |
+--------------------+--------------------+
| cp1251_bulgarian_ci | cp1251            |
| cp1251_ukrainian_ci | cp1251            |
| cp1251_bin          | cp1251            |
| cp1251_general_ci   | cp1251            |
| cp1251_general_cs   | cp1251            |
+--------------------+--------------------+
```

В листинге 36.9 приводится пример запроса, позволяющего выбрать сортировки, начинающиеся с подстроки 'cp1251'.

36.4. Представление *COLUMN_PRIVILEGES*

Представление COLUMN_PRIVILEGES содержит информацию о привилегиях текущего пользователя на столбцы таблиц.

> ***Замечание***
>
> Данное представление не имеет эквивалента среди операторов SHOW.

В листинге 36.10 приведена результирующая таблица оператора DESCRIBE, примененного к представлению COLUMN_PRIVILEGES.

Листинг 36.10. Список столбцов представления COLUMN_PRIVILEGES

```
mysql> DESCRIBE INFORMATION_SCHEMA.COLUMN_PRIVILEGES;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
```

```
+---------------+--------------+------+-----+---------+-------+
| GRANTEE       | varchar(81)  | NO   |     |         |       |
| TABLE_CATALOG | varchar(512) | YES  |     | NULL    |       |
| TABLE_SCHEMA  | varchar(64)  | NO   |     |         |       |
| TABLE_NAME    | varchar(64)  | NO   |     |         |       |
| COLUMN_NAME   | varchar(64)  | NO   |     |         |       |
| PRIVILEGE_TYPE| varchar(64)  | NO   |     |         |       |
| IS_GRANTABLE  | varchar(3)   | NO   |     |         |       |
+---------------+--------------+------+-----+---------+-------+
```

Столбцы представления `COLUMN_PRIVILEGES` имеют следующие значения:

- `GRANTEE` — учетная запись в форме *`'user'@'host'`*, к которой относится привилегия;
- `TABLE_CATALOG` — в СУБД MySQL это поле всегда содержит `NULL`, т. к. в MySQL отсутствует понятие каталога баз данных;
- `TABLE_SCHEMA` — имя базы данных, содержащей таблицу, которой принадлежит столбец;
- `TABLE_NAME` — имя таблицы, содержащей столбец, на который выставляются привилегии;
- `COLUMN_NAME` — имя столбца, для которого выставляются привилегии;
- `PRIVILEGE_TYPE` — привилегия, поле может принимать одно из четырех значений: `SELECT`, `INSERT`, `UPDATE` или `REFERENCES`;
- `IS_GRANTABLE` — столбец показывает, может ли пользователь передавать свои привилегии на столбцы другим пользователям, т. е. объявлена ли привилегия с ключевым словом `WITH GRANT OPTION` или нет. Если пользователь обладает полномочиями передавать привилегии другим пользователям, поле принимает значение `YES`, в противном случае — `NO`.

В листинге 36.11 приводится пример выборки из представления `INFORMATION_SCHEMA.COLUMN_PRIVILEGES`.

Листинг 36.11. Выборка из `INFORMATION_SCHEMA.COLUMN_PRIVILEGES`

```
mysql> SELECT * FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES\G;
*************************** 1. row ***************************
       GRANTEE: 'root'@'localhost'
 TABLE_CATALOG: NULL
  TABLE_SCHEMA: shop
    TABLE_NAME: catalogs
```

```
   COLUMN_NAME: name
PRIVILEGE_TYPE: SELECT
  IS_GRANTABLE: NO
*************************** 2. row ***************************
       GRANTEE: 'root'@'localhost'
 TABLE_CATALOG: NULL
  TABLE_SCHEMA: shop
    TABLE_NAME: catalogs
   COLUMN_NAME: id_catalog
PRIVILEGE_TYPE: INSERT
  IS_GRANTABLE: NO
*************************** 3. row ***************************
       GRANTEE: 'root'@'localhost'
 TABLE_CATALOG: NULL
  TABLE_SCHEMA: shop
    TABLE_NAME: catalogs
   COLUMN_NAME: name
PRIVILEGE_TYPE: UPDATE
  IS_GRANTABLE: NO
*************************** 4. row ***************************
       GRANTEE: 'root'@'localhost'
 TABLE_CATALOG: NULL
  TABLE_SCHEMA: shop
    TABLE_NAME: catalogs
   COLUMN_NAME: id_catalog
PRIVILEGE_TYPE: REFERENCES
  IS_GRANTABLE: NO
```

Как видно из листинга 36.11, учетная запись `'root'@'localhost'` имеет привилегии на отдельные столбцы таблицы `catalogs` учебной базы данных `shop`. Причем для столбца `name` имеются привилегии `SELECT` и `UPDATE`, а для `id_catalog` — `INSERT` и `REFERENCES`.

> ***Замечание***
>
> Если оператор `SHOW FULL COLUMNS` возвращает привилегии в нижнем регистре (`select`, `insert`, `update` и `references`), то в представлении `COLUMN_PRIVILEGES` привилегии возвращаются в верхнем регистре (`SELECT`, `INSERT`, `UPDATE` и `REFERENCES`).

36.5. Представление *COLUMNS*

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 число столбцов *(см. разд. 30.3)*. Соответствие столбцов представления COLUMNS и результирующей таблицы, возвращаемой оператором SHOW COLUMNS, приведено в табл. 36.3.

Замечание

Столбцы, помеченные "Расширение MySQL", не являются стандартными и свойственны только СУБД MySQL.

Таблица 36.3. *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

В листинге 36.12 приведена результирующая таблица оператора DESCRIBE, примененного к представлению COLUMNS.

Листинг 36.12. Список столбцов представления COLUMNS

```
mysql> 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` — число позиций, отводимых под число, если поле не является числовым, принимает значение `NULL`;
- `NUMERIC_SCALE` — число 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 36.13).

Листинг 36.13. Выборка представления `INFORMATION_SCHEMA.COLUMNS`

```
mysql> SELECT * FROM INFORMATION_SCHEMA.COLUMNS
    -> WHERE TABLE_SCHEMA = 'shop' 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:
```

36.6. Представление *KEY_COLUMN_USAGE*

Представление KEY_COLUMN_USAGE содержит информацию об индексированных столбцах, доступных текущему пользователю.

> ***Замечание***
>
> Данное представление не имеет эквивалента среди операторов SHOW.

> ***Замечание***
>
> Представление содержит сведения об индексированных столбцах, а не о самих индексах. Для получения информации об индексах следует обратиться к представлению STATISTICS *(см. разд. 36.10)*.

В листинге 36.14 приведена результирующая таблица оператора DESCRIBE, примененного к представлению KEY_COLUMN_USAGE.

Листинг 36.14. Список столбцов представления KEY_COLUMN_USAGE

```
mysql> DESCRIBE INFORMATION_SCHEMA.KEY_COLUMN_USAGE;
+------------------------------+-------------+------+-----+---------+
| Field                        | Type        | Null | Key | Default |
+------------------------------+-------------+------+-----+---------+
| CONSTRAINT_CATALOG           | varchar(512) | YES  |     | NULL    |
| CONSTRAINT_SCHEMA            | varchar(64)  | NO   |     |         |
| CONSTRAINT_NAME              | varchar(64)  | NO   |     |         |
| TABLE_CATALOG                | varchar(512) | YES  |     | NULL    |
| TABLE_SCHEMA                 | varchar(64)  | NO   |     |         |
| TABLE_NAME                   | varchar(64)  | NO   |     |         |
| COLUMN_NAME                  | varchar(64)  | NO   |     |         |
| ORDINAL_POSITION             | bigint(10)   | NO   |     | 0       |
| POSITION_IN_UNIQUE_CONSTRAINT | bigint(10)  | YES  |     | NULL    |
| REFERENCED_TABLE_SCHEMA      | varchar(64)  | YES  |     | NULL    |
| REFERENCED_TABLE_NAME        | varchar(64)  | YES  |     | NULL    |
| REFERENCED_COLUMN_NAME       | varchar(64)  | YES  |     | NULL    |
+------------------------------+-------------+------+-----+---------+
```

Каждая строка представления KEY_COLUMN_USAGE соответствует одному индексированному столбцу. Столбцы представления KEY_COLUMN_USAGE имеют следующие значения:

- CONSTRAINT_CATALOG — данное поле всегда принимает значение NULL, т. к. СУБД MySQL не имеет каталогов баз данных;

- CONSTRAINT_SCHEMA — имя базы данных, содержащей таблицу, в которой находится индексированный столбец;
- CONSTRAINT_NAME — имя индексированного столбца, к которому может получить доступ текущий пользователь;
- TABLE_CATALOG — так же как и CONSTRAINT_CATALOG, данное поле всегда принимает значение NULL для СУБД MySQL;
- TABLE_SCHEMA — имя базы данных, содержащей таблицу, в которой находится индексированный столбец;
- TABLE_NAME — имя таблицы, в которой находится индексированный столбец;
- COLUMN_NAME — имя индексированного столбца;
- ORDINAL_POSITION — номер столбца в многостолбцовом индексе;
- POSITION_IN_UNIQUE_CONSTRAINT — для внешнего ключа указывается номер позиции столбца, на который ссылается внешний ключ, в текущих версиях всегда принимает значение NULL;
- REFERENCED_TABLE_SCHEMA — если индекс является внешним ключом, данное поле содержит имя базы данных, в которой расположен первичный ключ;
- REFERENCED_TABLE_NAME — если индекс является внешним ключом, данное поле содержит имя таблицы, в которой расположен первичный ключ;
- REFERENCED_COLUMN_NAME — если индекс является внешним ключом, данное поле содержит имя первичного ключа.

В листинге 36.15 приведена выборка представления KEY_COLUMN_USAGE для таблицы products учебной базы данных shop.

> ***Замечание***
>
> Столбцы REFERENCED_TABLE_SCHEMA, REFERENCED_TABLE_NAME и REFERENCED_COLUMN_NAME добавлены в СУБД MySQL, начиная с версии 5.0.6.

Листинг 36.15. Выборка представления KEY_COLUMN_USAGE

```
mysql> SELECT * FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
    -> WHERE TABLE_SCHEMA = 'shop' AND TABLE_NAME = 'products'\G;
*************************** 1. row ***************************
           CONSTRAINT_CATALOG: NULL
            CONSTRAINT_SCHEMA: shop
              CONSTRAINT_NAME: PRIMARY
                TABLE_CATALOG: NULL
                 TABLE_SCHEMA: shop
                   TABLE_NAME: products
                  COLUMN_NAME: id_product
```

```
             ORDINAL_POSITION: 1
POSITION_IN_UNIQUE_CONSTRAINT: NULL
      REFERENCED_TABLE_SCHEMA: NULL
        REFERENCED_TABLE_NAME: NULL
       REFERENCED_COLUMN_NAME: NULL
```

36.7. Представление *ROUTINES*

Представление ROUTINES содержит список и параметры хранимых процедур и функций, доступных текущему пользователю для выполнения.

Информация в представлении ROUTINES отличается от результирующих таблиц операторов SHOW PROCEDURE STATUS и SHOW FUNCTION STATUS *(см. разд. 33.5.1)*. Соответствие столбцов представления ROUTINES и результирующих таблиц, возвращаемых операторами SHOW PROCEDURE STATUS и SHOW FUNCTION STATUS, дано в табл. 36.4.

Таблица 36.4. *Представление* ROUTINES

Имя столбца	SHOW-эквивалент	Замечание
SPECIFIC_NAME		
ROUTINE_CATALOG		Всегда NULL
ROUTINE_SCHEMA	db	
ROUTINE_NAME	name	
ROUTINE_TYPE	type	
DTD_IDENTIFIER		
ROUTINE_BODY		Всегда SQL
ROUTINE_DEFINITION	body	
EXTERNAL_NAME		Всегда NULL
EXTERNAL_LANGUAGE	language	Всегда NULL
PARAMETER_STYLE		Всегда SQL
IS_DETERMINISTIC	is_deterministic	
SQL_DATA_ACCESS	sql_data_access	
SQL_PATH		Всегда NULL
SECURITY_TYPE	security_type	
CREATED	created	
LAST_ALTERED	modified	
SQL_MODE	sql_mode	Расширение MySQL
ROUTINE_COMMENT	comment	Расширение MySQL
DEFINER	definer	Расширение MySQL

В листинге 36.16 приведена результирующая таблица оператора DESCRIBE, примененного к представлению ROUTINES.

Листинг 36.16. Список столбцов представления ROUTINES

```
mysql> DESCRIBE INFORMATION_SCHEMA.ROUTINES;
+--------------------+--------------+------+-----+---------------------+
| Field              | Type         | Null | Key | Default             |
+--------------------+--------------+------+-----+---------------------+
| SPECIFIC_NAME      | varchar(64)  | NO   |     |                     |
| ROUTINE_CATALOG    | varchar(512) | YES  |     | NULL                |
| ROUTINE_SCHEMA     | varchar(64)  | NO   |     |                     |
| ROUTINE_NAME       | varchar(64)  | NO   |     |                     |
| ROUTINE_TYPE       | varchar(9)   | NO   |     |                     |
| DTD_IDENTIFIER     | varchar(64)  | YES  |     | NULL                |
| ROUTINE_BODY       | varchar(8)   | NO   |     |                     |
| ROUTINE_DEFINITION | longtext     | NO   |     |                     |
| EXTERNAL_NAME      | varchar(64)  | YES  |     | NULL                |
| EXTERNAL_LANGUAGE  | varchar(64)  | YES  |     | NULL                |
| PARAMETER_STYLE    | varchar(8)   | NO   |     |                     |
| IS_DETERMINISTIC   | varchar(3)   | NO   |     |                     |
| SQL_DATA_ACCESS    | varchar(64)  | NO   |     |                     |
| SQL_PATH           | varchar(64)  | YES  |     | NULL                |
| SECURITY_TYPE      | varchar(7)   | NO   |     |                     |
| CREATED            | datetime     | NO   |     | 0000-00-00 00:00:00 |
| LAST_ALTERED       | datetime     | NO   |     | 0000-00-00 00:00:00 |
| SQL_MODE           | longtext     | NO   |     |                     |
| ROUTINE_COMMENT    | varchar(64)  | NO   |     |                     |
| DEFINER            | varchar(77)  | NO   |     |                     |
+--------------------+--------------+------+-----+---------------------+
```

Каждая строка представления ROUTINES соответствует одной хранимой процедуре или функции. Представление содержит информацию обо всех хранимых процедурах и функциях, доступных текущему пользователю. Столбцы представления ROUTINES имеют следующие значения:

- SPECIFIC_NAME — имя хранимой процедуры или функции;
- ROUTINE_CATALOG — данное поле всегда принимает значение NULL, т. к. СУБД MySQL не имеет каталогов баз данных;
- ROUTINE_SCHEMA — имя базы данных, в которой расположена хранимая процедура или функция;

- ROUTINE_NAME — имя хранимой процедуры или функции;
- ROUTINE_TYPE — тип, принимает значение PROCEDURE для хранимых процедур и FUNCTION для хранимых функций;
- DTD_IDENTIFIER — для хранимых функций в данном поле указывается тип возвращаемого значения, для хранимых процедур поле принимает значение NULL;
- ROUTINE_BODY — поле содержит название языка программирования, на котором написана хранимая процедура или функция; в текущих версиях СУБД MySQL поле принимает единственное значение — SQL;
- ROUTINE_DEFINITION — тело хранимой процедуры или функции;
- EXTERNAL_NAME — внешнее имя хранимой процедуры, для текущих версий СУБД MySQL это поле всегда принимает значение NULL, т. к. MySQL не поддерживает в настоящий момент внешних хранимых процедур;
- EXTERNAL_LANGUAGE — название языка программирования, на котором создана внешняя хранимая процедура; поскольку внешние хранимые процедуры не поддерживаются в настоящий момент, данное поле также принимает значение NULL;
- PARAMETER_STYLE — поле отображает, в каком стиле используются параметры хранимых процедур, в текущих версиях MySQL, данное поле принимает единственное значение — SQL;
- IS_DETERMINISTIC — сообщает, возвращает ли хранимая процедура один и тот же результат для одних и тех же входных параметров; если это так, поле принимает значение YES, иначе — NO;
- SQL_DATA_ACCESS — поле показывает, насколько зависит хранимая процедура от данных; 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;
- SQL_PATH — как и поле EXTERNAL_NAME, в текущих версиях СУБД MySQL принимает значение NULL;
- 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;
- CREATED — время создания хранимой процедуры;
- LAST_ALTERED — время последней модификации хранимой процедуры;
- SQL_MODE — режимы выполнения хранимой процедуры;
- ROUTINE_COMMENT — комментарий к хранимой процедуре;
- DEFINER — учетная запись, под которой была создана хранимая процедура.

В листинге 36.17 приведена выборка из представления ROUTINES по шаблону.

Листинг 36.17. Выборка представления ROUTINES

```
mysql> SELECT * FROM INFORMATION_SCHEMA.ROUTINES
    -> WHERE ROUTINE_NAME LIKE 'set%';
*************************** 1. row ***************************
     SPECIFIC_NAME: set_hello
   ROUTINE_CATALOG: NULL
    ROUTINE_SCHEMA: shop
      ROUTINE_NAME: set_hello
      ROUTINE_TYPE: FUNCTION
    DTD_IDENTIFIER: char(50)
      ROUTINE_BODY: SQL
ROUTINE_DEFINITION: RETURN CONCAT('Hello, ',name,'!')
     EXTERNAL_NAME: NULL
 EXTERNAL_LANGUAGE: NULL
   PARAMETER_STYLE: SQL
  IS_DETERMINISTIC: NO
   SQL_DATA_ACCESS: CONTAINS SQL
          SQL_PATH: NULL
     SECURITY_TYPE: DEFINER
           CREATED: 2005-07-23 20:58:23
      LAST_ALTERED: 2005-07-23 20:58:23
          SQL_MODE:
   ROUTINE_COMMENT:
           DEFINER: root@localhost
*************************** 2. row ***************************
     SPECIFIC_NAME: set_var
   ROUTINE_CATALOG: NULL
    ROUTINE_SCHEMA: shop
      ROUTINE_NAME: set_var
      ROUTINE_TYPE: PROCEDURE
    DTD_IDENTIFIER: NULL
      ROUTINE_BODY: SQL
ROUTINE_DEFINITION: BEGIN
  SELECT count INTO @count FROM products
  WHERE id_product = @id_product;
  SET @tmp = @count - @number;
```

```
  UPDATE shop.products
  SET products.count = @tmp
  WHERE products.id_product = @id_product;
END
     EXTERNAL_NAME: NULL
 EXTERNAL_LANGUAGE: NULL
   PARAMETER_STYLE: SQL
  IS_DETERMINISTIC: NO
   SQL_DATA_ACCESS: CONTAINS SQL
          SQL_PATH: NULL
     SECURITY_TYPE: DEFINER
           CREATED: 2005-07-22 21:28:00
      LAST_ALTERED: 2005-07-22 21:28:00
          SQL_MODE:
   ROUTINE_COMMENT:
           DEFINER: root@localhost
```

36.8. Представление *SCHEMA_PRIVILEGES*

Представление SCHEMA_PRIVILEGES содержит глобальные привилегии всех пользователей сервера MySQL.

> ***Замечание***
>
> Данное представление не имеет эквивалента среди операторов SHOW.

В листинге 36.18 приведена результирующая таблица оператора DESCRIBE, примененного к представлению SCHEMA_PRIVILEGES.

Листинг 36.18. Список столбцов представления SCHEMA_PRIVILEGES

```
mysql> DESCRIBE INFORMATION_SCHEMA.SCHEMA_PRIVILEGES;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| GRANTEE       | varchar(81) | NO   |     |         |       |
| TABLE_CATALOG | varchar(512)| YES  |     | NULL    |       |
| TABLE_SCHEMA  | varchar(64) | NO   |     |         |       |
| PRIVILEGE_TYPE| varchar(64) | NO   |     |         |       |
| IS_GRANTABLE  | varchar(3)  | NO   |     |         |       |
+---------------+-------------+------+-----+---------+-------+
```

Строка представления SCHEMA_PRIVILEGES соответствует одной привилегии каждого из пользователей. Столбцы представления SCHEMA_PRIVILEGES имеют следующие значения:

- GRANTEE — учетная запись в форме *'user'@'host'*, к которой относится привилегия;
- TABLE_CATALOG — в СУБД MySQL это поле всегда содержит NULL, т. к. в MySQL отсутствует понятие каталога баз данных;
- TABLE_SCHEMA — имя базы данных, к которой относится привилегия;
- PRIVILEGE_TYPE — имя привилегии, поле может принимать одно из значений табл. 27.1;
- IS_GRANTABLE — столбец показывает, может ли пользователь передавать свои привилегии на столбцы другим пользователям, т. е. объявлена ли привилегия с ключевым словом WITH GRANT OPTION или нет. Если пользователь снабжен полномочиями передавать привилегии другим пользователям, поле принимает значение YES, в противном случае — NO.

В листинге 36.19 приведена выборка из представления SCHEMA_PRIVILEGES с ограничением LIMIT.

Листинг 36.19. Выборка представления SCHEMA_PRIVILEGES

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SCHEMA_PRIVILEGES LIMIT 2\G;
*************************** 1. row ***************************
       GRANTEE: 'wet'@'localhost'
 TABLE_CATALOG: NULL
  TABLE_SCHEMA: shop
PRIVILEGE_TYPE: SELECT
  IS_GRANTABLE: NO
*************************** 2. row ***************************
       GRANTEE: 'wet'@'localhost'
 TABLE_CATALOG: NULL
  TABLE_SCHEMA: shop
PRIVILEGE_TYPE: INSERT
  IS_GRANTABLE: NO
```

Общее число привилегий даже при малом числе пользователей может быть большим (листинг 36.20).

Листинг 36.20. Число привилегий для каждого из пользователей

```
mysql> SELECT GRANTEE, COUNT(*) FROM INFORMATION_SCHEMA.SCHEMA_PRIVILEGES
    -> GROUP BY GRANTEE;
+-----------------------+----------+
| GRANTEE               | COUNT(*) |
+-----------------------+----------+
| ''@'%'                |       28 |
| 'softtime'@'localhost' |        7 |
| 'superuser'@'%'       |        5 |
| 'wet'@'localhost'     |       16 |
+-----------------------+----------+
```

36.9. Представление *SCHEMATA*

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 *(см. разд. 30.6)*. Соответствие столбцов представления SCHEMATA и результирующей таблицы, возвращаемой оператором SHOW DATABASES, показано в табл. 36.5.

Таблица 36.5. *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 36.21 приведена результирующая таблица оператора DESCRIBE, примененного к представлению SCHEMATA.

Листинг 36.21. Список столбцов представления SCHEMATA

```
mysql> 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 36.22 приведена выборка из представления SCHEMATA с ограничением LIMIT.

Листинг 36.22. Выборка представления SCHEMATA

```
mysql> 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
```

36.10. Представление *STATISTICS*

Представление STATISTICS содержит разнообразную информацию об индексах. Информация в представлении отличается от результирующей таблицы оператора SHOW INDEX, в частности, последний выводит сведения строго для определенной таблицы и содержит меньшее число столбцов *(см. разд. 30.11)*. Соответствие столбцов представления STATISTICS и результирующей таблицы, возвращаемой оператором SHOW INDEX, дано в табл. 36.6.

> ***Замечание***
>
> Представление содержит информацию об индексах, а не об индексированных столбцах. Для получения информации об индексированных столбцах следует обратиться к представлению KEY_COLUMN_USAGE *(см. разд. 36.6)*.

Таблица 36.6. *Представление* *STATISTICS*

Имя столбца	SHOW-эквивалент	Замечание
TABLE_CATALOG		Всегда NULL
TABLE_SCHEMA		
TABLE_NAME	Table	
NON_UNIQUE	Non_unique	
INDEX_SCHEMA		
INDEX_NAME	Key_name	
SEQ_IN_INDEX	Seq_in_index	
COLUMN_NAME	Column_name	
COLLATION	Collation	
CARDINALITY	Cardinality	
SUB_PART	Sub_part	Расширение MySQL
PACKED	Packed	Расширение MySQL
NULLABLE	Null	Расширение MySQL
INDEX_TYPE	Index_type	Расширение MySQL
COMMENT	Comment	Расширение MySQL

> ***Замечание***
>
> Столбцы, помеченные как "Расширение MySQL", не являются стандартными и свойственны только для СУБД MySQL.

В листинге 36.23 приведена результирующая таблица оператора DESCRIBE, примененного к представлению STATISTICS.

Листинг 36.23. Список столбцов представления `STATISTICS`

```
mysql> DESCRIBE INFORMATION_SCHEMA.STATISTICS;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| TABLE_CATALOG | varchar(512) | YES  |     | NULL    |       |
| TABLE_SCHEMA  | varchar(64)  | NO   |     |         |       |
| TABLE_NAME    | varchar(64)  | NO   |     |         |       |
| NON_UNIQUE    | bigint(1)    | NO   |     | 0       |       |
| INDEX_SCHEMA  | varchar(64)  | NO   |     |         |       |
| INDEX_NAME    | varchar(64)  | NO   |     |         |       |
| SEQ_IN_INDEX  | bigint(2)    | NO   |     | 0       |       |
| COLUMN_NAME   | varchar(64)  | NO   |     |         |       |
| COLLATION     | varchar(1)   | YES  |     | NULL    |       |
| CARDINALITY   | bigint(21)   | YES  |     | NULL    |       |
| SUB_PART      | bigint(3)    | YES  |     | NULL    |       |
| PACKED        | varchar(10)  | YES  |     | NULL    |       |
| NULLABLE      | varchar(3)   | NO   |     |         |       |
| INDEX_TYPE    | varchar(16)  | NO   |     |         |       |
| COMMENT       | varchar(16)  | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
```

Каждая строка представления `STATISTICS` соответствует одному индексу. Представление содержит информацию об индексах всех таблиц, доступных текущему пользователю. Столбцы представления `STATISTICS` 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 с данным индексом;
- `TABLE_NAME` — имя таблицы, в которой находится данный индекс;
- `NON_UNIQUE` — поле показывает, может ли индекс содержать неуникальные значения; если ответ положительный, поле принимает значение 1, если отрицательный, то 0;
- `INDEX_SCHEMA` — имя таблицы, в которой находится данный индекс;
- `INDEX_NAME` — имя индекса;
- `SEQ_IN_INDEX` — если индекс построен на нескольких столбцах, это поле содержит порядковый номер столбца в индексе;

- `COLUMN_NAME` — имя столбца;
- `COLLATION` — поле информирует о том, как столбец сортируется в индексе. Данное поле может принимать три значения: `A` — восходящая сортировка, `D` — нисходящая сортировка и `NULL` — отсутствие сортировки;
- `CARDINALITY` — количество уникальных значений в индексе. Это значение обновляется при помощи оператора `ANALYZE TABLE`;
- `SUB_PART` — длина префикса. Если индексируется только часть строки, например, первые 5 символов, значение этого поля будет равно 5. Если индексируется весь столбец без ограничений, поле `SUB_PART` содержит значение `NULL`;
- `PACKED` — поле информирует о способе упаковки индекса. Если индекс не упакован, возвращается значение `NULL`;
- `NULLABLE` — поле содержит значение `YES`, если столбец допускает в качестве значения `NULL`, иначе возвращается пустая строка;
- `INDEX_TYPE` — поле информирует об использованном методе индексации (`BTREE`, `FULLTEXT`, `HASH`, `RTREE`);
- `COMMENT` — комментарий для столбца.

В листинге 36.24 приведена выборка из представления `STATISTICS` для таблицы `products` учебной базы данных `shop`.

Листинг 36.24. Выборка представления `STATISTICS`

```
mysql> SELECT * FROM INFORMATION_SCHEMA.STATISTICS\G;
*************************** 1. row ***************************
TABLE_CATALOG: NULL
 TABLE_SCHEMA: shop
   TABLE_NAME: products
   NON_UNIQUE: 0
 INDEX_SCHEMA: test
   INDEX_NAME: PRIMARY
 SEQ_IN_INDEX: 1
  COLUMN_NAME: id_product
    COLLATION: A
  CARDINALITY: 30
     SUB_PART: NULL
       PACKED: NULL
     NULLABLE:
   INDEX_TYPE: BTREE
      COMMENT:
```

```
*************************** 2. row ***************************
TABLE_CATALOG: NULL
 TABLE_SCHEMA: shop
   TABLE_NAME: products
   NON_UNIQUE: 1
 INDEX_SCHEMA: test
   INDEX_NAME: id_catalog
 SEQ_IN_INDEX: 1
  COLUMN_NAME: id_catalog
    COLLATION: A
  CARDINALITY: NULL
     SUB_PART: NULL
       PACKED: NULL
     NULLABLE:
   INDEX_TYPE: BTREE
      COMMENT:
*************************** 3. row ***************************
TABLE_CATALOG: NULL
 TABLE_SCHEMA: shop
   TABLE_NAME: products
   NON_UNIQUE: 1
 INDEX_SCHEMA: test
   INDEX_NAME: search
 SEQ_IN_INDEX: 1
  COLUMN_NAME: name
    COLLATION: NULL
  CARDINALITY: NULL
     SUB_PART: NULL
       PACKED: NULL
     NULLABLE:
   INDEX_TYPE: FULLTEXT
      COMMENT:
*************************** 4. row ***************************
TABLE_CATALOG: NULL
 TABLE_SCHEMA: shop
   TABLE_NAME: products
   NON_UNIQUE: 1
 INDEX_SCHEMA: test
   INDEX_NAME: search
```

```
SEQ_IN_INDEX: 2
 COLUMN_NAME: description
   COLLATION: NULL
 CARDINALITY: NULL
    SUB_PART: NULL
      PACKED: NULL
    NULLABLE: YES
  INDEX_TYPE: FULLTEXT
     COMMENT:
```

36.11. Представление *TABLE_CONSTRAINTS*

Представление TABLE_CONSTRAINTS содержит информацию об ограничивающих индексах, т. е. индексах, которые имеют ограничение уникальности значения (PRIMARY KEY, UNIQUE) или ограничение внешнего ключа (FOREIGN KEY).

> ***Замечание***
>
> Данное представление не имеет эквивалента среди операторов SHOW.

В листинге 36.25 приведена результирующая таблица оператора DESCRIBE, примененного к представлению TABLE_CONSTRAINTS.

Листинг 36.25. Список столбцов представления TABLE_CONSTRAINTS

```
mysql> DESCRIBE INFORMATION_SCHEMA.TABLE_CONSTRAINTS;
+--------------------+--------------+------+-----+---------+-------+
| Field              | Type         | Null | Key | Default | Extra |
+--------------------+--------------+------+-----+---------+-------+
| CONSTRAINT_CATALOG | varchar(512) | YES  |     | NULL    |       |
| CONSTRAINT_SCHEMA  | varchar(64)  | NO   |     |         |       |
| CONSTRAINT_NAME    | varchar(64)  | NO   |     |         |       |
| TABLE_SCHEMA       | varchar(64)  | NO   |     |         |       |
| TABLE_NAME         | varchar(64)  | NO   |     |         |       |
| CONSTRAINT_TYPE    | varchar(64)  | NO   |     |         |       |
+--------------------+--------------+------+-----+---------+-------+
```

Каждая строка представления TABLE_CONSTRAINTS соответствует одному ключу. Представление содержит информацию об ограничивающих ключах (PRIMARY KEY, UNIQUE, FOREIGN KEY) во всех таблицах, доступных текущему пользователю. Столбцы представления TABLE_CONSTRAINTS имеют следующие значения:

- CONSTRAINT_CATALOG — в СУБД MySQL это поле всегда содержит NULL, т. к. в MySQL отсутствует понятие каталога баз данных;

- CONSTRAINT_SCHEMA — имя базы данных, в которой расположена таблица с данным индексом;
- CONSTRAINT_NAME — имя ограничивающего индекса;
- TABLE_SCHEMA — имя базы данных, в которой расположена таблица с данным индексом;
- TABLE_NAME — имя таблицы, в которой находится данный индекс;
- CONSTRAINT_TYPE — тип ограничения, может принимать значения PRIMARY KEY, UNIQUE и FOREIGN KEY.

В листинге 36.26 приведена частичная выборка из представления TABLE_CONSTRAINTS.

Листинг 36.26. Выборка представления TABLE_CONSTRAINTS

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS\G;
*************************** 1. row ***************************
CONSTRAINT_CATALOG: NULL
 CONSTRAINT_SCHEMA: wet
   CONSTRAINT_NAME: PRIMARY
      TABLE_SCHEMA: wet
        TABLE_NAME: products
   CONSTRAINT_TYPE: PRIMARY KEY
*************************** 2. row ***************************
CONSTRAINT_CATALOG: NULL
 CONSTRAINT_SCHEMA: wet
   CONSTRAINT_NAME: products_ibfk_1
      TABLE_SCHEMA: wet
        TABLE_NAME: products
   CONSTRAINT_TYPE: FOREIGN KEY
```

36.12. Представление *TABLE_PRIVILEGES*

Представление TABLE_PRIVILEGES содержит информацию о табличных привилегиях.

> ***Замечание***
>
> Данное представление не имеет эквивалента среди операторов SHOW.

В листинге 36.27 приведена результирующая таблица оператора DESCRIBE, примененного к представлению TABLE_PRIVILEGES.

Листинг 36.27. Список столбцов представления TABLE_PRIVILEGES

```
mysql> DESCRIBE INFORMATION_SCHEMA.TABLE_PRIVILEGES;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| GRANTEE        | varchar(81)  | NO   |     |         |       |
| TABLE_CATALOG  | varchar(512) | YES  |     | NULL    |       |
| TABLE_SCHEMA   | varchar(64)  | NO   |     |         |       |
| TABLE_NAME     | varchar(64)  | NO   |     |         |       |
| PRIVILEGE_TYPE | varchar(64)  | NO   |     |         |       |
| IS_GRANTABLE   | varchar(3)   | NO   |     |         |       |
+----------------+--------------+------+-----+---------+-------+
```

Каждая строка представления TABLE_PRIVILEGES соответствует одной из привилегий, установленной на одну из таблиц базы данных. Столбцы представления имеют следующие значения:

- GRANTEE — учетная запись в форме *'user'@'host'*, к которой относится привилегия;
- TABLE_CATALOG — в СУБД MySQL это поле всегда содержит NULL, т. к. в MySQL отсутствует понятие каталога баз данных;
- TABLE_SCHEMA — имя базы данных, содержащей таблицу, для которой выставлена привилегия;
- TABLE_NAME — имя таблицы, для которой выставлена привилегия;
- PRIVILEGE_TYPE — тип привилегии, может принимать следующие значения: SELECT, INSERT, UPDATE, REFERENCES, ALTER, INDEX, DROP, CREATE VIEW;
- IS_GRANTABLE — столбец показывает, может ли пользователь передавать свои привилегии на столбцы другим пользователям, т. е. объявлена ли привилегия с ключевым словом WITH GRANT OPTION или нет. Если пользователь обладает полномочиями передавать привилегии другим пользователям, поле принимает значение YES, в противном случае — NO.

В листинге 36.28 приведена выборка из представления TABLE_PRIVILEGES для таблиц, для которых выставлена привилегия SELECT.

Листинг 36.28. Выборка представления TABLE_PRIVILEGES

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES
    -> WHERE PRIVILEGE_TYPE = 'SELECT'\G;
*************************** 1. row ***************************
       GRANTEE: 'superuser'@'%'
```

```
 TABLE_CATALOG: NULL
  TABLE_SCHEMA: test
    TABLE_NAME: catalogs
PRIVILEGE_TYPE: SELECT
  IS_GRANTABLE: NO
*************************** 2. row ***************************
       GRANTEE: 'root'@'localhost'
 TABLE_CATALOG: NULL
  TABLE_SCHEMA: test
    TABLE_NAME: catalogs
PRIVILEGE_TYPE: SELECT
  IS_GRANTABLE: NO
```

36.13. Представление *TABLES*

Представление `TABLES` содержит список таблиц и их характеристик. Информация в представлении дублирует результирующую таблицу оператора `SHOW STATUS` *(см. разд. 30.15)*, 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. 36.7.

Таблица 36.7. *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

Таблица 36.7 (окончание)

Имя столбца	SHOW-эквивалент	Замечание
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
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 для СУБД MySQL.

В листинге 36.29 приведена результирующая таблица оператора DESCRIBE, примененного к представлению TABLES.

Листинг 36.29. Список столбцов представления TABLES

```
mysql> 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 *(см. главу 11)*;
- `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` — время создания таблицы;
- `UPDATE_TIME` — 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 36.30 приведена выборка из представления `TABLES` для таблицы `products`.

Листинг 36.30. Выборка представления `TABLE_PRIVILEGES`

```
mysql> 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:
```

36.14. Представление *USER_PRIVILEGES*

Представление `USER_PRIVILEGES` содержит информацию о глобальных привилегиях базы данных.

> ***Замечание***
>
> Представление `USER_PRIVILEGES` не является стандартным.

В листинге 36.31 приведена результирующая таблица оператора `DESCRIBE`, примененного по отношению к представлению `USER_PRIVILEGES`.

Листинг 36.31. Список столбцов представления USER_PRIVILEGES

```
mysql> DESCRIBE INFORMATION_SCHEMA.USER_PRIVILEGES;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| GRANTEE        | varchar(81)  | NO   |     |         |       |
| TABLE_CATALOG  | varchar(512) | YES  |     | NULL    |       |
| PRIVILEGE_TYPE | varchar(64)  | NO   |     |         |       |
| IS_GRANTABLE   | varchar(3)   | NO   |     |         |       |
+----------------+--------------+------+-----+---------+-------+
```

Каждая строка представления TABLES соответствует одной таблице, доступной текущему пользователю. Столбцы представления имеют следующие значения:

- GRANTEE — учетная запись в форме *'user'@'host'*, к которой относится привилегия;
- TABLE_CATALOG — в СУБД MySQL это поле всегда содержит NULL, т. к. в MySQL отсутствует понятие каталога баз данных;
- PRIVILEGE_TYPE — тип привилегии *(см. главу 27)*;
- IS_GRANTABLE — столбец показывает, может ли пользователь передавать свои привилегии на столбцы другим пользователям, т. е. объявлена привилегия с ключевым словом WITH GRANT OPTION или нет. Если пользователь обладает полномочиями передавать привилегии другим пользователям, поле принимает значение YES, в противном случае — NO.

В листинге 36.32 приведена выборка из представления USER_PRIVILEGES для привилегии FILE.

Листинг 36.32. Выборка представления USER_PRIVILEGES для привилегии FILE

```
mysql> SELECT * FROM INFORMATION_SCHEMA.USER_PRIVILEGES
    -> WHERE PRIVILEGE_TYPE = 'FILE';
+-------------------+---------------+----------------+--------------+
| GRANTEE           | TABLE_CATALOG | PRIVILEGE_TYPE | IS_GRANTABLE |
+-------------------+---------------+----------------+--------------+
| 'root'@'localhost' | NULL          | FILE           | YES          |
+-------------------+---------------+----------------+--------------+
```

36.15. Представление *VIEWS*

Представление VIEWS содержит информацию о глобальных привилегиях базы данных. В листинге 36.33 приведена результирующая таблица оператора DESCRIBE, примененного к представлению VIEWS.

Листинг 36.33. Список столбцов представления VIEWS

```
mysql> DESCRIBE INFORMATION_SCHEMA.VIEWS;
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| TABLE_CATALOG   | varchar(512) | YES  |     | NULL    |       |
| TABLE_SCHEMA    | varchar(64)  | NO   |     |         |       |
| TABLE_NAME      | varchar(64)  | NO   |     |         |       |
| VIEW_DEFINITION | longtext     | NO   |     |         |       |
| CHECK_OPTION    | varchar(8)   | NO   |     |         |       |
| IS_UPDATABLE    | varchar(3)   | NO   |     |         |       |
+-----------------+--------------+------+-----+---------+-------+
```

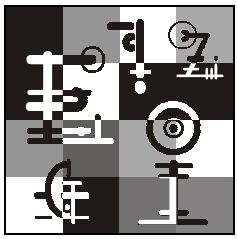
Каждая строка представления VIEWS соответствует одному представлению, доступному текущему пользователю. Столбцы представления VIEWS имеют следующие значения:

- 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 расположено представление;
- TABLE_NAME — имя представления;
- VIEW_DEFINITION — определение представления;
- CHECK_OPTION — значение оператора WITH CHECK OPTION, которое может принимать значения NONE, LOCALE и CASCADED;
- IS_UPDATABLE — поле показывает, могут ли обновляются данные (YES) или нет (NO).

В листинге 36.34 приведена выборка из представления VIEWS.

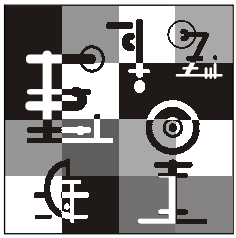
Листинг 36.34. Выборка из представления VIEWS

```
mysql> SELECT * FROM INFORMATION_SCHEMA.VIEWS
    -> WHERE TABLE_NAME = 'cat'\G;
*************************** 1. row ***************************
  TABLE_CATALOG: NULL
   TABLE_SCHEMA: shop
     TABLE_NAME: cat
VIEW_DEFINITION: select `shop`.`catalogs`.`id_catalog` AS
`id_catalog`,`shop`.`catalogs`.`name` AS `name`
from `shop`.`catalogs`
   CHECK_OPTION: NONE
   IS_UPDATABLE: YES
```



ЧАСТЬ V

ВЗАИМОДЕЙСТВИЕ MYSQL С ЯЗЫКАМИ ПРОГРАММИРОВАНИЯ



Глава 37

Взаимодействие MySQL и C/C++

Основное назначение СУБД MySQL — это хранение и управление информацией. В состав MySQL входит ряд вспомогательных утилит, таких как `mysql`, `mysqladmin`, `mysqlimport` и др., однако они предназначены в основном для административных задач и для работы в консоли. Построение графического интерфейса, будь то Windows-программа или Web-сайт, ложится на плечи программиста.

В состав MySQL входит библиотека libmySQL, которая предоставляет программный интерфейс (Application Programming Interface, API). Библиотеку можно найти в каталоге mysql/lib, обычно поставляются два варианта: в подкаталоге debug — отладочный вариант, в подкаталоге opt — оптимизированный. Библиотека создана с использованием языка программирования C, но т. к. большинство других языков программирования также разработаны на C, они подключают библиотеку MySQL. Это позволяет легко взаимодействовать с MySQL из PHP, Perl, Java, Python, Tcl и многих других языков программирования.

Замечание

Предполагается, что читатель знаком либо с языком C, либо C++, и поэтому данная глава не может служить полным руководством по языку программирования.

37.1. Взаимодействие с MySQL в Linux

Традиционно для компиляции программ в операционной системе Linux применяется свободно-распространяемый компилятор gcc совместно с утилитой `make`. К сожалению, рассмотреть их подробно не представляется возможным, поэтому мы лишь укажем параметры, необходимые для компиляции простейших программ, предоставляя читателю самостоятельно изучить все тонкости gcc.

Замечание

Компилятор `gcc` имеет подробное руководство, ознакомиться с которым можно на страницах `info`. Те читатели, которые не знакомы с английским языком, могут ориентироваться на книгу Артура Гриффитца "GCC. Настольная книга пользователей, программистов и системных администраторов". В данном разделе мы приводим описание создания программ для дистрибутива Fedora Core 2.

Компилятор gcc позволяет компилировать программы на нескольких языках программирования. Так, по умолчанию gcc предполагает, что программа написана на языке C, однако, помимо этого языка, gcc может компилировать файлы на C++, Fortran, Java, Ada. Например, для компиляции программ на C++ необходимо вместо gcc использовать вариант g++.

Создадим небольшую программу, которая будет осуществлять подключение к серверу MySQL и сразу же разрывать соединение (листинг 37.1).

> ***Замечание***
>
> Все коды, приведенные в данной главе, можно найти на компакт-диске, прилагаемом к данной книге, в каталоге \code\37.

Листинг 37.1. Соединение с сервером MySQL

```
#include <my_global.h>
#include <mysql.h>

int main(int argc, char *argv[])
{
  // Дескриптор соединения
  MYSQL *conn;
  // Получаем дескриптор соединения
  conn = mysql_init(NULL);
  if(conn == NULL)
  {
    // Если дескриптор не получен — выводим сообщение об ошибке
    fprintf(stderr, "Error: can't create MySQL-descriptor\n");
    exit(1);
  }
  // Подключаемся к серверу
  if(!mysql_real_connect(conn,
                         NULL,
                         "root",
                         NULL,
                         NULL,
                         NULL,
                         0
                        ))
  {
```

```
    // Если нет возможности установить соединение с сервером
    // базы данных, выводим сообщение об ошибке
    fprintf(stderr,
            "Error: can't connect to database %s\n",
            mysql_error(conn));
  }
  else
  {
    // Если соединение успешно установлено, выводим фразу - "Success!"
    fprintf(stdout, "Success!\n");
  }
  // Закрываем соединение с сервером базы данных
  mysql_close(conn);
}
```

Подробно синтаксис функций и значения переменных, приведенные в листинге 37.1, будут обсуждаться позже. Сейчас нам нужна короткая рабочая программа для того, чтобы разобраться в процессе компиляции.

Любой клиентский код для соединения с сервером MySQL должен начинаться с подключения, как минимум, двух заголовочных файлов: my_global.h и mysql.h. Причем заголовочный файл my_global.h должен предварять файл mysql.h. Файл my_global.h включает другие общепринятые заголовочные файлы, такие как stdio.h, math.h, crypt.h и т. п. Файл mysql.h определяет константы и структуры данных, необходимые для работы с сервером MySQL.

Далее в листинге 37.1 глобально объявляются параметры, которые используются для доступа к серверу MySQL. В главной функции `main()` при помощи функции `mysql_init()` указателю `conn` назначается дескриптор соединения с сервером MySQL. Получение дескриптора еще не означает, что соединение с сервером установлено. Для установки соединения с сервером необходимо вызвать функцию `mysql_real_connect()`, которая принимает в качестве первого аргумента дескриптор соединения, а в качестве последующих аргументов — объявленные ранее параметры соединения. Если по какой-либо причине возникает ошибка, то в стандартный поток ошибок выводится сообщение, для этого применяется функция `mysql_error()`. Если соединение устанавливается успешно, программа выводит сообщение "Success!". В завершение программы вызывается функция `mysql_close()`, которая разрывает соединение с сервером.

Теперь необходимо разобраться, каким образом можно откомпилировать представленную в листинге 37.1 программу. Как и любой компилятор, gcc преобразует текстовый файл с программой в объектный код, который в Linux традиционно имеет расширение o. После этого объектные файлы собираются в конечную программу.

Пусть код из листинга 37.1 сохранен в файл `connect.c`. Для успешной компиляции программы, взаимодействующей с СУБД MySQL, необходимо сообщить компилято-

ру gcc, где он может найти дополнительные заголовочные файлы my_global.h и mysql.h. Для этого используется параметр `-I`, в котором следует указать путь к заголовочным файлам MySQL, например, `-I/usr/include/mysql` (листинг 37.2).

> ***Замечание***
>
> Путь /usr/include/mysql характерен для RadHat-подобных дистрибутивов, таких как Fedora Core, в других дистрибутивах и операционных системах этот путь может быть иным.

Листинг 37.2. Компиляция файла connect.c

```
gcc -c -I/usr/include/mysql connect.c
```

Если все введено правильно, то в каталоге, где находится файл connect.c, будет создан объектный файл с именем connect.o. Для сборки исполняемого файла из объектного кода компилятор gcc должен найти библиотеку `mysqlclient`. Для этого в параметре `-l` необходимо указать ее имя, а также задать путь к заголовочным файлам MySQL при помощи параметра `-L` (листинг 37.3).

> ***Замечание***
>
> Большое значение имеет версия библиотеки `mysqlclient`. Долгое время в дистрибутиве Fedora Core поставлялась СУБД MySQL версии 3.23.58, т. к. большая часть программного обеспечения ориентируется на библиотеку этой версии. Лишь начиная с Fedora Core 4, в дистрибутив была введена СУБД MySQL 4.1. Это означает, что, скомпилировав программу совместно с динамической библиотекой `mysqlclient` версии 5.0, клиент вынужден будет устанавливать MySQL версии 5.0, даже если он использует MySQL с более низкой версией.

Листинг 37.3. Сборка исполняемого файла из объектного кода

```
gcc -o connect connect.o -L/usr/include/mysql -lmysqlclient
```

При успешном выполнении команды из листинга 37.3 в каталоге появится файл с именем connect, права доступа которого будут настроены таким образом, чтобы файл можно было выполнять. В RadHat Linux-дистрибутивах для запуска программы `connect` необходимо выполнить команду, представленную в листинге 37.4, или указать полный путь к файлу.

Листинг 37.4. Запуск файла connect на исполнение

```
./connect
```

Последовательность `./` необходима для того, чтобы сообщить операционной системе, что файл запускается из текущего каталога.

Однако набирать каждый раз длинные команды не очень удобно, т. к. можно очень легко ошибиться. Кроме этого, программа может состоять не из одного, а из большого числа файлов. Для автоматизации процесса компиляции обычно применяют утилиту `make`, которая автоматически собирает проект по правилам, указанным в конфигурационном файле Makefile. Конфигурационный файл содержит список зависимостей, команд, а также вспомогательный код, например, макроопределения. Например, для получения объектного файла connect.o, конфигурационный файл Makefile должен содержать связь, представленную в листинге 37.5.

Листинг 37.5. Связь для создания объектного файла connect.o

```
connect.o: connect.c
    gcc -c -I/usr/include/mysql connect.c
```

Первая строка определяет зависимость — файл connect.o создается из файла connect.c. Вторая строка определяет команду (или команды), при помощи которой реализуется зависимость, т. е. в нашем случае создается файл connect.o. Под строкой, задающей зависимость (`connect.o: connect.c`), может быть указано несколько команд, например, если объектный файл собирается из нескольких файлов с исходным кодом.

Следует заметить, что утилита `make` создавалась в первые годы существования UNIX и имеет достаточно специфический синтаксис: командам под строкой, определяющей зависимость (`connect.o: connect.c`), должен предшествовать символ табуляции.

> ***Замечание***
>
> При построении конфигурационного файла Makefile лучше добавлять два символа табуляции вместо одного и использовать редактор, который обеспечивает подсветку синтаксиса Makefile и не осуществляет автоматической замены символов табуляции пробельными символами. Готовый Makefile можно найти на компакт-диске, поставляемом вместе с книгой.

Помимо зависимостей и команд в конфигурационном файле Makefile можно использовать макроопределения. Например, параметры компиляции и сборки `-I/usr/include/mysql` и `-L/usr/include/mysql -lmysqlclient` будут встречаться практически в каждой команде, поэтому их можно обозначить переменными `INCLUDES` и `LIBS`. Для того чтобы значения переменных `INCLUDES` и `LIBS` были подставлены в команды, следует применять последовательности `$(INCLUDES)` и `$(LIB)`, соответственно. Тогда конфигурационный файл для описанной выше задачи может выглядеть так, как это представлено в листинге 37.6.

Листинг 37.6. Конфигурационный файл Makefile

```
CC = gcc
INCLUDES = -I/usr/include/mysql
LIBS = -L/usr/include/mysql -lmysqlclient
```

```
all: connect
connect.o: connect.c
        $(CC) -c $(INCLUDES) connect.c
connect: connect.o
        $(CC) -o connect connect.o $(LIBS)
```

Как правило, для имени компилятора gcc также вводится переменная, это позволяет легко изменять его в случае надобности. Так, если необходимо ввести поддержку C++ (gcc предполагает, что файлы написаны на C), достаточно исправить значение `CC` на `g++`. Кроме того, в ряде операционных систем компилятор может называться не gcc: например, в Mac OS X вместо gcc следует использовать cc.

После того как мы выяснили алгоритм создания программ, осуществляющих соединение с сервером MySQL, рассмотрим более подробно типы данных и функции программного интерфейса.

37.1.1. Типы данных

Интерфейс API C поддерживает несколько собственных типов данных, которые используются главным образом для дескрипторов.

- `MYSQL` — переменная данного типа представляет собой дескриптор соединения с базой данных. Используется почти во всех функциях MySQL. По значению данного типа различаются текущие соединения — это позволяет из одной программы обращаться к серверу сразу от имени нескольких учетных записей или сразу к нескольким MySQL-серверам.
- `MYSQL_RES` — переменная данного типа представляет собой дескриптор результата запроса (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`). Возвращенная из запроса информация далее в этом разделе называется *результирующим набором данных*.
- `MYSQL_ROW` — переменная данного типа является "типобезопасным" представлением записи. В настоящее время этот тип реализован как массив строк с фиксированным количеством байтов (их нельзя трактовать как строки с нулевым символом в конце, если величины полей могут содержать двоичные данные, поскольку они могут содержать 0 байтов). Строки можно получить вызовом функции `mysql_fetch_row()`.
- `MYSQL_FIELD` — переменная данного типа содержит информацию об отдельном поле таблицы: имя поля, тип и его размер. Для каждого поля можно получить структуру `MYSQL_FIELD`, последовательно вызывая функцию `mysql_fetch_field()`. Величины полей не являются частью данной структуры, они содержатся в структуре `MYSQL_ROW`.
- `MYSQL_FIELD_OFFSET` — переменная данного типа является "типобезопасным" представлением позиции поля в списке полей MySQL (используется функцией `mysql_field_seek()`). Позиции являются номерами полей внутри записи, причем нумерация начинается с нуля.

- `my_ulonglong` — данный тип используется для возврата количества записей, а также в функциях `mysql_affected_rows()`, `mysql_num_rows()` и `mysql_insert_id()`. Этот тип обеспечивает диапазон изменений величин от 0 до $1{,}84\times10^{19}$. Для вывода подобной величины следует преобразовать ее в тип `unsigned long` и использовать формат `%lu`. Пример: `printf (Количество строк: %lu\n", (unsigned long) mysql_num_rows(result))`.

Структура `MYSQL_FIELD` является достаточно важной и на ней следует остановиться подробнее. Эта структура включает следующие поля.

- `char *name`. Элемент `name` определяет имя столбца в результирующей таблице и представляет собой строку с нулевым символом в конце. Если полю назначается псевдоним при помощи оператора `AS`, данный элемент будет содержать псевдоним, а не оригинальное имя столбца.
- `char *org_name`. Элемент `org_name` определяет имя столбца в результирующей таблице и представляет собой строку с нулевым символом в конце. Если полю назначается псевдоним при помощи оператора `AS`, данный элемент будет содержать оригинальное имя столбца, а не псевдоним. Данный элемент добавлен в структуру `MYSQL_FIELD`, начиная с версии MySQL 4.1.0.
- `char *table`. Элемент `table` определяет имя таблицы, которой принадлежит текущий столбец, и представляет собой строку с нулевым символом в конце. Следует заметить, что этот элемент `table` структуры `MYSQL_FIELD` заполняется, только если в таблице действительно имеется данный столбец и он не был получен в результате вычислений. Для вычисляемых полей величина *`table`* представляет собой пустую строку. Если таблице назначается псевдоним при помощи оператора `AS`, данный элемент будет содержать псевдоним, а не оригинальное имя таблицы.
- `char *org_table`. Элемент `org_table` определяет имя таблицы, которой принадлежит текущий столбец, и представляет собой строку с нулевым символом в конце. Если таблице назначается псевдоним при помощи оператора `AS`, данный элемент будет содержать оригинальное имя таблицы, а не псевдоним. Данный элемент добавлен в структуру `MYSQL_FIELD`, начиная с версии MySQL 4.1.0.
- `char *db`. Элемент `db` содержит имя базы данных, содержащей таблицу, которой в свою очередь принадлежит текущее поле, и представляет собой строку с нулевым символом в конце. Если текущее поле является вычисляемым, то элемент `db` принимает в качестве значения пустую строку. Данный элемент добавлен в структуру `MYSQL_FIELD`, начиная с версии MySQL 4.1.0.
- `char *catalog`. Элемент `catalog` содержит имя каталога и в текущих версиях MySQL всегда принимает значение `"def"`. Данный элемент добавлен в структуру `MYSQL_FIELD`, начиная с версии MySQL 4.1.1.
- `char *def`. По умолчанию данный элемент принимает пустую строку с нулевым символом в конце. Значение элемента `def` отлично от пустой строки только при использовании функции `mysql_list_fields()`.

- `unsigned int length`. Элемент `length` определяет размер данного поля в том виде, в каком он указан в определении таблицы.
- `unsigned int max_length`. Элемент `max_length` определяет максимальный размер текущего поля в результирующей таблице (по длине самой большой величины поля). При использовании функций `mysql_store_result()` или `mysql_list_fields()` эта переменная содержит максимальную длину для данного поля. При использовании `mysql_use_result()` значение этой переменной равно нулю.
- `enum enum_field_types type`. Элемент определяет тип текущего поля. Величина *type* может принимать одно из значений, представленных в табл. 37.1.
- `unsigned int name_length`. Элемент `name_length` определяет длину строки в элементе `name`. Данный элемент добавлен в структуру `MYSQL_FIELD`, начиная с версии MySQL 4.1.0.
- `unsigned int org_name_length`. Элемент `org_name_length` определяет длину строки в элементе `org_name`. Данный элемент добавлен в структуру `MYSQL_FIELD`, начиная с версии MySQL 4.1.0.
- `unsigned int table_length`. Элемент `table_length` определяет длину строки в элементе *table*. Данный элемент добавлен в структуру `MYSQL_FIELD`, начиная с версии MySQL 4.1.0.
- `unsigned int org_table_length`. Элемент `org_table_length` определяет длину строки в элементе `org_table`. Данный элемент добавлен в структуру `MYSQL_FIELD`, начиная с версии MySQL 4.1.0.
- `unsigned int db_length`. Элемент `db_length` определяет длину строки в элементе `db`. Данный элемент добавлен в структуру `MYSQL_FIELD`, начиная с версии MySQL 4.1.0.
- `unsigned int catalog_length`. Элемент `catalog_length` определяет длину строки в элементе `catalogs`. Данный элемент добавлен в структуру `MYSQL_FIELD`, начиная с версии MySQL 4.1.1.
- `unsigned int def_length`. Элемент `def_length` определяет длину строки в элементе `def`. Данный элемент добавлен в структуру `MYSQL_FIELD`, начиная с версии MySQL 4.1.0.
- `unsigned int flags`. Элемент `flags` определяет статус текущего поля и в качестве значения принимает наборы двоичных флагов. Величина `flags` может иметь ноль или больше двоичных значений из табл. 37.1.

Таблица 37.1. *Значения, которые может принимать элемент* `flags`

Значение флага	Описание флага
`NOT_NULL_FLAG`	Поле не может содержать значение `NULL`
`PRI_KEY_FLAG`	Поле является частью первичного ключа
`UNIQUE_KEY_FLAG`	Поле является частью уникального ключа

***Таблица 37.1** (окончание)*

Значение флага	Описание флага
MULTIPLE_KEY_FLAG	Поле является частью неуникального ключа
UNSIGNED_FLAG	Поле имеет атрибут UNSIGNED
ZEROFILL_FLAG	Поле имеет атрибут ZEROFILL
BINARY_FLAG	Поле имеет атрибут BINARY
AUTO_INCREMENT_FLAG	Поле имеет атрибут AUTO_INCREMENT
ENUM_FLAG	Поле имеет тип ENUM (не рекомендуется)
SET_FLAG	Поле имеет тип SET (не рекомендуется)
BLOB_FLAG	Поле имеет тип BLOB или TEXT (не рекомендуется)
TIMESTAMP_FLAG	Поле имеет тип TIMESTAMP (не рекомендуется)

Использование флагов BLOB_FLAG, ENUM_FLAG, SET_FLAG и TIMESTAMP_FLAG не рекомендуется, поскольку они указывают скорее тип поля, чем атрибут этого типа. Вместо этого более предпочтительно определять тип поля при помощи элемента type, который описывается далее. Пример, представленный в листинге 37.7, демонстрирует наиболее типичное использование элемента flags.

Листинг 37.7. Типичное использование элемента `flags`

```
if (field->flags & NOT_NULL_FLAG)
{
  printf("Поле не может принимать значение NULL\n");
}
```

Для работы с элементом flags можно использовать макросы, представленные в табл. 37.2.

***Таблица 37.2.** Макросы для совместной работы с элементом `flags`*

Статус флага	Описание
IS_NOT_NULL(flags)	Макрос возвращает значение TRUE, если текущее поле снабжено атрибутом NOT NULL

Таблица 37.2 *(окончание)*

Статус флага	Описание
IS_PRI_KEY(flags)	Макрос возвращает значение TRUE, если текущее поле является первичным ключом
IS_BLOB(flags)	Макрос возвращает значение TRUE, если текущее поле имеет тип BLOB или TEXT (не рекомендуется; более предпочтительно field->type)

- unsigned int decimals. Элемент decimals возвращает число десятичных знаков для числовых полей.
- unsigned int charset_nr. Элемент charset_nr определяет номер кодировки поля. Данный элемент добавлен в структуру MYSQL_FIELD, начиная с версии MySQL 4.1.0.
- enum enum_field_types type. Элемент type определяет тип текущего поля. Может принимать одно из значений, представленных в табл. 37.3.

Таблица 37.3. *Допустимые значения элемента* `type`

Значение поля type	Описание
MYSQL_TYPE_TINY	Поле имеет тип TINYINT
MYSQL_TYPE_SHORT	Поле имеет тип SMALLINT
MYSQL_TYPE_LONG	Поле имеет тип INTEGER
MYSQL_TYPE_INT24	Поле имеет тип MEDIUMINT
MYSQL_TYPE_LONGLONG	Поле имеет тип BIGINT
MYSQL_TYPE_DECIMAL	Поле имеет тип DECIMAL или NUMERIC
MYSQL_TYPE_NEWDECIMAL	Поле имеет тип DECIMAL или NUMERIC и используется совместно с математической библиотекой повышенной точности
MYSQL_TYPE_FLOAT	Поле имеет тип FLOAT
MYSQL_TYPE_DOUBLE	Поле имеет тип DOUBLE или REAL
MYSQL_TYPE_BIT	Поле имеет тип BIT
MYSQL_TYPE_TIMESTAMP	Поле имеет тип TIMESTAMP

Таблица 37.3 *(окончание)*

Значение поля `type`	Описание
`MYSQL_TYPE_DATE`	Поле имеет тип `DATE`
`MYSQL_TYPE_TIME`	Поле имеет тип `TIME`
`MYSQL_TYPE_DATETIME`	Поле имеет тип `DATETIME`
`MYSQL_TYPE_YEAR`	Поле имеет тип `YEAR`
`MYSQL_TYPE_STRING`	Поле имеет тип `CHAR` или `BINARY`
`MYSQL_TYPE_VAR_STRING`	Поле имеет тип `VARCHAR` или `VARBINARY`
`MYSQL_TYPE_BLOB`	Поле имеет тип `BLOB` или `TEXT` (используется элемент `max_length` для определения максимальной длины поля)
`MYSQL_TYPE_SET`	Поле имеет тип `SET`
`MYSQL_TYPE_ENUM`	Поле имеет тип `ENUM`
`MYSQL_TYPE_GEOMETRY`	Географический элемент
`MYSQL_TYPE_NULL`	Поле снабжено атрибутом `NULL`
`MYSQL_TYPE_CHAR`	Устаревшее значение, вместо него следует использовать `MYSQL_TYPE_TINY`

Для проверки того факта, является ли текущее поле числовым, можно использовать макрос `IS_NUM()`. В макросе `IS_NUM()` следует указать величину `type` и, если поле имеет числовой тип, будет возвращено значение `TRUE` (листинг 37.8).

Листинг 37.8. Использование макроса `IS_NUM()`

```
if (IS_NUM(field->type))
   printf("Field is numeric\n");
```

37.1.2. Функции интерфейса C

В данном разделе будут кратко рассмотрены функции API C для взаимодействия с MySQL. Рассмотреть детально каждую функцию не представляется возможным, поэтому мы, приведя краткое описание функций, рассмотрим подробно лишь те функции, которые используются наиболее часто для решения типичных задач взаимодействия с MySQL-сервером (соединение с сервером, выполнение запросов, получение результатов и т. п.). Функции интерфейса C перечисляются в табл. 37.4.

Таблица 37.4. *Функции интерфейса C*

Функция	Описание
`mysql_affected_rows()`	Возвращает количество строк, измененных/удаленных/вставленных последним оператором `UPDATE`, `DELETE` или `INSERT`
`mysql_autocommit()`	Переключает режим автоподтверждения транзакции
`mysql_change_user()`	Для уже открытого соединения с базой данных, функция производит переключение на другого пользователя и базу данных
`mysql_charset_name()`	Возвращает название кодировки, установленной для данного соединения
`mysql_close()`	Закрывает соединение с сервером, открытое ранее при помощи функции `mysql_real_connect()`
`mysql_commit()`	Завершает текущую транзакцию *(см. главу 26)*
`mysql_connect()`	Создает соединение с сервером баз данных MySQL. Данная функция является устаревшей, вместо нее следует использовать функцию `mysql_real_connect()`
`mysql_create_db()`	Создает новую базу данных. Данная функция является устаревшей, вместо нее следует использовать оператор `CREATE DATABASE`
`mysql_data_seek()`	Ищет строку по номеру в результирующей таблице
`mysql_debug()`	Отладочные операции `DBUG_PUSH` с заданной строкой
`mysql_drop_db()`	Удаляет базу данных. Данная функция является устаревшей, вместо нее следует использовать оператор `DROP DATABASE`
`mysql_dump_debug_info()`	Заставляет сервер записывать отладочную информацию в журнал
`mysql_eof()`	Определяет, была ли данная строка последней из прочитанных в результирующей таблице. Данная функция является устаревшей, вместо нее рекомендуется применять функции `mysql_errno()` и `mysql_error()`
`mysql_errno()`	Возвращает номер ошибки для последнего выполненного оператора
`mysql_error()`	Возвращает сообщение об ошибке для последнего выполненного оператора

Таблица 37.4 *(продолжение)*

Функция	Описание
`mysql_escape_string()`	Экранирует спецсимволы в строке таким образом, чтобы ее было возможно использовать в SQL-запросе
`mysql_fetch_field()`	Возвращает тип следующего поля записи
`mysql_fetch_field_direct()`	Возвращает тип поля записи по заданному номеру поля
`mysql_fetch_fields()`	Возвращает массив структур, содержащих информацию обо всех полях
`mysql_fetch_lengths()`	Возвращает массив длин всех столбцов в текущей строке
`mysql_fetch_row()`	Извлекает следующую строку из результирующей таблицы
`mysql_field_seek()`	Устанавливает курсор столбцов на заданный столбец
`mysql_field_count()`	Возвращает количество столбцов в результирующей таблице
`mysql_field_tell()`	Возвращает значение положения курсора поля для последнего вызова `mysql_fetch_field()`
`mysql_free_result()`	Освобождает память, зарезервированную для результирующей таблицы
`mysql_get_client_info()`	Возвращает строку с информацией о версии клиента
`mysql_get_client_version()`	Возвращает версии клиента в виде целого числа
`mysql_get_host_info()`	Возвращает строку, описывающую параметры текущего соединения
`mysql_get_server_version()`	Возвращает номер версии сервера как целое число (функция введена, начиная с версии MySQL 4.1)
`mysql_get_proto_info()`	Возвращает версию протокола, используемого для текущего соединения
`mysql_get_server_info()`	Возвращает номер версии сервера баз данных
`mysql_info()`	Возвращает информацию о последнем выполненном запросе
`mysql_init()`	Выделяет или инициализирует структуру `MYSQL`, которая затем используется для установки соединения с сервером MySQL

Таблица 37.4 *(продолжение)*

Функция	Описание
`mysql_insert_id()`	Возвращает идентификатор, сгенерированный для столбца `AUTO_INCREMENT` последним запросом
`mysql_kill()`	Уничтожает заданный поток
`mysql_library_end()`	Отключает библиотеку C API
`mysql_library_init()`	Инициирует библиотеку C API
`mysql_list_dbs()`	Возвращает имена баз данных, совпадающие с упрощенным регулярным выражением
`mysql_list_fields()`	Возвращает имена полей таблицы, совпадающие с упрощенным регулярным выражением
`mysql_list_processes()`	Возвращает список текущих потоков на сервере
`mysql_list_tables()`	Возвращает имена таблиц, совпадающих с упрощенным регулярным выражением
`mysql_more_results()`	Проверяет, существует ли результирующая таблица
`mysql_next_result()`	Возвращает/инициализирует следующую результирующую таблицу при выполнении сразу нескольких запросов
`mysql_num_fields()`	Возвращает количество столбцов в результирующей таблице
`mysql_num_rows()`	Возвращает количество строк в результирующем наборе
`mysql_options()`	Устанавливает параметры соединения для `mysql_connect()`
`mysql_ping()`	Проверяет, работает ли текущее соединение с сервером, и восстанавливает соединение в случае необходимости
`mysql_query()`	Выполняет SQL-запрос, который задается в виде строки с нулевым символом в конце
`mysql_real_connect()`	Осуществляет соединение с сервером MySQL
`mysql_real_escape_string()`	Экранирует специальные символы в строке, чтобы обеспечить возможность использования ее в команде SQL, с учетом установленной для данного соединения кодировки

Таблица 37.4 *(окончание)*

Функция	Описание
`mysql_real_query()`	Выполняет SQL-запрос, заданный в виде фиксированной строки
`mysql_refresh()`	Очищает кэши и таблицы
`mysql_reload()`	Перезагружает таблицы привилегий
`mysql_rollback()`	Отменяет транзакцию *(см. главу 26)*
`mysql_row_seek()`	Устанавливает курсор на заданную строку в результирующей таблице, используя величину, возвращенную функцией `mysql_row_tell()`
`mysql_row_tell()`	Возвращает положение курсора строки
`mysql_select_db()`	При помощи этой функции можно выбрать базу данных
`mysql_server_end()`	Завершает работу встроенного сервера MySQL
`mysql_server_init()`	Инициирует работу встроенного сервера MySQL
`mysql_set_server_option()`	Устанавливает параметры соединения
`mysql_sqlstate()`	Возвращает переменную `SQLSTATE` с кодом ошибки для последней ошибки
`mysql_shutdown()`	Останавливает MySQL-сервер
`mysql_stat()`	Возвращает информацию о текущем статусе сервера MySQL в виде строки
`mysql_store_result()`	Извлекает полную результирующую таблицу для текущего клиента
`mysql_thread_id()`	Возвращает идентификатор текущего потока
`mysql_thread_safe()`	Возвращает 1, если клиенты скомпилированы как поддерживающие потоки
`mysql_use_result()`	Инициализирует построчное извлечение результирующей таблицы
`mysql_warning_count()`	Возвращает количество предупреждений для предыдущего SQL-оператора

Рассмотрим наиболее интересные функции более подробно. Простейший жизненный цикл программы, обращающейся к MySQL-серверу, может выглядеть следующим образом. Перед соединением с сервером необходимо вызвать функцию `mysql_init()` для инициализации дескриптора соединения, которая имеет следующий синтаксис:

```
MYSQL *mysql_init(MYSQL *mysql)
```

Функция возвращает указатель на дескриптор соединения `MYSQL` *(см. разд. 37.1.1)* или `NULL`, если для выделения нового дескриптора не хватает памяти. Если в качестве аргумента функции передается значение `NULL`, функция выделяет память под новый дескриптор и возвращает указатель на него (см. листинг 37.1). Однако функция может принимать в качестве аргумента уже созданную структуру `MYSQL`, в этом случае память не выделяется, а просто производится инициализация структуры (листинг 37.9).

> ***Замечание***
>
> Можно не заботиться об освобождении памяти, выделяемой под дескриптор `MYSQL`, т. к. она освобождается при вызове функции `mysql_close()`.

Листинг 37.9. Использование функции `mysql_init()`

```
// Заголовочные файлы
#include <my_global.h>
#include <mysql.h>
// Дескриптор соединения
MYSQL conn;
int main(int argc, char *argv[])
{
  // Получаем дескриптор соединения
  if(mysql_init(&conn) == NULL)
  {
    // Если дескриптор не получен - выводим сообщение об ошибке
    fprintf(stderr, "Error: can't create MySQL-descriptor\n");
    exit(1);
  }
}
```

Обратите внимание, что функция `mysql_init()` принимает в качестве аргумента указатель, т. е. если объявляется полноценная структура `MYSQL`, необходимо получить ее адрес при помощи оператора `&`.

После того как дескриптор соединения инициирован при помощи функции `mysql_init()`, можно осуществлять соединение с сервером MySQL. Для этого используется функция `mysql_real_connect()`, которая имеет следующий синтаксис:

```
MYSQL *mysql_real_connect (MYSQL *mysql,
                           const char *host,
                           const char *user,
                           const char *passwd,
                           const char *db,
                           unsigned int port,
                           const char *unix_socket,
                           unsigned int client_flag)
```

Если функция успешно устанавливает соединения, она возвращает дескриптор соединения `MYSQL*`, который передается ей в качестве первого аргумента, в противном случае функция возвращает `NULL`. Функция принимает восемь аргументов, которые имеют следующее значение:

- в качестве первого параметра указывается дескриптор, представляющий собой указатель на структуру `MYSQL`, которая должна быть инициализирована функцией `mysql_init()`, до вызова функции `mysql_real_connect()`;
- через параметр *`host`* передается имя хоста или IP-адрес MySQL-сервера. Если параметр принимает значение `NULL` или "`localhost`", то подразумевается соединение с локальным хостом;
- через параметр *`user`* передается имя пользователя MySQL. Если параметр *`user`* принимает значение `NULL`, то подразумевается текущий пользователь. В операционной системе UNIX это будет имя пользователя, которое использовалось для входа в систему. В Windows имя пользователя должно быть указано явным образом;
- через параметр *`passwd`* передается пароль для пользователя *`user`*. Если параметр *`passwd`* равен `NULL`, то успешная авторизация возможна лишь в том случае, если в качестве пароля используется пустая строка. Шифровать пароль перед вызовом функции `mysql_real_connect()` не нужно: шифрование пароля производится автоматически библиотекой `mysqlclient`;
- через параметр *`db`* передается имя базы данных. Если параметр *`db`* не равен `NULL`, то данное соединение установит эту величину в качестве базы данных по умолчанию, в противном случае потребуется дополнительно выбрать базу данных при помощи функции `mysql_select_db()`;
- через параметр *`port`* передается номер порта для TCP/IP-соединения, если параметр принимает значение 0, то в качестве порта будет использоваться номер по умолчанию (3306);
- если параметр *`unix_socket`* не равен `NULL`, то данная строка указывает сокет или именованный канал, который следует использовать;

- величина параметра *client_flag* обычно равна 0, но при особых обстоятельствах может быть установлена как комбинация флагов, представленных в табл. 37.5.

Таблица 37.5. *Допустимые значения параметра* `client_flag`

Флаг	Описание
`CLIENT_COMPRESS`	Предписывает использовать сжатие в протоколе обмена клиента и сервера
`CLIENT_FOUND_ROWS`	Предписывает возвращать количество найденных (совпавших) строк, а не количество строк, подвергшихся воздействию
`CLIENT_IGNORE_SPACE`	Разрешает использовать пробелы после имен встроенных функций. При этом имена всех функций становятся зарезервированными словами
`CLIENT_INTERACTIVE`	Разрешает простой длительностью `interactive_timeout` секунд (вместо `wait_timeout` секунд) перед закрытием данного соединения
`CLIENT_LOCAL_FILES`	Разрешает использование оператора `LOAD DATA LOCAL`
`CLIENT_MULTI_STATEMENTS`	Сообщает серверу, что клиент может посылать сразу несколько операторов, разделенных точкой с запятой (`;`). Данный флаг введен, начиная с версии MySQL 4.1
`CLIENT_MULTI_RESULTS`	Сообщает серверу, что клиент может обрабатывать несколько результирующих таблиц, если серверу передается несколько операторов, разделенных точкой с запятой, или используется хранимая процедура. Этот флаг автоматически устанавливается, если установлен флаг `CLIENT_MULTI_STATEMENTS`. Данный флаг введен, начиная с версии MySQL 4.1
`CLIENT_NO_SCHEMA`	Запрещает использование формы *db_name.tbl_name.col_name*. Это делается для ODBC и заставляет синтаксический анализатор генерировать ошибку при использовании данного синтаксиса
`CLIENT_ODBC`	Сообщает серверу, что клиент является клиентом ODBC. В результате сервер MySQL настраивается для большей совместимости с ODBC
`CLIENT_SSL`	Предписывает использовать SSL (протокол шифрования). Данный флаг не должен устанавливаться прикладной программой, т. к. устанавливается внутри библиотеки `mysqlclient`

Для закрытия соединения вызывается функция `mysql_close()`, которая имеет следующий синтаксис:

```
void mysql_close(MYSQL *mysql)
```

Пример использования функций `mysql_real_connect()` и `mysql_close()` приводится в листинге 37.10.

Листинг 37.10. Использование функций `mysql_real_connect()` и `mysql_close()`

```
// Заголовочные файлы
#include <my_global.h>
#include <mysql.h>
// Дескриптор соединения
MYSQL conn;
int main(int argc, char *argv[])
{
  // Получаем дескриптор соединения
  if(mysql_init(&conn) == NULL)
  {
    // Если дескриптор не получен - выводим сообщение об ошибке
    fprintf(stderr, "Error: can't create MySQL-descriptor\n");
    exit(1);
  }
  // Устанавливаем соединение с базой данных
  if(!mysql_real_connect(&conn,
                         "localhost",
                         "user",
                         "passwd",
                         "database",
                         0,
                         NULL,
                         0))
  {
    // Если соединение не установлено, выводим сообщение об ошибке
    fprintf(stderr,
            "Error: can't connect to database %s\n",
            mysql_error(&conn));
    exit(1);
```

```
  }
  // Закрываем соединение с сервером базы данных
  mysql_close(&conn);
}
```

Для контроля успешного или неудачного выполнения функций API C можно следить при помощи двух специальных функций `mysql_error()` и `mysql_errno()`. Функция `mysql_error()` возвращает сообщение об ошибке для последней вызванной функции и имеет следующий синтаксис:

```
char *mysql_error(MYSQL *mysql)
```

Если ошибка не возникала, то возвращается пустая строка (""). Функция `mysql_errno()` возвращает код ошибки для последней запущенной функции и имеет следующий синтаксис:

```
unsigned int mysql_errno(MYSQL *mysql)
```

Если функция возвращает значение, отличное от нуля, это означает, что предыдущая функция завершилась с ошибкой. Номера сообщений об ошибке для клиентов перечислены в заголовочном файле MySQL errmsg.h. Номера серверных сообщений об ошибке даны в файле mysqld_error.h.

> ***Замечание***
>
> В исходном дистрибутиве MySQL можно найти полный список сообщений об ошибках и номеров ошибок в файле Docs\mysqld_error.txt.

В файлах errmsg.h и mysqld_error.h каждому коду ошибки сопоставляется своя константа. Если планируется использовать их в программе, необходимо включить данные заголовочные файлы. Пусть функции `mysql_real_connect()` передается неправильный адрес MySQL-сервера — `"wrong-host"` (листинг 37.11).

Листинг 37.11. Обработка ошибок

```
// Заголовочные файлы
#include <my_global.h>
#include <mysql.h>
#include <errmsg.h>
#include <mysqld_error.h>
// Дескриптор соединения
MYSQL conn;
int main(int argc, char *argv[])
{
  // Получаем дескриптор соединения
  if(!mysql_init(&conn))
```

```
  {
    // Если дескриптор не получен - выводим сообщение об ошибке
    fprintf(stderr, "Error: can't create MySQL-descriptor\n");
    exit(1);
  }
  // Устанавливаем соединение с базой данных
  if(!mysql_real_connect(&conn,
                         "wrong-host",
                         "user",
                         "passwd",
                         "database",
                         0,
                         NULL,
                         0))
  {
    // Если соединение не установлено, выводим сообщение об ошибке
    fprintf(stderr,
            "Error: can't connect to database %s\n",
            mysql_error(&conn));
    fprintf(stderr,
            "Error: number of error %d\n",
            mysql_errno(&conn));
    fprintf(stderr,
            "CR_UNKNOWN_HOST %d\n",
            CR_UNKNOWN_HOST);
    exit(1);
  }
  // Закрываем соединение с сервером базы данных
  mysql_close(&conn);
}
```

При выполнении программы из листинга 37.11 результат, который выводится на консоль, может выглядеть так, как это представлено в листинге 37.12.

Листинг 37.12. Результат выполнения программы из листинга 37.11

```
Error: can't connect to database Unknown MySQL server host 'wrong-host' (2)
Error: number of error 2005
CR_UNKNOWN_HOST 2005
```

Как видно из листинга 37.12, ошибке "Не найден хост MySQL-сервера" (код 2005) соответствует константа `CR_UNKNOWN_HOST`. Использование констант вместо чисел гораздо нагляднее и позволяет изменять нумерацию ошибок без изменения кода программы. Список клиентских ошибок, которые могут происходить при вызове функции `mysql_real_connect()`, представлен в табл. 37.6.

Таблица 37.6. *Список клиентских ошибок для функции* `mysql_real_connect()`

Константа	Код	Описание
`CR_CONN_HOST_ERROR`	2003	Не удалось соединиться с сервером MySQL
`CR_CONNECTION_ERROR`	2002	Не удалось соединиться с локальным сервером MySQL
`CR_IPSOCK_ERROR`	2004	Не удалось создать IP-сокет
`CR_OUT_OF_MEMORY`	2008	Недостаток памяти
`CR_SOCKET_CREATE_ERROR`	2001	Не удалось создать UNIX-сокет
`CR_UNKNOWN_HOST`	2005	Не удалось найти IP-адрес для данного имени хоста
`CR_VERSION_ERROR`	2007	Несоответствие протокола, что явилось результатом попытки соединения с сервером с клиентской библиотекой, использующей иную версию протокола. Это может произойти при использовании очень старой клиентской библиотеки для подключения к новому серверу, при запуске которого не был установлен параметр `--old-protocol`
`CR_NAMEDPIPEOPEN_ERROR`	2017	Не удалось создать именованный канал (только для операционной системы Windows)
`CR_NAMEDPIPEWAIT_ERROR`	2016	Не удалось дождаться именованного канала (только для операционной системы Windows)
`CR_NAMEDPIPESETSTATE_ERROR`	2018	Не удалось получить обработчик канала (только для операционной системы Windows)
`CR_SERVER_LOST`	2013	Если `connect_timeout > 0` и требовалось больше, чем `connect_timeout` секунд для соединения с сервером или если сервер прекратил работу во время выполнения `init-command`

Для выполнения запросов обычно используется одна из двух функций: `mysql_query()` или `mysql_real_query()`. Для простоты выполним операцию вставки новой записи в таблицу `catalogs` учебной базы данных `shop` при помощи оператора `INSERT`. Для этого воспользуемся функцией `mysql_query()`, которая имеет следующий синтаксис:

```
int mysql_query(MYSQL *mysql, const char *query)
```

Функция mysql_query() выполняет SQL-запрос, который передается через аргумент *query* в виде строки с нулевым окончанием.

> **Замечание**
>
> Запрос *query* должен состоять из одного SQL-оператора и не содержать завершающих элементов: точку с запятой (;) или \G.

При успешном выполнении запроса функция возвращает нулевое значение, в противном случае возвращается числовой код, которому соответствует одна из констант из табл. 37.7.

***Таблица 37.7.** Список клиентских ошибок для функции mysql_query()*

Константа	Код	Описание
CR_COMMANDS_OUT_OF_SYNC	2014	Команды были выполнены в ненадлежащем порядке
CR_SERVER_GONE_ERROR	2006	Сервер MySQL неожиданно завершил работу
CR_SERVER_LOST	2013	Соединение с сервером прервалось в процессе данного запроса
CR_UNKNOWN_ERROR	2000	Произошла неизвестная ошибка

Теперь можно приступить к вставке новой записи в таблицу catalogs. Программа, осуществляющая эту операцию, может выглядеть так, как это представлено в листинге 37.13.

Листинг 37.13. Вставка новой записи в таблицу catalogs

```
// Заголовочные файлы
#include <my_global.h>
#include <mysql.h>
#include <errmsg.h>
#include <mysqld_error.h>
int main(int argc, char *argv[])
{
  // Дескриптор соединения
  MYSQL conn;

  // Получаем дескриптор соединения
  if(!mysql_init(&conn))
  {
    // Если дескриптор не получен - выводим сообщение об ошибке
```

```
    fprintf(stderr, "Error: can't create MySQL-descriptor\n");
    exit(1);
  }
  // Устанавливаем соединение с базой данных
  if(!mysql_real_connect(&conn,
                         "localhost",
                         "root",
                         "",
                         "shop",
                         0,
                         NULL,
                         0))
  {
    // Если соединение не установлено, выводим сообщение об ошибке
    fprintf(stderr, "Error: %s\n", mysql_error(&conn));
    exit(1);
  }
  // Устанавливаем кодировку соединения, чтобы предотвратить
  // искажения русского текста
  if(mysql_query(&conn, "SET NAMES 'cp1251'") != 0)
  {
    // Если кодировку установить невозможно,
    // выводим сообщение об ошибке
    fprintf(stderr, "Error: can't set character set\n");
    exit(1);
  }
  // Добавляем новый раздел - "Накопители"
  if(mysql_query(&conn,
              "INSERT INTO catalogs VALUES (NULL,'Накопители')") != 0)
  {
    // Если добавить запись не получилось - выводим
    // сообщение об ошибке
    fprintf(stderr, "Error: can't execute INSERT-query\n");
    exit(1);
  }

  // Закрываем соединение с сервером базы данных
  mysql_close(&conn);
}
```

Следует отметить, что перед тем как помещать в базу данных русский текст, следует настроить кодировку соединения при помощи оператора `SET NAMES`, в противном случае русский текст будет искажен (текст будет заменен знаками вопроса). После выполнения запроса можно удостовериться в том, что запись добавлена при помощи запроса `SELECT` (листинг 37.14).

Листинг 37.14. Содержимое таблицы `catalogs` учебной базы данных `shop`

```
mysql> SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
|          6 | Накопители         |
+------------+--------------------+
```

Однако в больших программах контроль правильности выполнения SQL-оператора необходимо осуществлять программно. Для этого, помимо рассмотренных ранее функций `mysql_error()` и `mysql_errno()`, можно применять функцию `mysql_affected_rows()`. Эта функция возвращает число записей, которые затронул последний SQL-оператор (`INSERT`, `UPDATE`, `DELETE`). Функция `mysql_affected_rows()` имеет следующий синтаксис:

```
my_ulonglong mysql_affected_rows(MYSQL *mysql)
```

Функция возвращает число подвергнувшихся изменению записей. Если SQL-запрос, переданный функции `mysql_query()`, завершился ошибкой, функция `mysql_affected_rows()` возвращает значение –1.

Особенностью строк в C и C++ является тот факт, что их длина вычисляется от начала строки до первого нулевого символа. Однако при вставке в таблицу бинарных данных, нулевой символ может встречаться много раз и первый же символ будет воспринят как конец строки, а следовательно, и SQL-оператора. Для того чтобы предотвратить такую трактовку, вместо функции `mysql_query()` применяют функцию `mysql_real_query()`, которая имеет следующий синтаксис:

```
int mysql_real_query(MYSQL *mysql,
                     const char *query,
                     unsigned long length)
```

Функция `mysql_real_query()` выполняет SQL-запрос, указанный в *`query`*, который должен быть строкой длиною *`length`* байтов.

Замечание

Запрос *query* должен состоять из одного SQL-оператора и не содержать завершающих элементов: точку с запятой (;) или \G.

До этого момента нами рассматривались запросы, не возвращающие результата. Однако больший интерес представляют запросы, возвращающие результирующую таблицу (SELECT, SHOW, DESCRIBE, EXPLAIN).

После того как SQL-запрос был выполнен при помощи функции mysql_query() или mysql_real_query(), через функцию mysql_store_result() можно получить указатель на структуру MYSQL_RES, которая является дескриптором результирующей таблицы. Функция mysql_store_result() имеет следующий синтаксис:

```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

Функция принимает дескриптор соединения с сервером *mysql* и возвращает дескриптор результирующей таблицы или NULL в случае неудачи. Список возможных ошибок, которые возвращают функции mysql_error() и mysql_errno(), для функции mysql_store_result() точно такой же, как и для функции mysql_query() (см. табл. 37.7).

После того как дескриптор результирующей таблицы получен, извлечь строки из данной таблицы можно при помощи функции mysql_fetch_row(), которая имеет следующий синтаксис:

```
MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)
```

Функция принимает дескриптор результирующей таблицы *result* и возвращает дескриптор строки MYSQL_ROW. При первом вызове функции возвращается первая строка результирующей таблицы, при втором вызове — вторая и т. д. Если в результирующей таблице больше не осталось строк, то функция возвращает NULL. Это позволяет извлекать строки результирующей таблицы в цикле.

В листинге 37.15 демонстрируется извлечение версии сервера MySQL при помощи SQL-запроса SELECT VERSION().

Листинг 37.15. Извлечение версии MySQL-сервера

```
// Заголовочные файлы
#include <my_global.h>
#include <mysql.h>

// Прототип функции обработки ошибок
void puterror(char *);

// Главная функция программы
int main(int argc, char *argv[])
```

```
{
  // Дескриптор соединения
  MYSQL conn;
  // Дескриптор результирующей таблицы
  MYSQL_RES *res;
  // Массив полей текущей строки
  MYSQL_ROW row;

  // Получаем дескриптор соединения
  if(!mysql_init(&conn))
     puterror("Error: can't create MySQL-descriptor\n");

  // Устанавливаем соединение с базой данных
  if(!mysql_real_connect(&conn,
                         "localhost",
                         "root",
                         "",
                         "shop",
                         0,
                         NULL,
                         0))
     puterror("Error: can't connect to MySQL server\n");

  // Выполняем SQL-запрос
  if(mysql_query(&conn, "SELECT VERSION()") != 0)
     puterror("Error: can't execute SQL-query\n");

  // Получаем дескриптор результирующей таблицы
  res = mysql_store_result(&conn);
  if(res == NULL)
  {
    puterror("Error: can't get the result description\n");
  }

  // Получаем первую строку из результирующей таблицы
  row = mysql_fetch_row(res);
  if(mysql_errno(&conn) > 0) puterror("Error: can't fetch result\n");

  // Выводим результат в стандартный поток
```

```
  fprintf(stdout, "Version: %s\n", row[0]);

  // Освобождаем память, занятую результирующей таблицей
  mysql_free_result(res);

  // Закрываем соединение с сервером базы данных
  mysql_close(&conn);
}

void puterror(char * str)
{
  fprintf(stderr, str);
  exit(1);
}
```

Как видно из листинга 37.15, для обработки ошибок вводится специальная функция `puterror()`, которая выводит сообщение об ошибке в стандартный поток ошибок и останавливает работу программы. В начале функции `main()` объявляются три дескриптора:

- `MYSQL` — дескриптор соединения с сервером;
- `MYSQL_RES *` — дескриптор результирующей таблицы;
- `MYSQL_ROW` — массив полей текущей строки.

Функция `mysql_fetch_row()` возвращает массив `row` типа `MYSQL_ROW` для первой строки результирующей таблицы. Так как нам заранее известно, что результат запроса `SELECT VERSION()` находится в первой строке и первом элементе результирующей таблицы, мы просто обращаемся к первому элементу массива — `row[0]`. В стандартный поток версия MySQL-сервера выводится при помощи строки

```
fprintf(stdout, "Version: %s\n", row[0]);
```

Результат работы программы может выглядеть так, как это представлено в листинге 37.16.

Листинг 37.16. Результат выполнения программы из листинга 37.15

```
./mysql_store_result
Version: 5.0.18-standard
```

Результирующая таблица, которая формируется оператором `mysql_store_result()`, занимает определенный объем оперативной памяти, если его не освободить, он останется зарезервированным за программой до тех пор, пока она не завершит работу. Это особенно заметно, когда программа выполняется многие часы или дни, без пере-

загрузки: процесс разбухания процесса программы без возврата памяти в систему называется *утечкой памяти*. Для предотвращения утечки памяти необходимо применять оператор `mysql_free_result()`, который освобождает память, выделенную под результирующую таблицу. Оператор `mysql_free_result()` имеет следующий синтаксис:

```
void mysql_free_result(MYSQL_RES *result)
```

В качестве единственного параметра функция принимает указатель на структуру `MYSQL_RES`, которую возвращает функции `mysql_store_result()`.

Выяснить версию MySQL-сервера можно также специальной функцией интерфейса `mysql_get_server_version()`, которая имеет следующий синтаксис:

```
unsigned long mysql_get_server_version(MYSQL *mysql)
```

Функция принимает дескриптор соединения и возвращает версию MySQL-сервера, которая вычисляется по следующей схеме:

```
main_version * 10000 + minor_version * 100 + sub_version
```

Здесь *main_version* — первая цифра версии, *minor_version* — вторая, а *sub_version* — третья. Таким образом, для версии 5.0.18 будет возвращено число 50018 (листинг 37.17).

Листинг 37.17. Альтернативный способ извлечения версии MySQL-сервера

```
// Заголовочные файлы
#include <my_global.h>
#include <mysql.h>
// Вспомогательная функция для вывода ошибок
void puterror(char *);

// Главная функция программы
int main(int argc, char *argv[])
{
  // Дескриптор соединения
  MYSQL conn;

  // Получаем дескриптор соединения
  if(!mysql_init(&conn))
     puterror("Error: can't create MySQL-descriptor\n");

  // Устанавливаем соединение с базой данных
  if(!mysql_real_connect(&conn,
                         "localhost",
```

```
                        "root",
                        "",
                        "shop",
                        0,
                        NULL,
                        0))
    puterror("Error: can't connect to MySQL server\n");

  // Выводим результат в стандартный поток
  fprintf(stdout, "Version: %lu\n", mysql_get_server_version(&conn));

  // Закрываем соединение с сервером базы данных
  mysql_close(&conn);
}

void puterror(char * str)
{
  fprintf(stderr, str);
  exit(1);
}
```

Большинство функций интерфейса API C для взаимодействия с СУБД MySQL возвращают результат типа `unsigned long`, поэтому в строке форматирования вывода функции `fprintf()` необходимо использовать спецификатор `%lu`.

В рассмотренном выше примере обработки запроса, возвращающего результат, было заранее известно число строк и столбцов в результирующей таблице. Когда это число строк в результирующей таблице не известно заранее, его можно оценить при помощи функции `mysql_num_rows()`, которая имеет следующий синтаксис:

```
my_ulonglong mysql_num_rows(MYSQL_RES *result)
```

Функция принимает указатель на результирующую таблицу и возвращает число строк в результирующей таблице. Создадим программу, которая будет выводить названия каталогов из таблицы `catalogs`, при условии, что в таблице имеется хотя бы одна запись (листинг 37.18).

Листинг 37.18. Вывод содержимого таблицы `catalogs`

```
// Заголовочные файлы
#include <my_global.h>
#include <mysql.h>
```

```
// Вспомогательная функция для вывода ошибок
void puterror(char *);

// Главная функция программы
int main(int argc, char *argv[])
{
  // Дескриптор соединения
  MYSQL conn;
  // Дескриптор результирующей таблицы
  MYSQL_RES *res;
  // Дескриптор строки
  MYSQL_ROW row;

  // Получаем дескриптор соединения
  if(!mysql_init(&conn))
     puterror("Error: can't create MySQL-descriptor\n");

  // Устанавливаем соединение с базой данных
  if(!mysql_real_connect(&conn,
                         "localhost",
                         "root",
                         "",
                         "shop",
                         0,
                         NULL,
                         0))
     puterror("Error: can't connect to MySQL server\n");

  // Устанавливаем кодировку соединения, чтобы предотвратить
  // искажение русского текста
  if(mysql_query(&conn, "SET NAMES 'utf8'") != 0)
     puterror("Error: can't set character set\n");

  // Выполняем SQL-запрос
  if(mysql_query(&conn, "SELECT * FROM catalogs") != 0)
     puterror("Error: can't execute SQL-query\n");

  // Получаем дескриптор результирующей таблицы
  res = mysql_store_result(&conn);
```

```
  if(res == NULL) puterror("Error: can't get the result description\n");

  // Если имеется хотя бы одна запись - выводим
  // список каталогов
  if(mysql_num_rows(res) > 0)
  {
    // В цикле перебираем все записи
    // результирующей таблицы
    while((row = mysql_fetch_row(res)) != NULL)
    {
      // Выводим результат в стандартный поток
      fprintf(stdout, "%s\n", row[1]);
    }
  }

  // Освобождаем память, занятую результирующей таблицей
  mysql_free_result(res);

  // Закрываем соединение с сервером базы данных
  mysql_close(&conn);
}

void puterror(char * str)
{
  fprintf(stderr, str);
  exit(1);
}
```

Как видно из листинга 37.18, после того как при помощи функции `mysql_num_rows()` программа убеждается, что в результирующей таблице имеется хотя бы одна запись, управление передается циклу `while()`. Цикл `while()` извлекает строки из результирующей таблицы до тех пор, пока функция `mysql_fetch_row()` не вернет `NULL`, свидетельствующий о том, что в результирующей таблице не осталось необработанных строк.

При выводе содержимого строки из массива `row` подразумевается, что строка является обычной последовательностью символов, завершающейся нулевым символом. Однако если в таблице содержатся бинарные данные, в состав которых может входить несколько нулевых символов, перед извлечением строки необходимо определить размер извлекаемых полей. Это можно осуществить при помощи функции `mysql_fetch_lengths()`, которая имеет следующий синтаксис:

```
unsigned long *mysql_fetch_lengths(MYSQL_RES *result)
```

Функция принимает в качестве единственного аргумента указатель на результирующую таблицу и возвращает массив длин полей в результирующей таблице.

В листинге 37.19 выполняется `SELECT`-запрос для таблицы `products` учебной базы данных `shop`. Для каждого поля таблицы `products` (нумерация полей ведется с 0) выводится максимальная длина в байтах.

Листинг 37.19. Вычисление реального размера столбцов в результирующей таблице

```
// Заголовочные файлы
#include <my_global.h>
#include <mysql.h>
// Вспомогательная функция для вывода ошибок
void puterror(char *);

// Главная функция программы
int main(int argc, char *argv[])
{
  // Дескриптор соединения
  MYSQL conn;
  // Дескриптор результирующей таблицы
  MYSQL_RES *res;
  // Дескриптор строки
  MYSQL_ROW row;

  unsigned long *lengths;
  unsigned int i;

  // Получаем дескриптор соединения
  if(!mysql_init(&conn))
     puterror("Error: can't create MySQL-descriptor\n");

  // Устанавливаем соединение с базой данных
  if(!mysql_real_connect(&conn,
                         "localhost",
                         "root",
                         "",
                         "shop",
                         0,
                         NULL,
```

```
                          0))
     puterror("Error: can't connect to MySQL server\n");

  // Выполняем SQL-запрос
  if(mysql_query(&conn, "SELECT * FROM products") != 0)
     puterror("Error: can't execute SQL-query\n");

  // Получаем дескриптор результирующей таблицы
  res = mysql_store_result(&conn);
  if(res == NULL)
  {
    puterror("Error: can't get the result description\n");
  }

  // Читаем первую строку
  row = mysql_fetch_row(res);
  if(row)
  {
    // Получаем максимальный размер каждого из столбца
    lengths = mysql_fetch_lengths(res);
    for(i = 0; i < mysql_num_fields(res); i++)
    {
      // Выводим максимальный размер каждого столбца
      // таблицы products
      printf("Column %u is %lu bytes in length.\n", i, lengths[i]);
    }
  }

  // Освобождаем память, занятую результирующей таблицей
  mysql_free_result(res);

  // Закрываем соединение с сервером базы данных
  mysql_close(&conn);
}

void puterror(char * str)
{
  fprintf(stderr, str);
  exit(1);
}
```

Результат работы программы из листинга 37.19 может выглядеть так, как это представлено в листинге 37.20.

Листинг 37.20. Результат выполнения программы из листинга 37.19

```
./mysql_fetch_lengths
Column 0 is 1 bytes in length.
Column 1 is 11 bytes in length.
Column 2 is 7 bytes in length.
Column 3 is 2 bytes in length.
Column 4 is 3 bytes in length.
Column 5 is 59 bytes in length.
Column 6 is 1 bytes in length.
```

37.2. Взаимодействие с MySQL в Windows

Операционная система Windows задумывалась как графическая операционная система, максимально дружелюбная пользователю. Архитектура ее такова, что поддержка графического пользовательского интерфейса реализована на уровне операционной системы. Для сравнения, например, в UNIX-подобной операционной системе используется графический сервер X Window, который устанавливается подобно Web-серверу или серверу MySQL и является надстройкой. Жесткое интегрирование графического интерфейса в ядро операционной системы имеет как свои достоинства (интерфейс работает очень быстро и эффективно в отличие от X Window), так и недостатки (даже если компьютер под управлением Windows используется как сервер и графика не нужна — она все равно потребляет память). Главной особенностью программирования под Windows является тот факт, что программировать гораздо сложнее, чем в UNIX, т. к. требуется не просто выводить текстовую информацию на консоль, но сразу же создавать графический интерфейс программы. Точно так же, как и СУБД MySQL, операционная система Windows имеет программный интерфейс (Application Programming Interface, API). Однако в настоящий момент программы с использованием API Windows в полном объеме не создаются, а используются библиотеки, облегчающие написание программ и берущие на себя большинство рутинных операций. Исторически сложилось, что широкое распространение получили две таких библиотеки: MFC, которая развивается главным образом в рамках среды программирования Visual Studio, и VCL, которая развивается в рамках сред программирования Delphi и C++Builder. Сложность программирования на низком уровне для операционной системы Windows породила большое число коммерческих библиотек, которые позволяют, затрачивая незначительные усилия, достаточно легко реализовывать сложные программные проекты. СУБД MySQL не стала исключением, можно найти достаточно большое число библиотек, позволяющих взаимодействовать с СУБД из различных программных сред. Рассмотреть все их не представляется возможным, поэтому мы остановимся на библиотеке dbExpress для сред программирования Delphi

и C++Builder. К достоинствам этой библиотеки относится тот факт, что она представляет собой драйвер и работает напрямую с базой данных MySQL, поэтому это одна из самых быстрых библиотек для работы с MySQL на сегодняшний день. Кроме того, она позволяет работать с СУБД MySQL на уровне запросов: отправляется запрос, на который приходит ответ. В результате система обладает очень гибким программным интерфейсом, в отличие от других библиотек, которые сразу пытаются представить результирующую таблицу в виде графического элемента управления. К достоинствам библиотек и dbExpress относится также тот факт, что программы, построенные с использованием данной библиотеки, занимают меньший объем по сравнению с аналогами.

> ***Замечание***
>
> Библиотека dbExpress также обладает развитыми средствами представления результата, однако объем книги не позволяет полностью рассмотреть все тонкости этой библиотеки. В данном разделе будут описаны лишь простейшие приемы работы, которые, тем не менее, позволят создать реальные программы, взаимодействующие с MySQL-сервером.

При описании примеров данной главы будем ориентироваться на среду программирования C++Builder версии 6.0. В вариант Enterprise входит пакет dbExpress, однако перед тем как приступить к работе, необходимо загрузить и установить свежий драйвер для работы с MySQL с сайта компании CorelLab (**http://crlab.com/dbx/download.html**). Его потребуется установить на компьютере, где будет выполняться компилирование программ. Драйверы реализованы в виде динамически подключаемых библиотек, которые устанавливаются в каталог C:\Program Files\Borland\CBuilder6\Bin. В табл. 37.8 перечисляются имена динамических библиотек, соответствующих драйверам для различных баз данных. Для обеспечения работоспособности клиентской программы необходимо скопировать драйвер MySQL dbExpress в папку C:\Program Files\Borland\CBuilder6\Bin на компьютере, куда устанавливается программа.

Таблица 37.8. *Динамические библиотеки, соответствующие драйверам dbExpress*

База данных	Динамическая библиотека драйвера
Interbase dbExpress	dbexpint.dll
MySQL dbExpress	dbexpmysql.dll
Old MySQL dbExpress	dbexpmys.dll
DB2 dbExpress	dbexpdb2.dll
Oracle dbExpress	dbexpora.dll
dbExpress for Microsoft SQL Server	dbexpmss.dll
Informix dbExpress	dbexpinf.dll

Кроме этого, потребуются библиотеки libmySQL.dll и libmySQL.lib из дистрибутива MySQL, желательно версии 3.23.58, т. к. с библиотеками из более новых дистрибутивов

драйвер dbExpress может отказаться работать. Библиотеку libmySQL.dll необходимо будет помещать в каталог, где находится рабочая программа.

> ***Замечание***
>
> Столь старая библиотека требуется из-за того, что в более новых версиях изменился API C, и это приводит к конфликтам со старым клиентским кодом. Причем эта проблема характерна как для операционной системы Windows, так и Linux. Поэтому лучше не копировать библиотеку libmySQL.dll в системный каталог C:\Windows\system32\, чтобы не вызывать конфликты версий с другими программами, осуществляющими взаимодействие с СУБД MySQL, которые могут использовать библиотеку другой версии. Библиотеки libmySQL.dll и libmySQL.lib версии 3.23.58 можно найти на компакт-диске, поставляемом вместе с книгой, в проектах данной главы.

Страница **dbExpress** панели компонентов C++Builder выглядит так, как это представлено на рис. 37.1.



Рис. 37.1. Страница **dbExpress** панели компонентов C++Builder

Среди компонентов, расположенных на панели, имеются следующие:

- `SQLConnection` — компонент устанавливает соединение с базой данных;
- `SQLDataSet` — компонент предоставляет доступ к результирующей таблице;
- `SQLQuery` — позволяет выполнять SQL-запрос и получать результат;
- `SQLStoredProc` — компонент позволяет вызвать хранимую процедуру сервера;
- `SQLTable` — компонент позволяет получить доступ к содержимому таблицы без оформления SQL-запросов при помощи компонента `SQLQuery`;
- `SQLMonitor` — перехватывает сообщения, которыми обмениваются компонент `SQLConnection` и сервер базы данных, и помещает их в массив строк. Это позволяет вести собственные журнальные файлы;
- `SQLClientDataSet` — при помощи этого компонента можно получить доступ к данным, расположенным в кэше.

Доступ к базе данных осуществляется при помощи компонента `SQLConnection`, который необходимо разместить на форме (рис. 37.2).

После того как компонент будет помещен на форму и выбран, инспектор объектов (**Object Inspector**) выглядит так, как это представлено на рис. 37.3.

Теперь необходимо настроить компонент и выбрать драйвер MySQL. Список установленных драйверов располагается в файле C:\Program Files\Common Files\Borland Shared\DBExpress\dbxdrivers.ini. Если драйвер для MySQL от компании CoreLab уже установлен, то содержимое файла выглядит так, как это представлено в листинге 37.21.

Для удобства, записи, соответствующие драйверам для баз данных, отличных от MySQL, исключены из листинга 37.21.

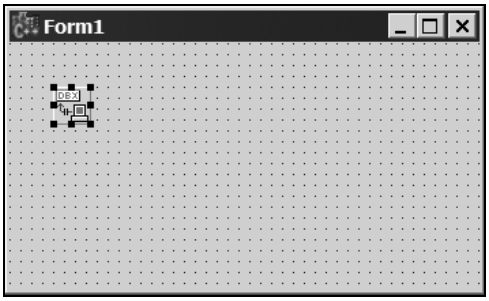


Рис. 37.2. Компонент `SQLConnection` на главной форме приложения

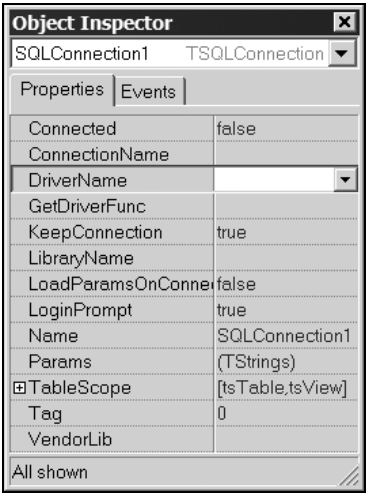


Рис. 37.3. Содержимое инспектора объектов для только что добавленного компонента `SQLConnection`

Замечание

Данный файл может понадобиться на машинах, где будет устанавливаться программа, осуществляющая соединение с сервером MySQL.

Листинг 37.21. Содержимое файла dbxdrivers.ini

```
[Installed Drivers]
DB2=1
Interbase=1
MySQL=1
```

```
Oracle=1
Informix=1
MySQL (Core Lab)=1
MySQL Direct (Core Lab)=1
MySQL Embedded (Core Lab)=1

...

[MySQL]
GetDriverFunc=getSQLDriverMYSQL
LibraryName=dbexpmysql.dll
VendorLib=libmysql.dll
HostName=ServerName
Database=DBNAME
User_Name=user
Password=password
BlobSize=-1
ErrorResourceFile=
LocaleCode=0000

...

[MySQL (Core Lab)]
GetDriverFunc=getSQLDriverMySQL
LibraryName=dbexpmda.dll
VendorLib=libmysql.dll
BlobSize=-1
HostName=
DataBase=
User_Name=
Password=
[MySQL Direct (Core Lab)]
GetDriverFunc=getSQLDriverMySQLDirect
LibraryName=dbexpmda.dll
VendorLib=not used
BlobSize=-1
HostName=
DataBase=
User_Name=
```

```
Password=
[MySQL Embedded (Core Lab)]
GetDriverFunc=getSQLDriverMySQLEmbedded
LibraryName=dbexpmda.dll
VendorLib=libmysqld.dll
BlobSize=-1
HostName=
DataBase=
User_Name=
Password=
```

В секции `[Installed Drivers]` указывается, какие драйверы установлены на текущем компьютере, а в остальных секциях описываются параметры доступа по умолчанию. Файл dbxdrivers.ini изменять не следует. Параметры соединения каждого из драйверов dbExpress описываются в файле C:\Program Files\Common Files\Borland Shared\DBExpress\dbxconnections.ini (листинг 37.22). Для удобства, записи, соответствующие драйверам для баз данных, отличных от MySQL, исключены из листинга 37.22.

Листинг 37.22. Содержимое файла dbxconnections.ini

```
...
[MySQLConnection]
DriverName=MySQL
HostName=ServerName
Database=DBNAME
User_Name=user
Password=password
BlobSize=-1
ErrorResourceFile=
LocaleCode=0000

...

[MySQL (Core Lab)]
BlobSize=-1
HostName=
DataBase=
DriverName=MySQL (Core Lab)
User_Name=
Password=
```

```
FetchAll=True
[MySQL Direct (Core Lab)]
BlobSize=-1
HostName=
DataBase=
DriverName=MySQL Direct (Core Lab)
User_Name=
Password=
FetchAll=True
[MySQL Embedded (Core Lab)]
BlobSize=-1
HostName=
DataBase=
DriverName=MySQL Embedded (Core Lab)
User_Name=
Password=
FetchAll=True
```

Редактировать данный файл вручную нет необходимости, достаточно произвести двойной щелчок мыши по компоненту `SQLConnection`, как будет выведена форма для редактирования параметров соединения (рис. 37.4).

Параметры соединения, перечисленные в форме на рис. 37.4, описаны в табл. 37.9.

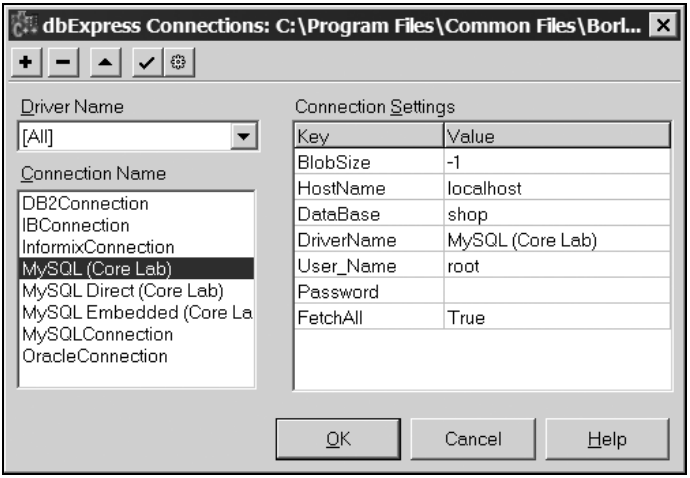


Рис. 37.4. Редактирование файла dbxconnections.ini

Таблица 37.9. *Параметры соединения с сервером базы данных*

Параметр	Значение
`BlobSize`	Ограничение на размер пакета данных. Если вводится значение –1, следовательно, ограничения отсутствуют
`DataBase`	Имя базы данных по умолчанию
`DriverName`	Название драйвера базы данных
`ErrorResourceFile`	Путь к файлу, в который помещаются сообщения об ошибках (не используется в данном драйвере)
`LocaleCode`	Код локализации, определяющий влияние национальных символов на сортировку данных (не используется в данном драйвере)
`Password`	Пароль для доступа к базе данных
`User_Name`	MySQL-пользователь, от имени которого осуществляется доступ к базе данных

После нажатия кнопки **OK** в форме, приведенной на рис. 37.4, в инспектор объектов автоматически будут добавлены параметры соединения с сервером MySQL (рис. 37.5).

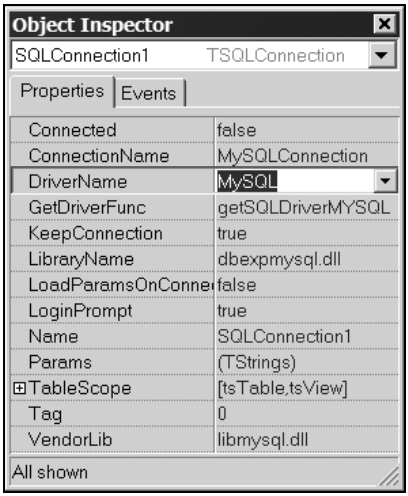


Рис. 37.5. Настроенное окно инспектора объектов

Параметры соединения с базой данных можно редактировать в поле `Params`.

В форме редактирования файла dbxconnections.ini (см. рис. 37.4) можно проверить успешность или неуспешность соединения с базой данных. Для этого следует нажать кнопку . Если соединение установлено успешно, то будут запрошены имя пользователя и пароль (рис. 37.6).

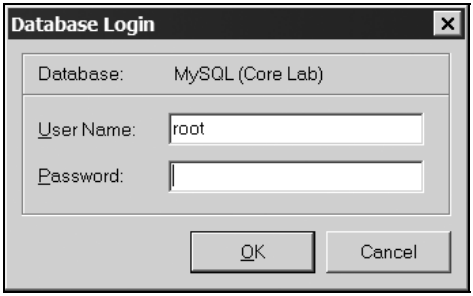


Рис. 37.6. Запрос имени пользователя и его пароля для доступа к серверу MySQL

Если введены корректное имя пользователя и пароль, будет сообщено об успешной установке соединения (рис. 37.7).



Рис. 37.7. Соединение успешно установлено

Если возникнет какая-нибудь ошибка, например, в каталоге проекта будут отсутствовать библиотеки libmySQL.dll и libmySQL.lib той версии, с которой сможет работать текущий драйвер, то появится соответствующее сообщение.

По умолчанию компонент `SQLConnection` не осуществляет попыток установить соединение с базой данных. Для того чтобы соединение было установлено при старте программы, необходимо в инспекторе объектов установить поле `Connected` в значение `true` (см. рис. 37.5). Впрочем, это всегда можно сделать программным способом (листинг 37.23).

Листинг 37.23. Установка соединения с базой данных

```
SQLConnection1->Connected = true;
```

Здесь `SQLConnection1` — имя компонента `SQLConnection` на форме, его можно изменить в поле `Name` инспектора объектов. До того момента, как поле `Connected` компонента не будет установлено в `true`, никакие операции с MySQL-сервером производить нельзя.

При попытке установить соединение с базой данных каждый раз будет производиться запрос имени пользователя и пароля так, как это представлено на рис. 37.6. Для того

чтобы избежать этого, необходимо установить поле `LoginPrompt` компонента `SQLConnection` в `false` либо в инспекторе объектов, либо программно (листинг 37.24).

Листинг 37.24. Подавление запроса на ввод имени пользователя и пароля

```
SQLConnection1->LoginPrompt = false;
```

После того как вывод формы авторизации на MySQL-сервере подавлен, имя MySQL-пользователя и его пароль будут извлекаться из файла dbxconnections.ini.

Компонент обладает пятью событиями, каждому из которых можно назначить обработчик (рис. 37.8).

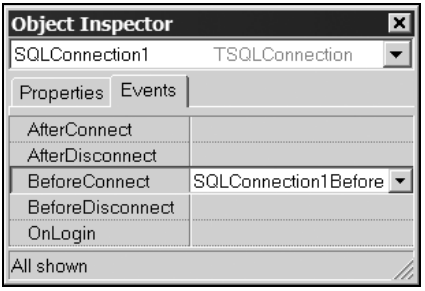


Рис. 37.8. События компонента `SQLConnection`

Всего компонент `SQLConnection` может реагировать на пять следующих событий:

- `AfterConnect` — событие, которое возникает непосредственно после того, как соединение с MySQL-сервером установлено. Это событие удобно использовать для настройки кодировки соединения при помощи оператора `SET NAMES` *(см. главу 14)*;
- `AfterDisconnect` — событие, которое возникает непосредственно после разрыва соединения с MySQL-сервером;
- `BeforeConnect` — событие, которое возникает непосредственно перед установкой соединения. Это событие удобно использовать для настройки параметров соединения;
- `BeforeDisconnect` — событие, которое возникает непосредственно после разрыва соединения с MySQL-сервером;
- `OnLogin` — событие, возникающее в момент установки соединения с MySQL-сервером.

Осуществив двойной щелчок по событию в окне инспектора объектов, можно перейти к редактированию обработчика данного события. В листинге 37.25 приводится пример обработки события `BeforeConnect`.

Листинг 37.25. Обработчик события `BeforeConnect`

```
void __fastcall TForm1::SQLConnection1BeforeConnect(TObject *Sender)
{
  // Если имя пользователя и пароль не запрашиваются,
  // устанавливаем их самостоятельно
  if(SQLConnection1->LoginPrompt == false)
  {
    SQLConnection1->Params->Values["User_Name"] = "wet";
    SQLConnection1->Params->Values["Password"] = "";
  }
}
```

Обработчик события `BeforeConnect` из листинга 37.25 позволит установить соединение с сервером MySQL, используя учетную запись `wet`, а не заданную по умолчанию учетную запись `root`.

Создадим небольшое приложение, которое будет устанавливать соединение с сервером MySQL при нажатии кнопки и закрывать его при повторном нажатии этой кнопки (листинг 37.26). Форма приложения будет содержать кнопку `btnMySQLConnect` и текстовую надпись `lblStatus`.

> ***Замечание***
>
> Проект, представленный в листинге 37.26, можно найти на компакт-диске, прилагаемом к данной книге.

Листинг 37.26. Управление соединением с MySQL-сервером

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "main.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TMySQL *MySQL;
//---------------------------------------------------------------------
__fastcall TMySQL::TMySQL(TComponent* Owner)
        : TForm(Owner)
{
  UpdateForm();
}
```

```
//---------------------------------------------------------------------
void __fastcall TMySQL::btnMySQLConnectClick(TObject *Sender)
{
  // Проверяем статус соединения
  if(SQLConnection1->ConnectionState == csStateClosed)
  {
    SQLConnection1->Connected = true;
  }
  else
  {
    SQLConnection1->Connected = false;
  }
  UpdateForm();
}
//---------------------------------------------------------------------
void TMySQL::UpdateForm(void)
{
  // Проверяем статус соединения
  if(SQLConnection1->ConnectionState == csStateClosed)
  {
    // Название кнопки
    btnMySQLConnect->Caption = "Установить";
    // Текст надписи
    lblStatus->Caption = "Соединение разорвано";
  }
  else
  {
    // Название кнопки
    btnMySQLConnect->Caption = "Разорвать";
    // Текст надписи
    lblStatus->Caption = "Соединение установлено";
  }
}
```

Результат работы программы из листинга 37.26 может выглядеть так, как это представлено на рис. 37.9.

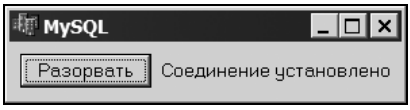


Рис. 37.9. Установка соединения с MySQL-сервером

Проверка статуса соединения с сервером осуществляется в методе `UpdateForm()`. Если соединение не установлено, кнопке `btnMySQLConnect` присваивается название "Установить", а статусной надписи `lblStatus` — "Соединение разорвано". В противном случае кнопке `btnMySQLConnect` присваивается название "Разорвать", а статусной надписи `lblStatus` — "Соединение установлено". Метод `UpdateForm()` вызывается как в конструкторе формы, так и в обработчике события нажатия кнопки `btnMySQLConnect`. В методе `UpdateForm()` и в обработчике кнопки проверка состояния соединения осуществляется при помощи свойства `ConnectionState`, которое может принимать следующие значения:

- `csStateClosed` — соединение закрыто;
- `csStateOpen` — соединение установлено;
- `csStateConnecting` — идет процесс установки соединения;
- `csStateExecuting` — ожидается исполнение переданного SQL-запроса;
- `csStateFetching` — идет процесс получения данных с сервера;
- `csStateDisconnection` — идет процесс завершения соединения.

Для того чтобы выполнять SQL-запросы и получать доступ к результирующим таблицам или таблицам баз данных, на форму потребуется поместить второй компонент из панели **dbExpress** — `SQLDataSet`. После того как на форму добавлен компонент `SQLDataSet`, окно инспектора объектов принимает вид, представленный на рис. 37.10.

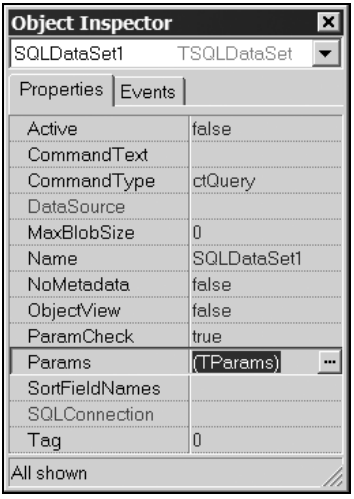


Рис. 37.10. Параметры компонента `SQLDataSet`

Параметр `CommandType` позволяет выбрать тип результирующей таблицы, с которой предстоит работать. Данный параметр принимает следующие значения:

- `ctQuery` — при выборе данного значения в свойстве `CommandText` указывается SQL-запрос;

- `ctTable` — при выборе данного значения в свойстве `CommandText` указывается имя таблицы текущей базы данных, при этом SQL-запрос, извлекающий содержимое таблицы, генерируется автоматически;
- `ctStoredProc` — в свойстве `CommandText` указывается имя хранимой процедуры.

Параметр `CommadType` лучше задать в инспекторе объектов, в то время как значение свойства `CommandText` лучше назначать программным путем. Это позволит более гибко работать с SQL-запросами.

Кроме этого, в инспекторе объектов следует связать компонент `SQLDataSet` с добавленным ранее компонентом `SQLConnection`. Для этого в выпадающем списке параметра `SQLConnection` следует выбрать имя соединения (в нашем случае `SQLConnection1`).

Модифицируем приложение из листинга 37.26, добавив в форму кнопку `btnShowVersion` и текстовую надпись `lblVersion`. Теперь, когда с сервером MySQL будет установлена связь, при нажатии кнопки `btnShowVersion` в надпись `lblVersion` будет помещаться версия MySQL-сервера. Обработчик кнопки представлен в листинге 37.27.

> ***Замечание***
>
> Полный проект приложения можно найти на компакт-диске, поставляемом вместе с книгой.

Листинг 37.27. Получение версии MySQL-сервера

```
void __fastcall TMySQL::btnShowVersionClick(TObject *Sender)
{
  // Формируем SQL-запрос
  SQLDataSet1->CommandText = "SELECT VERSION()";
  // Открываем результирующую таблицу
  SQLDataSet1->Open();
  // Получаем первую строку результирующей таблицы
  SQLDataSet1->First();
  // Получаем версию MySQL-сервера
  lblVersion->Caption =
              SQLDataSet1->Fields->FieldByName("VERSION()")->AsString;
  // Закрываем результирующую таблицу
  SQLDataSet1->Close();
}
```

Результат работы программы может выглядеть так, как это представлено на рис. 37.11.

Как видно из листинга 37.27, в свойство `CommandText` компонента `SQLDataSet` помещается SQL-запрос, запрашивающий версию MySQL-сервера: `"SELECT VERSION()"`. Свойство `CommandText` можно не заполнять, если SQL-запрос помещается в него в инспекторе объектов на стадии настройки компонента.

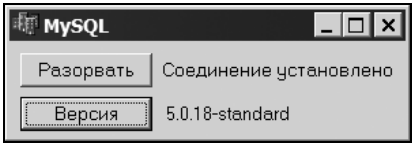


Рис. 37.11. Получение версии MySQL-сервера

Замечание

К столбцу с версией MySQL-сервера необходимо обращаться по имени функции, однако имя столбца в результирующей таблице можно изменить при помощи оператора `AS`. Так при использовании запроса `SELECT VERSION() AS ver` имя столбца в результирующей таблице будет не `VERSION()`, а `ver`.

После того как запрос задан, можно открыть таблицу при помощи метода `Open()`. Важно, чтобы в конце операции манипулирования результирующей таблицей она была закрыта, например, при помощи метода `Close()`, в противном случае нельзя будет использовать данный компонент для обработки следующего SQL-запроса. При помощи метода `First()` производится установка дескриптора результирующей таблицы на первую запись таблицы. После того как запись установлена, можно обращаться к свойству `Fields`, для того чтобы получить поля результирующей таблицы. Так как в случае запроса `SELECT VERSION()` только одно поле, то получить доступ к результату можно при помощи строки:

```
SQLDataSet1->Fields->FieldByName("VERSION()")->AsString;
```

Важно отметить, что результат необходимо преобразовать к строке при помощи свойства `AsString`, если же результат необходимо поместить в целочисленную переменную, то вместо `AsString` следует использовать `AsInteger`. Однако если преобразование в строку или число будет невозможно осуществить — будет сгенерированно соответствующее исключение.

Рассмотрим случай, когда результирующая таблица содержит не одну запись, а несколько записей, каждая из которых состоит из нескольких полей. Создадим приложение, которое будет получать содержимое таблицы `catalogs` и выводить его записи в таблице, для построения которой будет использоваться компонент `StringGird`. В листинге 37.28 представлен код такого приложения.

Замечание

Полный проект приложения можно найти на компакт-диске, поставляемом вместе с книгой.

Листинг 37.28. Вывод содержимого таблицы `catalogs`

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
```

```
#include "main.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TMySQL *MySQL;
//---------------------------------------------------------------------
__fastcall TMySQL::TMySQL(TComponent* Owner)
        : TForm(Owner)
{
  // Настраиваем кодировку соединения
  SQLDataSet1->CommandText = "SET NAMES 'cp1251'";
  SQLDataSet1->ExecSQL();

  ////////////////////////////////////////////
  // Возвращаем число записей в таблице catalogs
  ////////////////////////////////////////////
  SQLDataSet1->CommandText = "SELECT * FROM catalogs";
  int total = SQLDataSet1->RecordCount;

  ////////////////////////////////////////////
  // Формируем таблицу
  ////////////////////////////////////////////
  // Выставляем число строк таблицы
  Grid->RowCount = total + 1;
  // Формируем "шапку таблицы"
  Grid->Cells[0][0] = "id_catalog";
  Grid->ColWidths[0] = 100;
  Grid->Cells[1][0] = "name";
  Grid->ColWidths[1] = 200;
  // Выставляем размеры и положение таблицы
  Grid->Width = Grid->ColWidths[0] + Grid->ColWidths[1] + 8;
  Grid->Height = Grid->DefaultRowHeight*(total + 1) + total + 4;
  Grid->Top = 8;
  Grid->Left = 8;
  // Выставляем размер окна программы
  MySQL->Width = 8*2 + Grid->Width;
  MySQL->Height = 8*6 + Grid->Height;

  ////////////////////////////////////////////
  // Получаем результирующую таблицу для catalogs
```

```
  ////////////////////////////////////////////
  SQLDataSet1->Open();
  SQLDataSet1->First();
  for (int j = 1; j < total + 1; ++j)
  {
    // Заполняем таблицу
    Grid->Cells[0][j] =
             SQLDataSet1->Fields->FieldByName("id_catalog")->AsString;
    Grid->Cells[1][j] =
                  SQLDataSet1->Fields->FieldByName("name")->AsString;
    SQLDataSet1->Next();
  }
  SQLDataSet1->Close();
}
```

Результат работы программы из листинга 37.28 представлен на рис. 37.12.

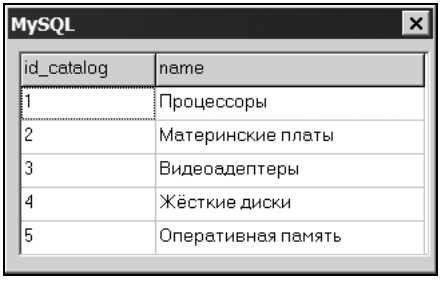


Рис. 37.12. Содержимое таблицы `catalogs`

Причем, если в таблицу `catalogs` добавляется еще одна запись, окно программы автоматически расширяется (рис. 37.13).

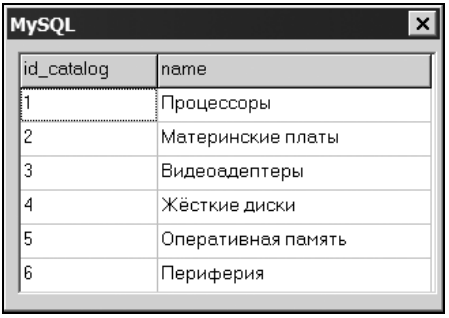


Рис. 37.13. Автоматическое расширение по высоте окна приложения

Как видно из листинга 37.28, программу можно условно поделить на три части. В первой части при помощи свойства `RecordCount` компонента `SQLDataSet` определяется количество записей в таблице `catalogs`, которое помещается в переменную `total`. Во второй части, используя полученное количество записей `total`, формируется таблица и подгоняются параметры окна приложения. В третьей, заключительной части программы поля таблицы в форме заполняются содержимым таблицы `catalogs`. Следует обратить внимание на тот факт, что для перехода к следующей записи результирующей таблицы используется метод `Next()` компонента `SQLDataSet` — если его не применять, дескриптор результирующей таблицы будет оставаться на первой записи при каждой итерации цикла.

> ***Замечание***
>
> Программу из листинга 37.28 можно было оформить альтернативным образом. Для этого свойству `CommandType` можно было присвоить значение `ctTable`, а `CommandText` — значение `catalogs`. В этом случае SQL-запрос был бы сформирован автоматически.

Для настройки кодировки соединения используется SQL-запрос `SET NAMES 'cp1251'`, в противном случае вместо русского текста будут возвращаться знаки вопросов. Следует отметить, что запрос выполняется при помощи метода `ExecSQL()`. Данный метод можно использовать со всеми SQL-запросами, выполнение которых не требует получения и анализа результирующей таблицы (`INSERT`, `UPDATE`, `DELETE`, `CREATE TABLE` и т. п.).

Число записей в таблице может быть значительным и вывод всех их на одной форме может оказаться затруднительным. В этом случае прибегают к постраничной навигации.

> ***Замечание***
>
> Многие компоненты сред разработки C++Builder и Delphi для работы с базами данных имеют встроенные панели постраничной навигации, но т. к. драйверы dbExpress имеют только однонаправленный курсор, для реализации постраничной навигации потребуется самостоятельное программирование логики приложения.

Рассмотрим постраничную навигацию на примере вывода таблицы `products` по 5 позиций на одной странице. Для этого добавим в форму будущего приложения четыре кнопки, первая из которых возвращает в начало результирующей таблицы, вторая кнопка возвращает на предыдущую страницу, третья — на следующую, а четвертая — на последнюю (рис. 37.14).

Приложение снабдим статусной панелью внизу главного окна, в которой будет указываться, какие позиции выводятся в настоящий момент. В основе постраничной навигации будут лежать три значения:

- `total` — общее число записей в таблице;
- `current_pos` — текущая позиция, начиная с которой будет осуществляться вывод записей;
- `pnumber` — число позиций на одной странице (в нашем случае эта величина равна 5).

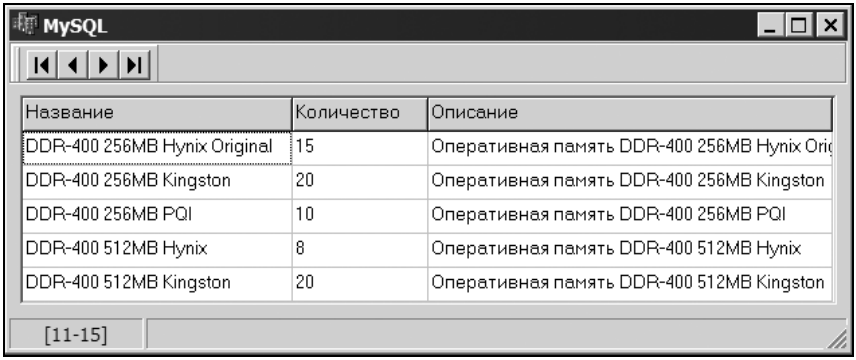


Рис. 37.14. Постраничная навигация

Код приложения, осуществляющего постраничную навигацию, может выглядеть так, как это представлено в листинге 37.29.

Замечание

Полный проект приложения можно найти на компакт-диске, поставляемом вместе с книгой.

Листинг 37.29. Постраничная навигация

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "main.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TMySQL *MySQL;
//---------------------------------------------------------------------
__fastcall TMySQL::TMySQL(TComponent* Owner)
        : TForm(Owner)
{
  // Настраиваем кодировку соединения
  SQLDataSet1->CommandText = "SET NAMES 'cp1251'";
  SQLDataSet1->ExecSQL();

  ////////////////////////////////////////////
  // Возвращаем число записей в таблице catalogs
```

```
  ////////////////////////////////////////////
  SQLDataSet1->CommandText = "SELECT COUNT(*) AS total FROM products";
  SQLDataSet1->Open();
  SQLDataSet1->First();
  total = SQLDataSet1->Fields->FieldByName("total")->AsInteger;
  SQLDataSet1->Close();
  // Текущая позиция
  current_pos = 0;
  // Число позиций на одной странице
  pnumber = 5;

  ////////////////////////////////////////////
  // Формируем таблицу
  ////////////////////////////////////////////
  // Выставляем число строк таблицы
  Grid->RowCount = 5 + 1;
  // Формируем "шапку таблицы"
  Grid->Cells[0][0] = "Название";
  Grid->ColWidths[0] = 200;
  Grid->Cells[1][0] = "Количество";
  Grid->ColWidths[1] = 100;
  Grid->Cells[2][0] = "Описание";
  Grid->ColWidths[2] = 300;

  // Обновляем содержимое формы
  UpdateForm();
}
//---------------------------------------------------------------------
void TMySQL::UpdateForm(void)
{
  ////////////////////////////////////////////
  // Получаем результирующую таблицу для catalogs
  ////////////////////////////////////////////
  SQLDataSet1->CommandText =
                          String("SELECT name, count, description ") +
                             "FROM products " +
                             "ORDER BY name " +
                             "LIMIT " + String(current_pos) + "," +
                                        String(pnumber);
```

```
  SQLDataSet1->Open();
  SQLDataSet1->First();
  for (int j = 1; j < 5 + 1; ++j)
  {
    // Заполняем таблицу
    Grid->Cells[0][j] =
                  SQLDataSet1->Fields->FieldByName("name")->AsString;
    Grid->Cells[1][j] =
                 SQLDataSet1->Fields->FieldByName("count")->AsString;
    Grid->Cells[2][j] =
           SQLDataSet1->Fields->FieldByName("description")->AsString;
    SQLDataSet1->Next();
  }
  SQLDataSet1->Close();

  // Выводим в панель статуса текущую позицию
  StatusBar->Panels->Items[0]->Text = String("[") +
                                      String(current_pos + 1) +
                                      "-" +
                                      String(current_pos + pnumber) +
                                       "]";
}
//---------------------------------------------------------------------
void __fastcall TMySQL::btnFirstClick(TObject *Sender)
{
  // Перемещаем указатель в начало
  current_pos = 0;
  // Обновляем содержимое формы
  UpdateForm();
}
//---------------------------------------------------------------------
void __fastcall TMySQL::btnPreviewClick(TObject *Sender)
{
  if(current_pos - pnumber < 0) return;
  else
  {
    // Уменьшаем текущую позицию
    current_pos -= pnumber;
    // Обновляем содержимое формы
```

```
    UpdateForm();
  }
}
//---------------------------------------------------------------------
void __fastcall TMySQL::btnNextClick(TObject *Sender)
{
  if(current_pos + pnumber >= total) return;
  else
  {
    // Увеличиваем текущую позицию
    current_pos += pnumber;
    // Обновляем содержимое формы
    UpdateForm();
  }
}
//---------------------------------------------------------------------
void __fastcall TMySQL::btnLastClick(TObject *Sender)
{
  if(total <= pnumber) return;
  else
  {
    // Перемещаем указатель в конец
    if(fmod((double)total, (double)pnumber)>0.001)
    {
      current_pos = (total/pnumber)*pnumber;
    }
    else
    {
      current_pos = (total/pnumber - 1)*pnumber;
    }
    // Обновляем содержимое формы
    UpdateForm();
  }
}
```

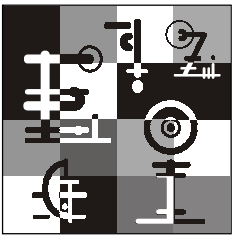
В конструкторе формы устанавливается кодировка соединения и инициализируются переменные `total`, `current_pos` и `pnumber`, а также осуществляется настройка таблицы.

За обновление содержимого таблицы несет ответственность функция `UpdateForm()`, которая в зависимости от значения `current_pos` выводит часть таблицы `products`.

Для этого используется конструкция `LIMIT`, подробное описание которой приводится в *главе 7*.

Перечисленные далее функции являются обработчиками навигационных кнопок и имеют следующее назначение:

- `btnFirstClick()` — устанавливает текущую позицию `current_pos` в начало;
- `btnNextClick()` — устанавливает текущую позицию `current_pos` на предыдущую страницу;
- `btnLastClick()` — устанавливает текущую позицию `current_pos` на следующую страницу;
- `btnLastClick()` — устанавливает текущую позицию `current_pos` на последнюю страницу.



Глава 38

Взаимодействие MySQL и Perl

Perl является интерпретируемым языком программирования, разработанным первоначально Ларри Уоллом для упрощения генерации отчетов в операционной системе UNIX. Ларри Уолл по образованию лингвист, и, разрабатывая новый язык программирования, одной из целей он ставил создание языка, чувствительного к контексту, т. е. поведение языка зависит от контекста, в котором вызывается та или иная конструкция. Это делает язык чрезвычайно выразительным, сложным и не похожим на другие языки программирования. Perl является стандартом де-факто в качестве языка для написания системных скриптов в операционной системе UNIX. Ни один дистрибутив UNIX не обходится без Perl, многие из них просто не смогут без него функционировать. Даже проверочные и тестовые скрипты для MySQL разработаны при помощи Perl. Долгое время Perl применялся для разработки CGI-скриптов для Web, пока эту нишу не заполнили более молодые и более приспособленные для этого технологии: Java, ASP.NET, PHP и т. д.

> ***Замечание***
>
> Данная глава не рассматривает разнообразный синтаксис языка Perl и не может служить основой для его изучения. Наилучшей книгой по изучению этого интереснейшего языка программирования является знаменитая на весь мир Camel-Book ("Верблюжья книга", Perl в переводе с английского — "колючка") "Программирование на Perl" Ларри Уолла, Тома Кристиансуна и Джона Орванта.[1]

Perl обладает базовыми и дополнительными возможностями, последние можно подключить при помощи модулей. Так Perl взаимодействует с MySQL при помощи модуля DBI (Database Interface), который предоставляет единый интерфейс для доступа к различным базам данных. То есть код, разработанный для одной базы данных, с минимальными исправлениями может работать с другой базой данных — DBI и драйверы баз данных сами позаботятся о передаче запроса соответствующей базе данных.

---

[1] Уолл Л., Кристиансен Т., Орвант Д. Программирование на Perl. — СПб.: Символ-плюс, 2004. — 1152 с.

Владельцы UNIX-подобной операционной системы не встретят затруднений с установкой Perl, он, скорее всего, уже присутствует в системе. Единственное, что может потребоваться, — это установить модуль DBI и DBD-драйвер для СУБД MySQL.

Владельцам операционной системы Windows придется обзавестись локализованной версией Perl. Одной из таких версий является ActivePerl, загрузить который можно с сайта **http://www.activestate.com**. После того как Perl будет установлен, необходимо установить модуль DBI при помощи утилиты `ppm` (Perl Package Manager). Для этого следует в командной строке перейти в каталог C:\Perl\bin и выполнить команды, представленные в листинге 38.1.

> ***Замечание***
>
> Если во время установки связать расширение pl с интерпретатором Perl, запускать Perl-скрипты можно будет двойным щелчком мыши или при помощи клавиши <Enter>.

Листинг 38.1. Установка модуля DBI и DBD-драйвера MySQL

```
C:\Perl\bin>ppm
ppm> install DBI
ppm> install DBD-mysql
```

Утилита `ppm` загрузит необходимые файлы с сайта **http://www.activestate.com** и установит их.

> ***Замечание***
>
> Все коды, приведенные в данной главе, можно найти на компакт-диске, прикладываемом к данной книге, в каталоге \code\38.

В качестве демонстрации приемов работы с СУБД MySQL в Perl рассмотрим скрипт, выводящий список элементов каталога из таблицы `catalogs` учебной базы данных `shop` (листинг 38.2).

Листинг 38.2. Извлечение содержимого таблицы `catalogs`

```
#!perl -w
use strict;
use DBI;

my ($con, $res);
# Устанавливаем соединение с базой данных
$con = DBI->connect("DBI:mysql:host=localhost;database=shop",
                    "root",
                    "",
```

```
                    {PrintError => 0, RaiseError => 1});
# Устанавливаем кодировку соединения (Windows-1251)
$con->do("SET NAMES 'cp1251'");

# Извлекаем содержимое таблицы catalogs
$res = $con->prepare("SELECT * FROM catalogs");
# Выполняем SQL-запрос
$res->execute();

# Выводим результаты запроса
while(my @val = $res->fetchrow_array())
{
  printf("%s, %s\n", $val[0], $val[1]);
}

# Освобождаем память, отведенную под результирующую таблицу
$res->finish();
# Разрываем соединение с MySQL-сервером
$con->disconnect();

# Завершаем работу скрипта
exit(0);
```

Результат работы программы из листинга 38.2 может выглядеть так:

```
C:\>connect.pl
1, Процессоры
2, Материнские платы
3, Видеоадаптеры
4, Жесткие диски
5, Оперативная память
```

Рассмотрим более подробно каждую строку данного скрипта. В начале скрипта размещается последовательность `#!`. Символ диеза (`#`) в Perl является комментарием, но если сразу за ним следует восклицательный знак, такой комментарий приобретает специальное значение — он указывает обработчик содержимого файла. Дело в том, что в UNIX-подобных операционных системах скрипты пишутся на нескольких языках: это и Perl, и язык оболочек, реже PHP или Python. Операционная система Windows при обработке файлов ориентируется на расширение файла, UNIX-подобные операционные системы на расширение файла, как правило, не ориентируются — оно зачастую отсутствует. Система читает первую строку и ищет его обработчик. Напри-

мер, `#!/usr/bin/php`, `#!/usr/bin/sh` или `#!/usr/bin/perl`, кроме того, интерпретаторы могут находиться не обязательно в `/usr/bin` — при помощи такого комментария можно уточнить путь и передать параметры интерпретатору.

> ***Замечание***
>
> В английском варианте последовательность `#!` часто называют bang line, а также hash-bang или she-bang.

Windows не проверяет содержимое файлов на наличие `#!` и ориентируется строго на расширение файла. Поэтому указать путь `#!`, присвоив файлу произвольное расширение, не получится. Однако в случае ActivePerl можно все равно указать последовательность `#!perl` — изменить путь к интерпретатору не получится, зато появляется возможность передать параметры, например, параметр `-w`, требующий от интерпретатора выводить все предупреждения, что может быть полезным при отладке программ.

Строка `use strict` переводит интерпретатор Perl в особый режим, при котором необходимо, чтобы любая переменная перед использованием была объявлена. Дело в том, что Perl допускает использование необъявленных переменных, что заметно сокращает объем скрипта, но служит причиной многочисленных трудно улавливаемых ошибок, когда опечатка в имени переменной приводит к тому, что формируется новая переменная с пустым значением.

Для того чтобы иметь возможность использовать функции модуля DBI, модуль подключается в начале любого скрипта при помощи оператора `use DBI`. Интерфейс библиотеки DBI выполнен в объектно-ориентированном стиле. Так для выполнения операций используются объекты и классы. Например, для подключения к базе данных применяется метод `connect()` класса `DBI`:

```
# Устанавливаем соединение с базой данных
$con = DBI->connect("DBI:mysql:host=localhost;database=shop",
                    "root",
                    "",
                    {PrintError => 0, RaiseError => 1});
```

В качестве первого параметра метода `connect()` передается строка, которую часто называют *источником данных*. Первая подстрока `DBI` сообщает метод доступа, вторая `mysql` указывает драйвер, который используется для установки соединения с базой данных. Так, если потребуется установить соединение с СУБД PostgreSQL, то вместо `mysql` следует указать подстроку `Pg`. Следующая подстрока `host=localhost` указывает адрес сервера базы данных, а подстрока `database=shop` — имя базы данных по умолчанию.

В качестве второго параметра метода `connect()` выступает имя пользователя базы данных (в данном случае `root`), а в качестве третьего — пароль пользователя.

Последний параметр метода `connect()` позволяет установить дополнительные параметры модуля DBI, в частности, отключить параметр `PrintError` и включить пара-

метр RaiseError. Если атрибут RaiseError принимает ненулевое значение, включается режим, при котором возникновение ошибки приводит к выводу сообщения и остановке работы программы. Если атрибут PrintError принимает ненулевое значение, возникновение ошибки приводит к выводу предупреждения, но программа продолжает работать.

При успешном выполнении соединения метод connect() возвращает дескриптор соединения $con, в противном случае возвращается значение undef. Далее устанавливается кодировка соединения (cp1251), для чего используется оператор SET NAMES *(см. главу 14)*. Так как оператор не возвращает результирующей таблицы, то можно воспользоваться методом do() дескриптора соединения $con:

```
# Устанавливаем кодировку соединения (Windows-1251)
$con->do("SET NAMES 'cp1251'");
```

> ***Замечание***
>
> Метод do() возвращает количество обработанных записей, –1, если число обработанных записей невозможно определить (например, выполняется многотабличный оператор DELETE или UPDATE), и undef при возникновении ошибочной ситуации.

Если не установить кодировку по умолчанию, то русский текст будет выводиться в виде знаков вопроса:

```
C:\>connect.pl
1, ??????????
2, ?????????? ?????
3, ?????????????
4, ??????? ?????
5, ?????????? ??????
```

Однако SQL-запрос SELECT * FROM catalogs возвращает результирующую таблицу, и для того чтобы получить к ней доступ, выполнения этого запроса при помощи метода do() недостаточно. Метод prepare() дескриптора $con позволяет задать SQL-запрос. Этот метод возвращает дескриптор запроса $res, при помощи метода execute() которого можно отправить SQL-запрос на выполнение серверу базы данных.

```
# Извлекаем содержимое таблицы catalogs
$res = $con->prepare("SELECT * FROM catalogs");
# Выполняем SQL-запрос
$res->execute();
```

> ***Замечание***
>
> Вместо конструкции $res->execute() допускается использование конструкции $res->execute.

Получить доступ к результирующей таблице, которую возвращает SELECT-запрос, можно при помощи метода fetchrow_array() объекта $res. Метод возвращает

поля текущей записи в виде массива. При первом вызове метод возвращает первую строку таблицы, при втором — вторую, и так до тех пор, пока не будут исчерпаны все записи результирующей таблицы. Как правило, для этого применяется цикл `while()`: как только записи в результирующей таблице заканчиваются, метод `fetchrow_array()` возвращает пустую строку, что рассматривается оператором `while()` как ложное условие, в результате чего цикл прекращает свою работу:

```
# Выводим результаты запроса
while(my @val = $res->fetchrow_array())
{
  printf("%s, %s\n", $val[0], $val[1]);
}
```

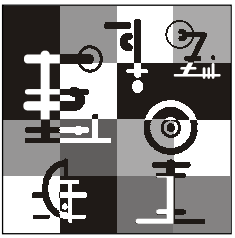
Освободить память, отводимую под результирующую таблицу, можно при помощи метода `finish()` дескриптора запроса `$res`:

```
# Освобождаем память, отведенную под результирующую таблицу
$res->finish();
```

Метод `disconnect()` позволяет разорвать соединение с базой данных:

```
# Разрываем соединение с MySQL-сервером
$con->disconnect();
```

Методы `finish()` и `disconnect()` являются лишними в данном контексте и приведены для полноты, т. к. далее программа завершает свою работу, и память, выделенная скрипту, будет возвращена в систему автоматически. Однако к этим методам часто прибегают в том случае, если скрипт выполняется длительное время (сутками), для того чтобы предотвратить утечки памяти.



Глава 39

Взаимодействие MySQL и PHP

Язык программирования PHP является сравнительно молодым языком программирования, специально созданным для генерации динамических Web-страниц. Так сложилось, что связка Web-сервера Apache, базы данных MySQL и языка программирование PHP получили очень широкое распространение среди Web-разработчиков.

> ***Замечание***
>
> Связка Apache, MySQL и PHP конкурирует с технологиями Java и ASP.NET. В Российской Федерации последние две технологии распространены слабо — большинство хост-провайдеров отдают предпочтение бесплатно распространяемым Apache, MySQL и PHP.

> ***Замечание***
>
> Для тестирования приведенных в данной главе примеров потребуется либо реальный сервер, арендуемый у хост-провайдера, либо локальный сервер, который можно настроить, следуя инструкциям в статье **http://www.softtime.ru/info/articlephp.php?id_article=24**.

Интерфейс PHP является почти точной копией интерфейса программирования C, однако благодаря тому, что PHP поддерживает ассоциативные массивы, не требует явной заботы о выделении и освобождении памяти и является интерпретируемым языком программирования, взаимодействовать с MySQL в нем гораздо проще.

> ***Замечание***
>
> Предполагается, что читатель знаком с основами PHP — данная глава не может рассматриваться как полноценное руководство по языку программирования.

39.1. Функция *mysql_connect()*

Соединение с базой данных MySQL осуществляется при помощи функции `mysql_connect()`, которая имеет следующий синтаксис:

```
resource mysql_connect ([string server [,
                         string username [,
```

```
string password [,
bool new_link [,
int 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.

> ***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`.

В листинге 39.1 приведен пример использования функции `mysql_connect()`.

Листинг 39.1. Функция `mysql_connect()`

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

  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
  if (!$dbcnx) // Если дескриптор равен 0, соединение не установлено
  {
```

```
    exit ("<P>В настоящий момент сервер базы данных не доступен, поэтому
          корректное отображение страницы невозможно.</P>");
  }
  else
  {
    echo "<P>Соединение установлено.</P>";
  }
?>
```

Замечание

Для подавления вывода сообщений об ошибках, генерируемых PHP в окно браузера, в листинге 39.1 перед функцией `mysql_connect()` размещен символ `@`.

Переменные `$dblocation`, `$dbuser` и `$dbpasswd` хранят имя сервера, имя пользователя и пароль и, как правило, прописываются в отдельном файле (к примеру, `config.php`), который потом вставляется в каждый PHP-файл. В последнем имеется код для работы с MySQL.

Начиная с MySQL версии 4.1, изменился порядок работы с кодировками — теперь перед началом работы необходимо выставить кодировки при помощи запроса `SET NAMES`, представленного в листинге 39.2. Иначе русский текст будет попадать в базу данных в виде знаков вопроса.

Замечание

Функция `mysql_query()` рассматривается более подробно в *разд. 39.4*.

Листинг 39.2. Установка кодировки

```
<?php
  mysql_query ("SET NAMES 'cp1251'");
?>
```

39.2. Функция *mysql_close()*

Закрытие соединения осуществляется при помощи функции `mysql_close()`. Ее синтаксис представлен ниже:

```
bool mysql_close ([resource 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 листинге 39.3 приведен пример использования функции.

Листинг 39.3. Закрытие соединения при помощи функции `mysql_close()`

```
<?php
  // Устанавливаем соединение с базой данных
  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
  if (!$dbcnx)
  {
    // Выводим предупреждение
    exit ("<P>В настоящий момент сервер базы данных не доступен, поэтому
    корректное отображение страницы невозможно.</P>");
  }
  if(mysql_close($dbcnx)) // разрываем соединение
  {
    echo("Соединение с базой данных прекращено");
  }
  else
  {
    echo("Не удалось завершить соединение");
  }
?>
```

39.3. Функция *mysql_select_db()*

Использование функции `mysql_select_db()` эквивалентно вызову команды `USE` в SQL-запросе, т. е. функция `mysql_select_db()` выбирает базу данных для дальнейшей работы, и все последующие SQL-запросы применяются к выбранной базе данных.

```
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*. Функция возвращает `TRUE` при успешном выполнении операции и `FALSE` в противном случае. В листинге 39.4 приводится пример использования данной функции.

Листинг 39.4. Использование функции `mysql_select_db()`

```
<?php
  // Код соединения с базой данных
  if (!@mysql_select_db($dbname, $dbcnx))
  {
    echo( "<P>В настоящий момент база данных не доступна, поэтому
```

```
                корректное отображение страницы невозможно.</P>" );
    exit();
  }
?>
```

Имеет смысл помещать функции для соединения и выбора базы данных в тот же файл (`config.php`), где объявлены переменные с именами сервера, пользователя и паролем (листинг 39.5), и подключать данный файл при помощи конструкции `include_once()` или `require_once()`, если требуется обращение к базе данных MySQL.

Листинг 39.5. Файл config.php

```
<?php
  // Адерс сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных на хостинге или локальной машине
  $dbname = "shop";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";

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

Код установки соединения, представленный в листинге 39.5, ориентирован на СУБД MySQL версии ниже 4.1. Для того чтобы настроить кодировку соединения (например, cp1251) для СУБД MySQL версии выше 4.1, необходимо выполнить запрос `SET NAMES 'cp1251'`. В листинге 39.6 приводится пример универсального файла

config.php, который запрашивает версию текущего сервера MySQL и в зависимости от полученного результата либо настраивает кодировку соединения, либо нет.

Листинг 39.6. Универсальный файл config.php

```
<?php
  // Адерс сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных на хостинге или локальной машине
  $dbname = "shop";
  // 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>" );
  }

  // Определяем версию сервера
  $query = "SELECT VERSION()";
  $ver = mysql_query($query);
  if(!$ver) exit("Ошибка при определении версии MySQL-сервера");
  $version = mysql_result($ver, 0);
  list($major, $minor) = explode(".", $version);
  // Если версия выше 4.1, сообщаем серверу, что будем работать
  // с кодировкой cp1251
  $ver = $major.".".$minor;
  if((float)$ver >= 4.1)
  {
    mysql_query("SET NAMES 'cp1251'");
  }
?>
```

39.4. Функция *mysql_query()*

Выполнение SQL-запросов осуществляется при помощи функции `mysql_query()`, которая имеет следующий синтаксис:

```
resource mysql_query (string query [, resource link_identifier])
```

Первый аргумент функции представляет собой строку с запросом *query*, второй (*link_identifier*) — дескриптор соединения, возвращаемый функцией `mysql_connect()`.

> ***Замечание***
>
> Если открытые соединения отсутствуют, функция пытается соединиться с СУБД, аналогично функции `mysql_connect()` без параметров.

> ***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. Если запрос не возвращает результирующей таблицы, можно не сохранять дескриптор в отдельной переменной. В листинге 39.7 представлен скрипт, создающий новую таблицу `catalogs`.

Листинг 39.7. Использование функции `mysql_query()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";
  // Формируем и выполняем SQL-запрос
  $query = "CREATE TABLE catalogs (
             id_catalog INT(11) NOT NULL AUTO_INCREMENT,
             name TINYTEXT NOT NULL,
             PRIMARY KEY (id_catalog))";
  if(mysql_query($query))
  {
     echo "Таблица создана успешно";
  }
  else
  {
     exit(mysql_erorr());
  }
?>
```

В листинге 39.7 при неудачном выполнении функции `mysql_query()`, функции `exit()`, останавливающей работу скрипта, в качестве параметра передается значение ошибки, возвращаемое функцией `mysql_erorr()`.

> ***Замечание***
>
> По аналогии с API C, PHP поддерживает также функцию `mysql_errno()`, которая возвращает номер ошибки.

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()`.

39.5. Функция *mysql_result()*

Первая функция `mysql_result()` возвращает результат запроса, выполненного функцией `mysql_query()`. С ее помощью можно получить доступ к отдельному полю записи. 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 39.8 при помощи данной функции выводится название первого каталога в таблице `catalogs` учебной базы данных `shop`.

Листинг 39.8. Использование функции `mysql_result()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT name FROM catalogs WHERE id_catalog = 1";
  // Выполняем SQL-запрос
  $cat = mysql_query($query);
  // Обрабатываем результаты запроса
  if($cat) echo mysql_result($cat, 0, 'name');
  else exit(mysql_error());
?>
```

Результат работы скрипта из листинга 39.8 представлен на рис. 39.1.

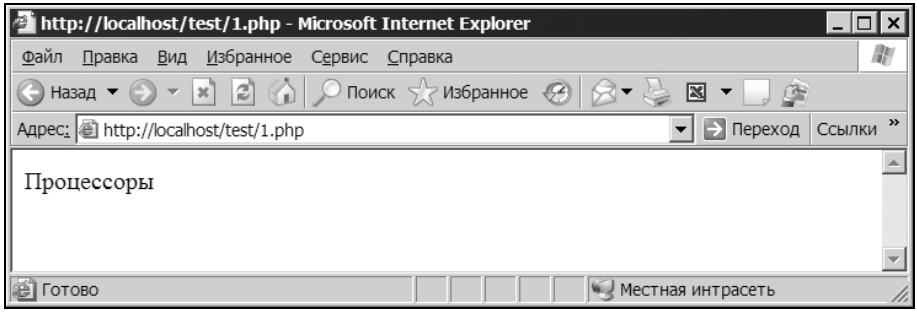


Рис. 39.1. Результат работы скрипта из листинга 39.8

39.6. Функция *mysql_fetch_row()*

Функция `mysql_fetch_row()` возвращает текущую запись в результирующей таблице в виде неассоциативного массива. Повторный вызов функции переводит курсор в результирующей таблице на следующую запись. Функция `mysql_fetch_row()` имеет следующий синтаксис:

```
array mysql_fetch_row (resource result)
```

В качестве единственного аргумента *result* функция принимает дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`. Функция возвращает массив, каждый элемент которого соответствует одному полю в записи, или `FALSE`, если курсор достиг конца результирующей таблицы. При работе с массивом, который возвращает функция `mysql_fetch_row()`, удобно использовать функцию `list()`, преобразующую элементы массива в переменные. В листинг 39.9 выводится HTML-таблица, в которой размещаются данные из таблицы `catalogs` учебной базы данных `shop`.

Листинг 39.9. Применение функции `mysql_fetch_row()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT * FROM catalogs";
  // Выполняем SQL-запрос
  $cat = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if(!$cat) exit(mysql_erorr());
  // Так как запрос может возвращать
  // несколько строк, применяем цикл
```

```
  echo "<table border=1>";
  while(list($id_author, $name) = mysql_fetch_row($cat))
  {
    echo "<tr>
            <td>$id_author</td>
            <td>$name</td>
          </tr>";
  }
  echo "</table>";
?>
```

Результат работы скрипта из листинга 39.9 представлен на рис. 39.2.



Рис. 39.2. Результат работы скрипта из листинга 39.9

39.7. Функция *mysql_fetch_assoc()*

Функция `mysql_fetch_assoc()` возвращает текущую запись в результирующей таблице в виде ассоциативного массива. Повторный вызов функции переводит курсор в результирующей таблице на следующую запись. Функция `mysql_fetch_assoc()` имеет следующий синтаксис:

```
array mysql_fetch_assoc (resource result)
```

В качестве единственного аргумента *result* функция принимает дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`. Функция возвращает массив, каждый элемент которого соответствует одному полю в записи, или `FALSE`, если курсор достиг конца результирующей таблицы. Причем в качестве ключей массива выступают имена полей в результирующей таблице. В листинг 39.10 выводится HTML-таблица, в которой размещаются данные из таблицы `catalogs` учебной базы данных `shop`.

Листинг 39.10. Использование функции `mysql_fetch_assoc()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT * FROM catalogs";
  // Выполняем SQL-запрос
  $cat = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if(!$cat) exit(mysql_error());
  // Формируем таблицу
  echo "<table border=1>";
  while($catalogs = mysql_fetch_assoc($cat))
  {
    echo "<tr>
            <td>".$catalogs['id_catalog']."</td>
            <td>".$catalogs['name']."</td>
          </tr>";
  }
  echo "</table>";
?>
```

Результат работы скрипта из листинга 39.10 выглядит точно так же, как результат работы скрипта из листинга 39.9 (см. рис. 39.2). Следует обратить внимание, что на каждой итерации цикла `while` формируется массив `$catalogs`, в качестве ключей которого выступают имена столбцов в таблице `catalogs`. Если столбец является вычисляемым, то для того чтобы обратиться к нему, потребуется полное обращение к имени столбца. Так в листинге 39.11 при помощи встроенной функции MySQL `VERSION()` извлекается версия сервера MySQL. При этом в качестве ключа массива выступает строка `'VERSION()'`.

Листинг 39.11. Получение версии сервера с использованием `mysql_fetch_assoc()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
```

```
  $query = "SELECT VERSION()";
  // Выполняем SQL-запрос
  $ver = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if(!$ver) exit(mysql_error());
  // Определяем таблицу и заголовок
  $version = mysql_fetch_assoc($ver);
  echo $version['VERSION()'];
?>
```

Использование в качестве ключей полной сигнатуры вычисляемого столбца не всегда удобно, т. к. ключ может получиться громоздким или динамически изменяться. Для упрощения имени столбца в этом случае прибегают к назначению псевдонима при помощи оператора `AS` *(см. главу 7)*. Поэтому скрипт из листинга 39.11 можно переписать так, как показано в листинге 39.12.

Листинг 39.12. Использование псевдонима, назначаемого при помощи оператора `AS`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT VERSION() AS ver";
  // Выполняем SQL-запрос
  $ver = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if(!$ver) exit(mysql_error());
  // Определяем таблицу и заголовок
  $version = mysql_fetch_assoc($ver);
  echo $version['ver'];
?>
```

39.8. Функция *mysql_fetch_array()*

Функция `mysql_fetch_array()` возвращает текущую запись в результирующей таблице в виде массива. Причем тип массива (численный или ассоциативный) может быть выбран. Повторный вызов функции переводит курсор в результирующей таблице на следующую запись. 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, так и ассоциативные индексы.

> ***Замечание***
>
> По умолчанию второй аргумент принимает значение MYSQL_BOTH.

> ***Замечание***
>
> Режим работы функции mysql_fetch_array(), принимающей в качестве второго аргумента константы MYSQL_ASSOC и MYSQL_NUM, аналогичен 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. Наиболее интересен режим MYSQL_BOTH. В листинге 39.13 извлекается первая запись таблицы products и выводится дамп массива, возвращаемого функцией mysql_fetch_array().

> ***Замечание***
>
> Для вывода дампа массива применяется функция print_r(), результат функции заключается в теги <pre> и </pre>, чтобы сохранить форматирование, т. к. браузер воспринимает символы перевода строки как пробельные символы.

Листинг 39.13. Использование функции mysql_fetch_array()

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT * FROM products WHERE id_product = 1";
  // Выполняем SQL-запрос
  $prd = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if(!$prd) exit(mysql_error());
  // Возвращаем результат в виде массива
  $product = mysql_fetch_array($prd);
  echo "<pre>";
```

```
  print_r($product);
  echo "</pre>";
?>
```

Результат работы скрипта из листинга 39.13 представлен на рис. 39.3.

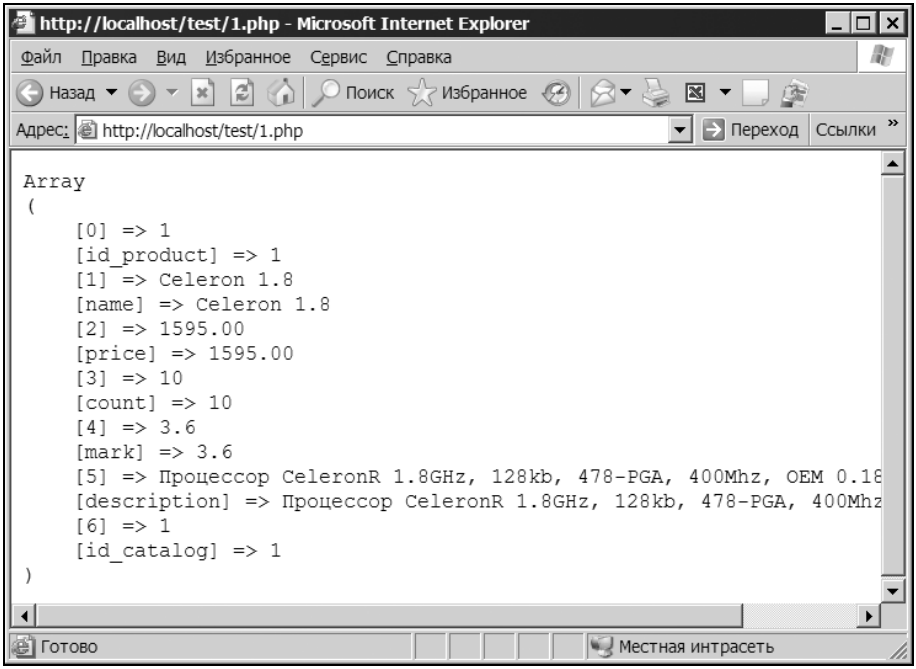


Рис. 39.3. Результат работы скрипта из листинга 39.13

Использовать функцию `mysql_fetch_array()` можно и для получения записей результирующей таблицы в цикле по аналогии с функциями `mysql_fetch_assoc()` и `mysql_fetch_row()` (листинг 39.14).

Листинг 39.14. Извлечение содержимого таблицы `catalogs`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT * FROM catalogs";
  // Выполняем SQL-запрос
```

```
  $cat = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if(!$cat) exit(mysql_error());
  // Формируем таблицу
  echo "<table border=1>";
  while($catalog = mysql_fetch_array($cat))
  {
    echo "<tr>
            <td>".$catalogs['id_catalog']."</td>
            <td>".$catalogs['name']."</td>
          </tr>";
  }
  echo "</table>";
?>
```

39.9. Функция *mysql_fetch_object()*

Функция `mysql_fetch_object()` возвращает текущую запись в виде объекта, имена членов которого совпадают с именами полей в результирующей таблице. Повторный вызов функции переводит курсор в результирующей таблице на следующую запись. Функция `mysql_fetch_object()` имеет следующий синтаксис:

```
object mysql_fetch_object (resource result)
```

В качестве единственного аргумента *result* функция принимает дескриптор запроса, возвращаемый функцией `mysql_query()`. Возвращает объект с членами, соответствующими столбцам в результирующей таблице, или `FALSE`, если рядов больше нет.

> ***Замечание***
>
> Имена полей, возвращаемые этой функцией, не зависят от регистра.

В листинге 39.15 приведен пример, в котором с помощью этой функции из таблицы `catalogs` выводятся первичный ключ таблицы `id_catalog` и имя каталога `name`.

Листинг 39.15. Использование функции `mysql_fetch_object()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT * FROM catalogs";
```

```
  // Выполняем SQL-запрос
  $cat = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if(!$cat) exit(mysql_error());
  while($catalog = mysql_fetch_object($cat))
  {
    echo "id_catalog: ".$catalog->id_catalog."<br>";
    echo "name: ".$catalog->name."<br>";
  }
?>
```

Результат работы скрипта из листинга 39.15 представлен на рис. 39.4.

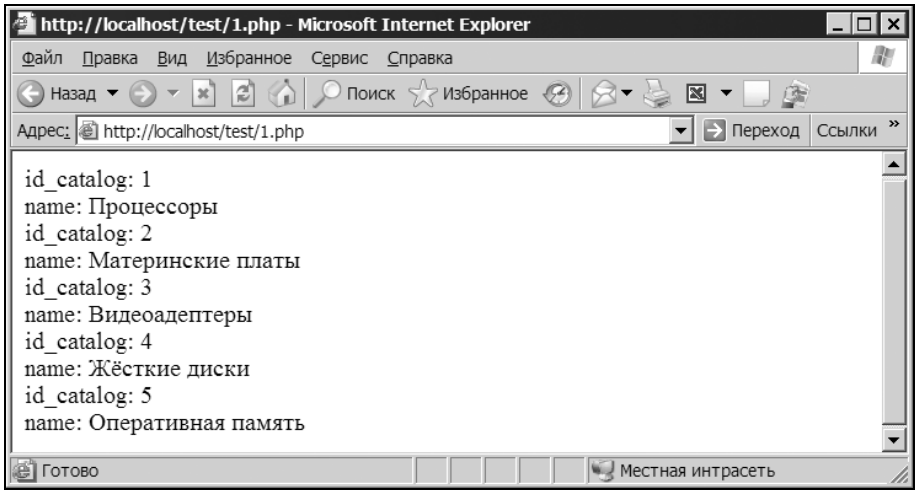


Рис. 39.4. Результат работы скрипта из листинга 39.15

39.10. Функция *mysql_num_rows()*

Одной из распространенных задач при выборке из базы данных является определение числа записей, которые возвращает функция `mysql_query()`. Для этого предназначена функция `mysql_num_rows()`, которая имеет следующий синтаксис:

```
int mysql_num_rows (resource result)
```

В качестве единственного аргумента *result* функция принимает дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`.

Замечание

Эта функция работает только с запросами SELECT. Чтобы получить количество рядов, обработанных функциями INSERT, UPDATE, DELETE, следует использовать функцию mysql_affected_rows().

Замечание

Функция mysql_num_fields() позволяет определить число столбцов в результате.

Создадим таблицу test, содержащую два поля: первичный ключ id_test и текстовое поле name (листинг 39.16).

Листинг 39.16. Таблица test

```
CREATE TABLE test (
  id_test int(11) NOT NULL auto_increment,
  name tinytext NOT NULL,
  PRIMARY KEY  (id_test)
);
```

В настоящий момент в таблице нет ни одной записи, поэтому попытка извлечь результирующую таблицу будет завершаться неудачей (листинг 39.17).

Листинг 39.17. Попытка извлечь несуществующую запись

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT name FROM test LIMIT 1";
  // Выполняем SQL-запрос
  $tst = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if(!$tst) exit(mysql_error());
  // Выводим результат
  echo mysql_result($tst,0);
?>
```

В результате работы скрипта из листинга 39.17 будет возникать ошибка (рис. 39.5).

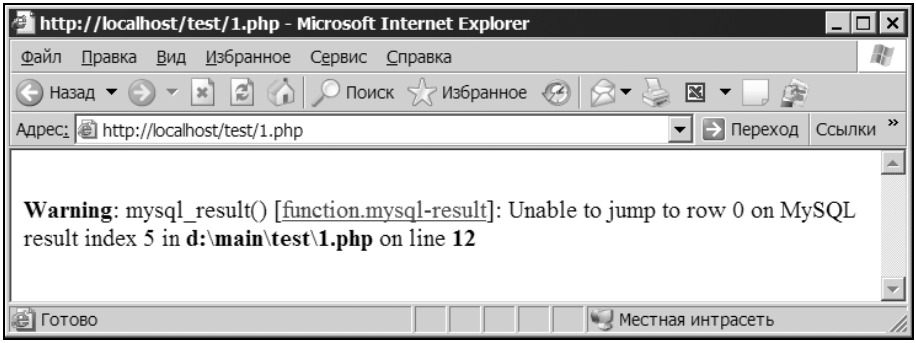


Рис. 39.5. Попытка извлечь несуществующую запись

Для предотвращения возникновения такой ошибки удобно воспользоваться функцией `mysql_num_rows()` (листинг 39.18).

Листинг 39.18. Использование функции `mysql_num_rows()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT name FROM test LIMIT 1";
  // Выполняем SQL-запрос
  $tst = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if(!$tst) exit(mysql_error());
  // Выводим результат, если в результирующей
  // таблице имеется хотя бы одна запись
  if(mysql_num_rows($tst) > 0)
  {
    echo mysql_result($tst,0);
  }
?>
```

Число строк можно так же определить при помощи отдельного SQL-запроса, воспользовавшись встроенной функцией MySQL — `COUNT()` *(см. главу 21)*:

```
SELECT COUNT(*) FROM test;
```

Так, скрипт из листинга 39.18 можно переписать следующим образом (листинг 39.19).

Листинг 39.19. Использование встроенной функции `COUNT()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Определяем количество записей в таблице test
  $query = "SELECT COUNT(*) FROM test";
  $cnt = mysql_query($query);
  if(!$cnt) exit(mysql_error());
  $total = mysql_result($cnt,0);
  if($total > 0)
  {
    // Формируем SQL-запрос
    $query = "SELECT name FROM test LIMIT 1";
    // Выполняем SQL-запрос
    $tst = mysql_query($query);
    // Проверяем успешность выполнения SQL-запроса
    if(!$tst) exit(mysql_error());
    // Выводим результат
    echo mysql_result($tst,0);
  }
?>
```

Использование функции `COUNT()` для определения числа записей на первый взгляд может показаться нерациональным, т. к. вместо одного запроса (листинг 39.18) используются два (листинг 39.19). Однако такой подход является единственно верным, когда результирующая таблица содержит только часть выборки, например, за счет ограничения конструкцией `LIMIT`, а для дальнейших вычислений требуется точное количество записей в таблице.

39.11. Система регистрации

В данном разделе будет разработана система регистрации, помещающая регистрационные данные пользователя в таблицу `userslist`, структура которой представлена в листинге 39.20.

Листинг 39.20. Таблица `userslist`

```
CREATE TABLE userslist (
  id_user int(11) NOT NULL auto_increment,
  name tinytext NOT NULL,
```

```
  pass tinytext NOT NULL,
  email tinytext NOT NULL,
  url tinytext NOT NULL,
  PRIMARY KEY  (id_user)
) TYPE=MyISAM;
```

Таблица `userslist` состоит из пяти полей, которые имеют следующие назначения:

- `id_user` — первичный ключ таблицы, обладающий атрибутом `AUTO_INCREMENT`;
- `name` — имя пользователя;
- `pass` — его пароль;
- `email` — адрес электронной почты пользователя;
- `url` — адрес домашней страницы пользователя.

В качестве обязательных для заполнения полей должны выступать только имя пользователя `name` и его пароль `pass`; адрес электронной почты `email` и адрес домашней страницы `url` могут не указываться. Система не должна позволять осуществлять регистрацию пользователя с именем, которое ранее было зарегистрировано. Скрипт, выполняющий регистрацию с такими условиями, может выглядеть так, как это представлено в листинге 39.21.

Листинг 39.21. Регистрация нового пользователя в базе данных

```
<table>
<form method=post>
<tr><td>Имя:</td><td><input type=text name=name></td></tr>
<tr><td>Пароль:</td><td><input type=password name=pass></td></tr>
<tr><td>Пароль:</td><td><input type=password name=pass_again></td></tr>
<tr><td>e-mail:</td><td><input type=text name=email></td></tr>
<tr><td>URL:</td><td><input type=text name=url></td></tr>
<tr><td></td><td><input type=submit value='Зарегистрировать'></td></tr>
</form>
</table>
<?php
  // Обработчик HTML-формы

  ////////////////////////////////////////////////
  // 1. Блок проверки правильности ввода данных
  ////////////////////////////////////////////////
  // Удаляем лишние пробелы
  $_POST['name'] = trim($_POST['name']);
```

```
$_POST['pass'] = trim($_POST['pass']);
$_POST['pass_again'] = trim($_POST['pass_again']);
// Проверяем, не пустой ли суперглобальный массив $_POST
if(empty($_POST['name'])) exit();
// Проверяем, правильно ли заполнены обязательные поля
if(empty($_POST['name'])) exit('Поле "Имя" не заполнено');
if(empty($_POST['pass'])) exit('Одно из полей "Пароль" не заполнено');
if(empty($_POST['pass_again'])) exit('Одно из полей "Пароль"
                                     не заполнено');
if($_POST['pass'] != $_POST['pass_again']) exit('Пароли не совпадают');
// Если введен e-mail, проверяем его на соответствие
if(!empty($_POST['email']))
{
  if(!preg_match("|^[-0-9a-z_]+@[-0-9a-z_]+\.[a-z]{2,6}$|i",
                 $_POST['email']))
  {
    exit('Поле "E-mail" должно соответствовать формату
          somebody@somewhere.ru');
  }
}
// Если на сервере не включены "магические кавычки",
// обрабатываем введенные пользователем данные
// функцией mysql_escape_string()
if (!get_magic_quotes_gpc())
{
  $_POST['name']  = mysql_escape_string($_POST['name']);
  $_POST['pass']  = mysql_escape_string($_POST['pass']);
  $_POST['email'] = mysql_escape_string($_POST['email']);
  $_POST['url']  = mysql_escape_string($_POST['url']);
}

////////////////////////////////////////////
// 2. Блок проверки имени на уникальность
////////////////////////////////////////////
// Устанавливаем соединение с базой данных
require_once("config.php");
// Проверяем, не зарегистрировано ли ранее переданное имя
$query = "SELECT COUNT(*) FROM userslist WHERE name = '$_POST[name]'";
$usr = mysql_query($query);
```

```
  if(!$usr) exit("Ошибка - ".mysql_error());
  $total = mysql_result($usr, 0);
  if($total > 0)
  {
    exit("Данное имя уже зарегистрировано. Пожалуйста, выберите другое");
  }

  ////////////////////////////////////////////////
  // 3. Блок регистрации пользователя
  ////////////////////////////////////////////////
  // Формируем и выполняем SQL-запрос на
  // добавление нового пользователя
  $query = "INSERT INTO userslist
            VALUES(NULL,
                   '$_POST[name]',
                   '$_POST[pass]',
                   '$_POST[email]',
                   '$_POST[url]')";
  if(mysql_query($query))
  {
    // Осуществляем перезагрузку страницы,
    // чтобы сбросить POST-данные
    echo "<HTML><HEAD>
          <META HTTP-EQUIV='Refresh' CONTENT='0; URL=$_SERVER[PHP_SELF]'>
          </HEAD></HTML>";
  } else exit("Ошибка при добавлении данных - ".mysql_error());
?>
```

Внешний вид приложения, представленного в листинге 39.21, может выглядеть так, как это показано на рис. 39.6.

Скрипт системы регистрации пользователей начинается с HTML-формы, состоящей из трех текстовых полей `name`, `email` и `url` для имени пользователя, его электронного адреса и адреса домашней страницы, соответственно, двух полей для пароля `pass` и `pass_again` (ввод и подтверждение) и кнопки, отправляющей содержимое HTML-формы обработчику. В качестве обработчика HTML-формы выступает сам скрипт.

Обработчик условно можно поделить на три части:

- блок проверки правильности ввода данных;
- блок проверки имени на уникальность;
- блок регистрации пользователя.

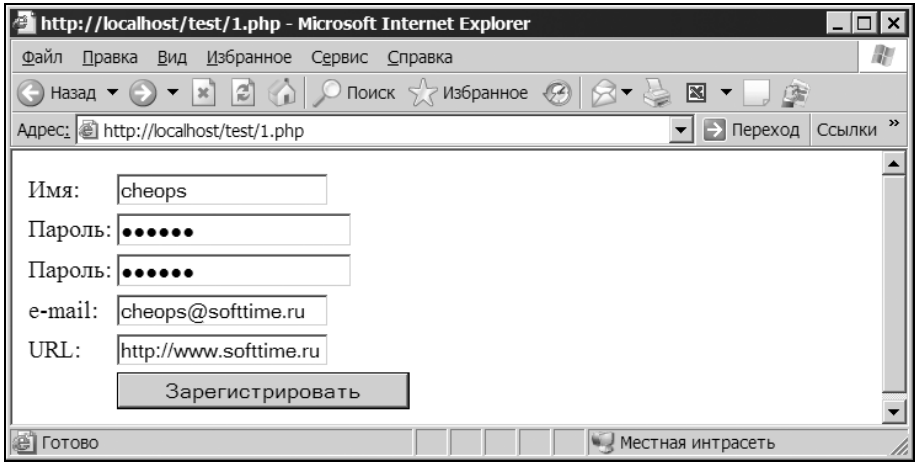


Рис. 39.6. Регистрация нового пользователя

Блок проверки правильности ввода данных выясняет, заполнены ли обязательные поля `name`, `pass` и `pass_again` и равны ли введенные пароли, если введен адрес электронной почты `email`, при помощи регулярных выражений проверяется соответствие его общепринятому формату. Здесь же посредством функции `get_magic_quotes_gpc()` анализируется, включен ли режим "магических кавычек", который экранирует специальные символы в SQL-запросе. Если режим отключен, то каждая из введенных пользователем строка подвергается обработке при помощи функции `mysql_escape_string()` — если в имени пользователя или его пароле используются одинарные кавычки, то они будут экранированы обратным слешем. В противном случае, если не подвергать такие строки экранированию, они нарушат синтаксис SQL-запроса `INSERT`, что приведет к ошибке.

Блок проверки имени на уникальность выполняет запрос, который проверяет, не было ли зарегистрировано такое имя ранее. Такая проверка сводится к подсчету числа строк в таблице, поле `name` которых совпадает с введенным пользователем именем. Если в таблице `userslist` уже имеются строки с таким именем, система прекращает работу.

Блок регистрации пользователя формирует SQL-запрос `INSERT` и выполняет его. В случае успешного выполнения запроса, страница перезагружается. Это необходимо для того, чтобы сбросить `POST`-данные, иначе перезагрузка страницы будет приводить к повторной попытке добавления записи в базу данных.

Для того чтобы проконтролировать список пользователей, находящихся в таблице `userslist`, разработаем дополнительный скрипт, выводящий список пользователей в виде гиперссылок, переход по которым будет приводить к странице с подробными сведениями о пользователе (листинг 39.22).

Листинг 39.22. Список пользователей в таблице `userslist`

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  if(empty($_GET['id_user']))
  {
    // Запрашиваем список всех пользователей
    $query = "SELECT * FROM userslist ORDER BY name";
    $usr = mysql_query($query);
    if(!$usr) exit("Ошибка - ".mysql_error());
    while($user = mysql_fetch_array($usr))
    {
      echo "<a href=$_SERVER[PHP_SELF]?id_user=$user[id_user]>
              $user[name]
            </a><br>";
    }
  }
  else
  {
    // Проверяем, является ли параметр целым числом
    if(!preg_match("|^[\d]+$|", $_GET['id_user']))
    {
      exit("Неверный формат URL-запроса");
    }
    // Запрашиваем информацию по текущему пользователю
    $query = "SELECT * FROM userslist WHERE id_user = $_GET[id_user]";
    $usr = mysql_query($query);
    if(!$usr) exit("Ошибка - ".mysql_error());
    $user = mysql_fetch_array($usr);
    echo "Имя пользователя - $user[name]<br>";
    if(!empty($user['email'])) echo "e-mail - $user[email]<br>";
    if(!empty($user['url'])) echo "URL - $user[url]<br>";
  }
?>
```

При первой загрузке скрипта `GET`-параметр `id_user` не определен, и скрипт выводит полный список пользователей. Каждая строка списка представляет собой гиперссылку

вида namescript.php?id_user=1, где namescript.php — имя скрипта, 1 — первичный ключ записи в таблице userslist, соответствующей текущему пользователю. Переход по каждой из ссылок приводит к загрузке этого же самого скрипта, но уже с GET-параметром id_user. После того как скрипт получает параметр id_user, управление передается блоку else, в котором выводится информация о пользователе с первичным ключом id_user. Страница с информацией о пользователе выглядит так, как это представлено на рис. 39.7.

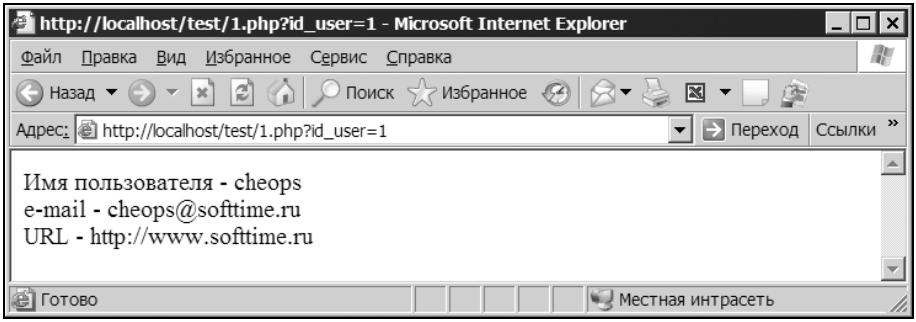


Рис. 39.7. Вывод информации о пользователе с первичным ключом id_user = 1

Важным элементом скрипта является блок проверки на соответствие GET-параметра id_user целому числу. Если проверка отсутствует, то возможно осуществление атаки SQL-инъекции. Для демонстрации данной угрозы создадим упрощенный вариант скрипта из листинга 39.22 (листинг 39.23).

Листинг 39.23. Скрипт вывода подробной информации о пользователе

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Запрашиваем информацию по текущему пользователю
  $query = "SELECT * FROM userslist WHERE id_user = $_GET[id_user]";
  $usr = mysql_query($query);
  if(!$usr) exit("Ошибка - ".mysql_error());
  $user = mysql_fetch_array($usr);
  echo "Имя пользователя - $user[name]<br>";
  if(!empty($user['email'])) echo "e-mail - $user[email]<br>";
  if(!empty($user['url'])) echo "URL - $user[url]<br>";
?>
```

SQL-инъекция для кода, приведенного в листинге 39.23, возможна благодаря тому, что GET-параметр `id_user` не проверяется на наличие посторонних символов. Вместо числа в SQL-запрос

```
SELECT * FROM userslist WHERE id_user = 1
```

может быть внедрена произвольная строка. Так, если в строку запроса подставить значение `'какой-то%20текст'`, то скрипт выдаст ошибку (рис. 39.8), т. к. будет осуществлена попытка выполнения запроса

```
SELECT * FROM userslist WHERE id_user = 'какой-то текст'
```

> ***Замечение***
>
> Символ `%20` обозначает пробел, запоминать коды недопустимых символов не обязательно, корректную для URL строку всегда можно получить, пропустив текст через функцию `urlencode()`.

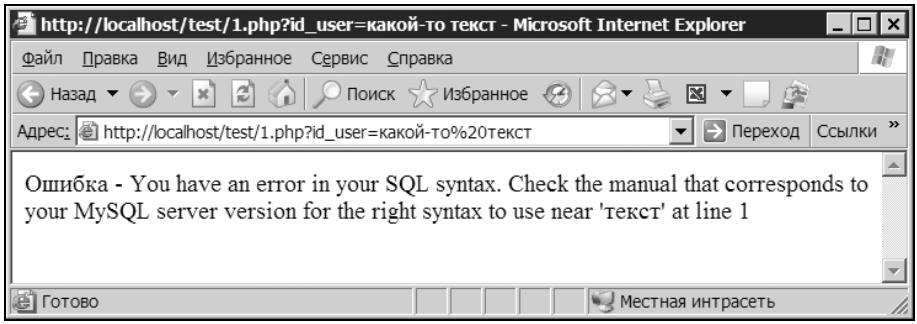


Рис. 39.8. Передача через строку запроса произвольного текста

Воспользовавшись конструкцией `UNION`, можно добавить еще один запрос того же формата, что и первый (листинг 39.24).

Листинг 39.24. Использование конструкции `UNION`

```
SELECT * FROM userslist WHERE id_user = -1
UNION
SELECT * FROM userslist WHERE id_user = 1
```

Код SQL-инъекции выделен жирным шрифтом. Как видно, первый запрос возвращает пустую результирующую таблицу, т. к. первичного ключа со значением `id_user`, равным –1, заведомо быть не может. Поэтому обрабатываться будет вторая запись, которая полностью может быть сформирована в строке запроса

```
1.php?id_user=
-1 %20UNION%20SELECT%20*%20FROM%20userslist%20WHERE%20id_user%20=%202
```

Добившись вывода результатов инъекционного SELECT-запроса, можно, например, получить доступ к паролю пользователя, который не выводится на странице подробной информации. Для этого следует расшифровать символ * в инъекционном запросе (листинг 39.25).

Листинг 39.25. Расшифровка символа * в инъекционном запросе

```
SELECT * FROM userslist WHERE id_user = -1
UNION
SELECT id_user, name, pass, email, url FROM userslist WHERE id_user = 1
```

Число столбцов в инъекционном запросе должно совпадать с числом столбцов из первого SELECT-запроса, иначе СУБД отклонит запрос как ошибочный. Однако оба столбца name и pass являются текстовыми — ничто не мешает поменять их местами (листинг 39.26).

Листинг 39.26. Перемена местами полей name и pass

```
SELECT * FROM userslist WHERE id_user = -1
UNION
SELECT id_user, pass, name, email, url FROM userslist WHERE id_user = 1
```

Так как имена столбцов формируются по первому запросу, вместо имени пользователя (name) подставляется его пароль (pass) (рис. 39.9).

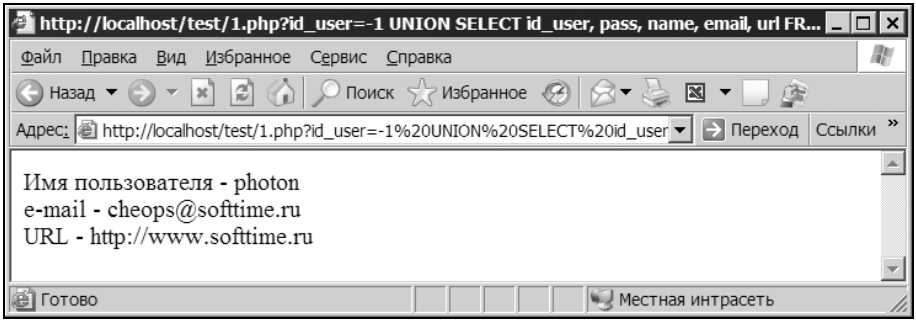


Рис. 39.9. Вывод пароля вместо имени пользователя

Как видно из рис. 39.9, на том месте, где должно выводиться имя пользователя, отображается его пароль.

39.12. Система авторизации

После того как создана система регистрации пользователей, можно ее реализовать — пользователи, посещая сайт, смогут вводить свое имя и пароль для получения доступа к своим персональным данным или сервисам.

Часто прибегают к авторизации посредством сессий — они обеспечивают большую безопасность, т. к. данные остаются на сервере, а клиент и сервер обмениваются только уникальным идентификатором сессии — SID, передающимся либо методом `GET` (как параметр адресной строки), либо через сессионные cookie.

Преимущество сессий заключается в том, что везде, где в начало скрипта будет помещена функция `session_start()`, можно обращаться к суперглобальному массиву `$_SESSION`, содержимое которого для каждого посетителя сайта будет уникальным.

Код системы авторизации может выглядеть так, как это представлено в листинге 39.27.

Листинг 39.27. Авторизация с использованием сессий

```
<?php
  // Инициируем сессию
  session_start();
?>
<form method=post>
Имя : <input type=text name=name
                        value='<?= $_SESSION['name']; ?>'><br>
Пароль : <input type=password name=password
                 value='<?= $_SESSION['password']; ?>'><br>
<input type=submit value=Отправить>
</form>
<?php
  // Обработчик формы
  if(!empty($_POST['name']) && !empty($_POST['password']))
  {
    // Устанавливаем соединение с базой данных
    require_once("config.php");
    // Защищаясь от SQL-инъекции, пропускаем
    // полученные пароль и имя пользователя через функцию
    // mysql_escape_string
    if (!get_magic_quotes_gpc())
    {
      $_POST['name'] = mysql_escape_string($_POST['name']);
```

```
      $_POST['password'] = mysql_escape_string($_POST['password']);
    }
    // Осуществляем запрос, который возвращает
    // число записей, удовлетворяющих паролю
    // и имени пользователя
    $query = "SELECT COUNT(*) FROM userslist
              WHERE name = '$_POST[name]' AND pass = '$_POST[password]'";
    $usr = mysql_query($query);
    if(!$usr) exit("Ошибка в блоке авторизации");
    // Получаем число записей
    if(mysql_result($usr,0) > 0) define("TOTAL", 1);
  }
  // Если число записей больше 0,
  // заносим данные о пользователе в сессию
  if(defined("TOTAL"))
  {
    $_SESSION['name'] = $_POST['name'];
    $_SESSION['password'] = $_POST['password'];
  }
  // Если посетитель прошел авторизацию - приветствуем его
  if(isset($_SESSION['name']))
  {
    // Устанавливаем соединение с базой данных
    require_once("config.php");
    // Выводим приветствие
    echo "Здравствуйте, ".$_SESSION['name']."!<br>";
    echo "Доступ к вашим секретным данным<br>";
    // Выводим данные пользователя
    $query = "SELECT * FROM userslist WHERE name = '$_SESSION[name]'";
    $usr = mysql_query($query);
    if(!$usr) exit(mysql_error());
    $user = mysql_fetch_array($usr);
    echo "Ваш e-mail: ".$user['email']."<br>";
    echo "Ваш URL: ".$user['url']."<br>";
  }
?>
```

Результат работы скрипта из листинга 39.28 может выглядеть так, как это представлено на рис. 39.10.

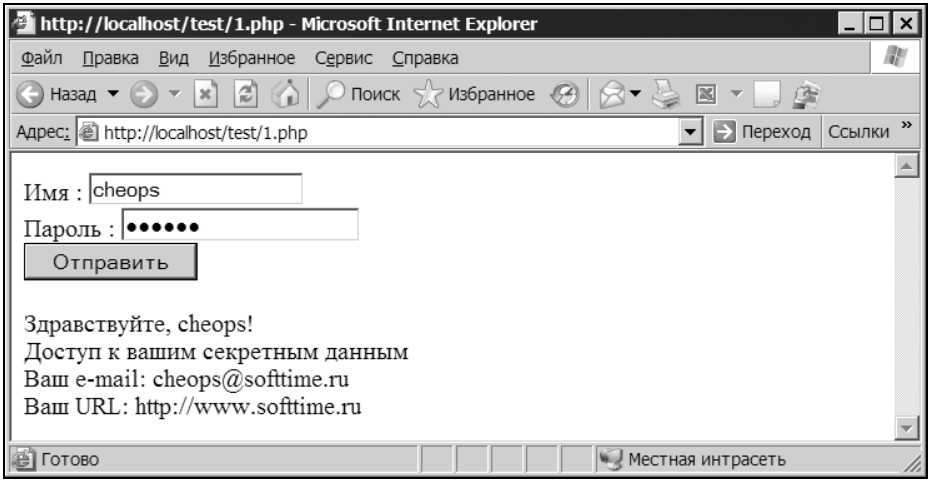


Рис. 39.10. Система авторизации с использованием сессий

39.13. Базовая HTTP-авторизация

Еще одним распространенным способом авторизации является использование так называемой базовой HTTP-авторизации (рис. 39.11).

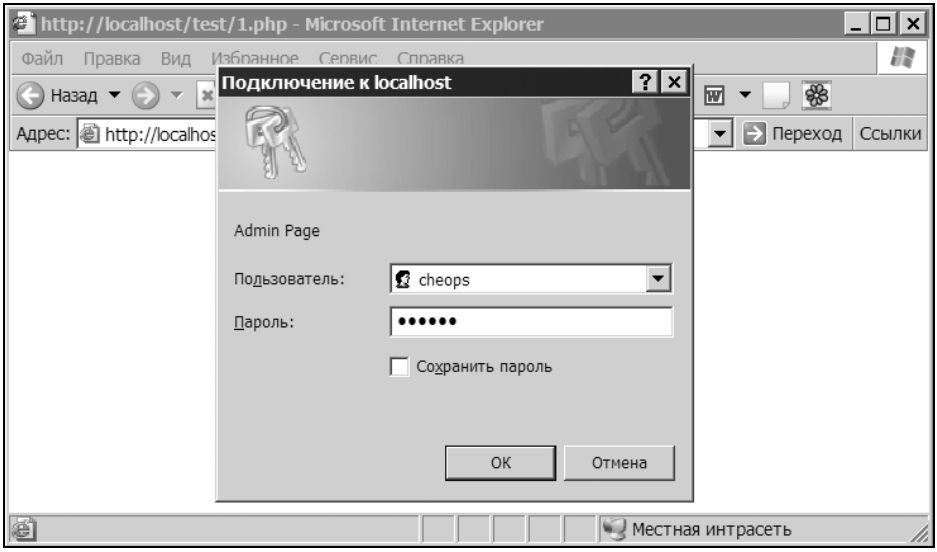


Рис. 39.11. Базовая HTTP-авторизация

Для реализации данного вида авторизации необходимо послать браузеру клиента HTTP-заголовки

```
WWW-Authenticate: Basic realm="Admin Page"
HTTP/1.0 401 Unauthorized
```

которые выведут форму для ввода имени пользователя и пароля, представленную на рис. 39.11. Имя пользователя будет помещено браузером в переменную суперглобального массива `$_SERVER['PHP_AUTH_USER']`, а пароль — в `$_SERVER['PHP_AUTH_PW']`.

> ***Замечание***
>
> Элементы суперглобального массива `$_SERVER['PHP_AUTH_USER']` и `$_SERVER['PHP_AUTH_PW']` доступны только в том случае, если PHP установлен в качестве модуля, а не CGI-приложения.

Необходимо создать файл `security_mod.php`, включение которого при помощи директивы `include` будет приводить к защите страницы паролем (листинг 39.28).

Листинг 39.28. Файл security_mod.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
  // Если пользователь не авторизовался - авторизуемся
  if(!isset($_SERVER['PHP_AUTH_USER']))
  {
    Header("WWW-Authenticate: Basic realm=\"Admin Page\"");
    Header("HTTP/1.0 401 Unauthorized");
    exit();
  }
  else
  {
    // Проверяем переменные $_SERVER['PHP_AUTH_USER'] и
    // $_SERVER['PHP_AUTH_PW'], чтобы предотвратить
    // SQL-инъекцию
    if (!get_magic_quotes_gpc())
    {
      $_SERVER['PHP_AUTH_USER'] =
                          mysql_escape_string($_SERVER['PHP_AUTH_USER']);
      $_SERVER['PHP_AUTH_PW'] =
                          mysql_escape_string($_SERVER['PHP_AUTH_PW']);
```

```
    }

    $query = "SELECT pass FROM userlist
              WHERE name='".$_SERVER['PHP_AUTH_USER']."'";
    $lst = @mysql_query($query);
    // Если ошибка в SQL-запросе - выдаем окно
    if(!$lst)
    {
      Header("WWW-Authenticate: Basic realm=\"Admin Page\"");
      Header("HTTP/1.0 401 Unauthorized");
      exit();
    }
    // Если такого пользователя нет - выдаем окно авторизации
    if(mysql_num_rows($lst) == 0)
    {
      Header("WWW-Authenticate: Basic realm=\"Admin Page\"");
      Header("HTTP/1.0 401 Unauthorized");
      exit();
    }
    // Если все проверки пройдены, сравниваем хэши паролей
    $pass = @mysql_fetch_array($lst);
    if(md5($_SERVER['PHP_AUTH_PW']) != $pass['pass'])
    {
      Header("WWW-Authenticate: Basic realm=\"Admin Page\"");
      Header("HTTP/1.0 401 Unauthorized");
      exit();
    }
  }
?>
```

Как видно из листинга, при неудачной авторизации клиенту отсылается повторное приглашение для ввода пароля:

```
WWW-Authenticate: Basic realm="Admin Page"
HTTP/1.0 401 Unauthorized
```

После чего работа скрипта останавливается при помощи функции `exit()`. Можно реализовать ограничение числа попыток ввода пароля. Для этого достаточно вместо приведенных выше HTTP-заголовков послать заголовок "Страница не найдена":

```
HTTP/1.0 404 Not Found
```

После того как модуль защиты `security_mod.php` создан, необходимо включить его при помощи директивы `include` в начале защищаемых страниц (листинг 39.29).

Листинг 39.29. Защита страницы

```
<?php
  // Модуль безопасности
  require_once("security_mod.php");
  echo "Данные страницы, к которой необходимо получить доступ";
?>
```

39.14. Пользователи online

Во многих Web-приложениях, таких как чаты, форумы, интернет-магазины требуется отслеживать посетителей, находящихся в данный момент на сайте, а также момент ухода их с сайта. Такая система может быть также полезна для контроля над закрытыми областями сайта: если к ней имеют доступ только два посетителя, а вы наблюдаете третьего, это позволит упредить взлом или минимизировать его последствия.

Будем считать, что для авторизации посетителей на сайте используется скрипт из листинга 39.27, т. е. имя пользователя регистрируется в элементе суперглобального массива `$_SESSION['name']`. Если пользователь не является авторизованным, вместо его имени должно выводиться "аноним".

Протокол HTTP не позволяет устанавливать сеансов и, как следствие, не позволяет отслеживать длительность работы посетителя с Web-ресурсом. Единственное, что можно зафиксировать, — это время обращения клиента к ресурсам сервера. После этого посетитель может часами читать загруженную страницу — Web-сервер не сможет узнать момент прекращения работы посетителя с загруженными ресурсами. Поэтому будем считать, что посетитель покинул Web-ресурс, если с момента последней загрузки им страницы прошло более 20 минут.

Для фиксирования времени обращения посетителей к страницам ресурса необходимо создать MySQL-таблицу `session`, в которой будут храниться имена пользователей и время последнего обращения их к странице (листинг 39.30).

Листинг 39.30. Таблица `session`

```
CREATE TABLE session (
  id_session tinytext NOT NULL,
  putdate datetime NOT NULL default '0000-00-00 00:00:00',
  name tinytext NOT NULL
) TYPE=MyISAM;
```

Таблица содержит три поля:

- `id_session` — идентификатор сессии;
- `putdate` — время последнего обращения посетителя к страницам сайта;
- `name` — имя пользователя.

Тогда для регистрации посетителей, которые находятся в данный момент на сайте, необходимо создать скрипт, представленный в листинге 39.31.

Листинг 39.31. Регистрация посетителей

```
<?php
  // Начинаем сессию
  session_start();
  // Получаем уникальный id сессии
  $id_session = session_id();
  // Устанавливаем соединение с базой данных
  include "config.php";
  // Проверяем, присутствует ли такой id в базе данных
  $query = "SELECT * FROM session
            WHERE id_session = '$id_session'";
  $ses = mysql_query($query);
  if(!$ses) exit("<p>Ошибка в запросе к таблице сессий</p>");
  // Если сессия с таким номером уже существует,
  // значит, пользователь в online - обновляем время его
  // последнего посещения
  if(mysql_num_rows($ses)>0)
  {
    $query = "UPDATE session SET putdate = NOW(),
                                 name = '$_SESSION[name]'
              WHERE id_session = '$id_session'";
    mysql_query($query);
  }
  // Иначе, если такого номера нет - посетитель только что
  // вошел - помещаем в таблицу нового посетителя
  else
  {
    $query = "INSERT INTO session
              VALUES('$id_session', NOW(), '$_SESSION[name]')";
    if(!mysql_query($query))
```

```
    {
      echo $query."<br>";
      echo "<p>Ошибка при добавлении пользователя</p>";
      exit();
    }
  }
  // Будем считать, что пользователи, которые отсутствовали
  // в течение 20 минут, покинули ресурс. Удаляем их
  // id_session из базы данных
  $query = "DELETE FROM session
            WHERE putdate < NOW() -  INTERVAL '20' MINUTE";
  mysql_query($query);
?>
```

В начале скрипта при помощи функции `session_id()` получается текущий идентификатор сессии. Если такой идентификатор отсутствует в таблице `session`, происходит добавление новой записи, что эквивалентно приходу на ресурс нового посетителя. Если же идентификатор присутствует в таблице `session`, следовательно, данный посетитель уже зашел на ресурс, и требуется обновить время его посещения и имя на тот случай, если он авторизовался под другим именем или осуществил авторизацию только сейчас. В конце скрипта происходит удаление всех записей таблицы, время посещения для которых было обновлено более чем 20 минут назад. Таким образом фиксируется уход посетителя с ресурса.

> ***Замечание***
>
> В чатах принято сообщать, что посетитель покинул Web-ресурс. Для этого пред удалением имеет смысл выбрать все записи, время обновления которых было произведено более чем 20 минут назад, и сообщить о том, что посетители с такими-то именами покинули чат.

Скрипт в листинге 39.31 следует подключить при помощи инструкции `include` ко всем часто посещаемым страницам сайта, для того чтобы надежно отслеживать присутствие посетителя на сайте.

После того как организовано динамическое обновление таблицы `session`, не составляет труда вывести список текущих посетителей, как это продемонстрировано в листинге 39.32.

Листинг 39.32. Выводим список текущих посетителей

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";
  // Выводим имена всех посетителей, записи о которых имеются
```

```
  // в таблице session
  $query = "SELECT * FROM session";
  $ath = mysql_query($query);
  if(!$ath) exit("<p>Ошибка в запросе к таблице сессий</p>");
  // Если хоть кто-то есть, выводим таблицу
  if(mysql_num_rows($ath)>0)
  {
    echo "<table>";
    while($author = mysql_fetch_array($ath))
    {
      // Если посетитель не зарегистрирован,
      // выводим вместо его имени - "аноним"
      if(empty($author['user'])) echo "<tr><td>аноним</td></tr>";
      else echo "<tr><td>".$author['name']."</td></tr>";
    }
    echo "</table>";
  }
?>
```

В цикле формируем таблицу с именами посетителей. Если имя неизвестно, например, посетитель не прошел авторизацию, выводим надпись "аноним".

39.15. Постраничная навигация

Создадим список гиперссылок с названиями элементов каталога (рис. 39.12), переход по которым приводил бы к списку товарных позиций (рис. 39.13), перечень которых выводился бы с элементами постраничной навигации. На странице будут отображаться три позиции.

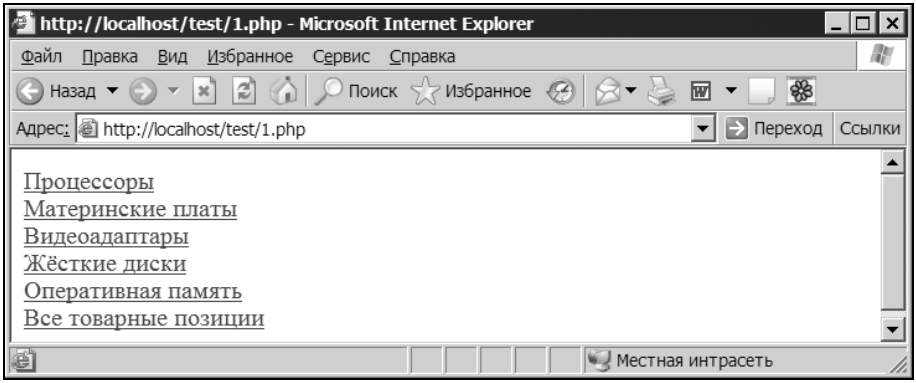


Рис. 39.12. Список элементов каталога

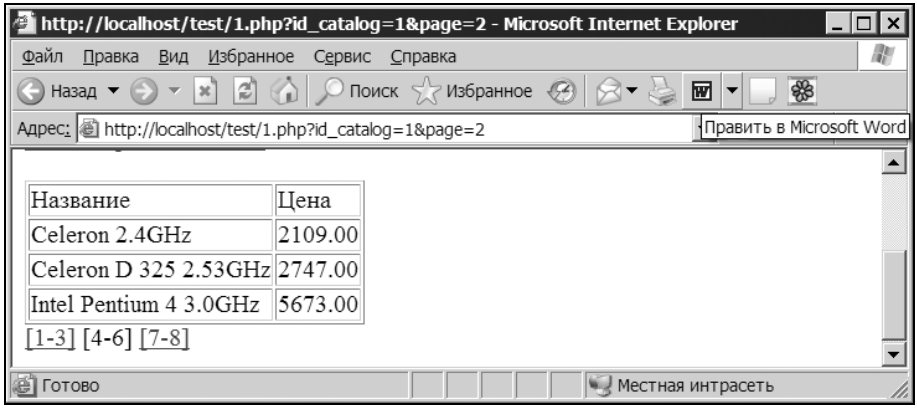


Рис. 39.13. Список товарных позиций с постраничной навигацией

Реализацию постраничной навигации средствами MySQL удобнее реализовывать при помощи конструкции `LIMIT N, M`, позволяющей извлечь `M` записей, начиная с позиции `N` (листинг 39.33).

Листинг 39.33. Реализация постраничной навигации

```
<?php
  // Число позиций на странице
  $pnumber = 3;

  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Формируем запрос на извлечение списка
  // элементов каталога
  $query = "SELECT * FROM catalogs";
  $cat = mysql_query($query);
  if(!$cat) exit(mysql_error());
  while($catalog = mysql_fetch_array($cat))
  {
    // Выводим ссылку на каталог
    echo "<a
href=$_SERVER[PHP_SELF]?id_catalog=$catalog[id_catalog]>$catalog[name]</a
><br>";
  }
  echo "<a href=$_SERVER[PHP_SELF]?id_catalog=0>Все товарные
позиции</a><br><br>";
```

```
  // Если передан параметр id_catalog, следовательно,
  // необходимо выводить информацию по товарным позициям
  if(preg_match("|^[\d]+$|i",$_GET['id_catalog']))
  {
    // Проверяем, передан ли номер текущей страницы
    if(isset($_GET['page'])) $page = $_GET['page'];
    else $page = 1;

    // Начальная позиция
    $start = (($page - 1)*$pnumber + 1);

    if($_GET[id_catalog] != 0) $where = "WHERE id_catalog =
$_GET[id_catalog]";
    else $where = "";
    // Формируем запрос на извлечение товарных
    // позиций текущего элемента каталога
    $query = "SELECT * FROM products
              $where
              ORDER BY price
              LIMIT $start, $pnumber";
    $prd = mysql_query($query);
    if(!$prd) exit(mysql_error());
    // Если для текущего элемента каталога имеется хотя бы
    // одна товарная позиция, выводим ее
    if(mysql_num_rows($prd) > 0)
    {
      echo "<table border=1>
              <tr>
                <td>Название</td>
                <td>Цена</td>
              </tr>";
      while($product = mysql_fetch_array($prd))
      {
        echo "<tr>
                <td>$product[name]</td>
                <td>$product[price]</td>
              </tr>";
      }
      echo "</table>";
    }

    // Число страниц
```

```
    $query = "SELECT COUNT(*) FROM products $where";
    $tot = mysql_query($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]?id_catalog=".
                       $_GET['id_catalog']."&page=".$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]?id_catalog=".
                       $_GET['id_catalog']."&page=".$i.">[".
                       (($i - 1)*$pnumber + 1)."-".
                       ($total - 1)."]</a> ";
        }
      }
    }
  }
?>
```

39.16. Алфавитная навигация

Реализуем алфавитную навигацию для содержимого таблицы `products` учебной базы данных `shop` (рис. 39.14). На страницу должен выводиться список гиперссылок, в виде первых букв алфавита, переход по которым должен приводить к списку товарных позиций, начинающихся на данную букву. Выводиться будут только те буквы, на которые начинается хотя бы одна товарная позиция из таблицы `products`.

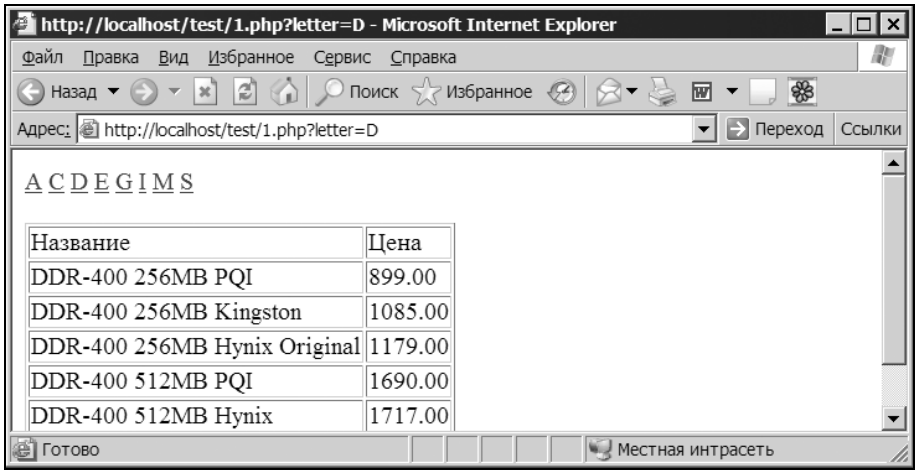


Рис. 39.14. Алфавитная навигация

Для реализации алфавитной навигации удобнее извлечь все первые буквы товарных позиций из таблицы `products` и сгруппировать их при помощи конструкции `GROUP BY`. Запрос, осуществляющий данную операцию, может выглядеть так, как это представлено в листинге 39.34.

Листинг 39.34. Извлечение первых букв товарных позиций

```
SELECT SUBSTRING(name,1,1) AS letter
FROM products
GROUP BY letter
ORDER BY letter
```

После этого достаточно сформировать ссылки с полученными буквами и передать через `GET`-параметр выбранную посетителем букву. Запрос, осуществляющий выбор всех товарных позиций, начинающихся на выбранную букву, может выглядеть так, как это представлено в листинге 39.35.

Листинг 39.35. Выбор записей

```
SELECT * FROM products
WHERE SUBSTRING(name,1,1) = 'D'
ORDER BY price
```

В листинге 39.36 оба запроса объединены в скрипт.

Листинг 39.36. Алфавитная навигация

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Формируем запрос на извлечение первых
  // букв товарных позиций
  $query = "SELECT SUBSTRING(name,1,1) AS letter
            FROM products
            GROUP BY letter
            ORDER BY letter";
  $prd = mysql_query($query);
  if(!$prd) exit(mysql_error());
  // Если имеется хотя бы одна запись,
  // выводим ее
  if(mysql_num_rows($prd) > 0)
  {
    while($product = mysql_fetch_array($prd))
    {
      echo "<a href=$_SERVER[PHP_SELF]?letter=$product[letter]>
                                          $product[letter]</a>";
    }
  }

  // Если передан параметр letter и он
  // состоит из одного символа, выводим
  // содержимое таблицы
  if(preg_match("|^[a-z]$|i", $_GET['letter']))
  {
    // Выводим товарные позиции
```

```
    $query = "SELECT * FROM products
              WHERE SUBSTRING(name,1,1) = '$_GET[letter]'
              ORDER BY price";
    $prd = mysql_query($query);
    if(!$prd) exit(mysql_error());
    // Если для текущего элемента каталога имеется хотя бы
    // одна товарная позиция, выводим ее
    if(mysql_num_rows($prd) > 0)
    {
      echo "<br><br><table border=1>
              <tr>
                <td>Название</td>
                <td>Цена</td>
              </tr>";
      while($product = mysql_fetch_array($prd))
      {
        echo "<tr>
                <td>$product[name]</td>
                <td>$product[price]</td>
              </tr>";
      }
      echo "</table>";
    }
  }
?>
```

39.17. Сортировка

Создадим таблицу с названием, стоимостью, оценкой и числом товарных позиций на складе из таблицы `products` учебной базы данных `shop`. В шапке страницы поместим названия столбцов в виде гиперссылок, переход по которым осуществлял бы сортировку выбранного столбца. Повторный переход по гиперссылке должен приводить к сортировке данных в обратном порядке (рис. 39.15).

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 products
         ORDER BY $order $desc
```

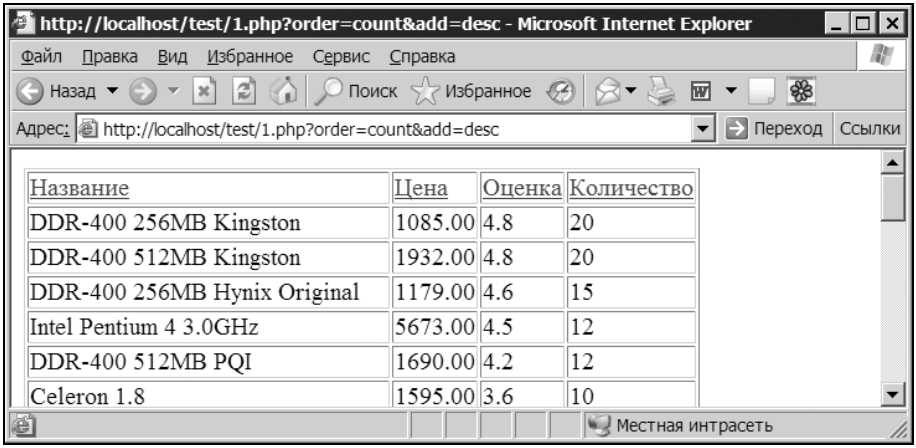


Рис. 39.15. Таблица с возможностью сортировки

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 39.37).

Листинг 39.37. Сортировка

```
<?php
  // Проверяем параметры, переданные скрипту
  $order = "name";
  if($_GET['order'] == 'name')  $order = "name";
  if($_GET['order'] == 'price') $order = "price";
  if($_GET['order']  == 'mark')  $order = "mark";
  if($_GET['order'] == 'count') $order = "count";
  if($_GET['add']   == 'desc')
  {
    $desc = "DESC";
    $add  = "";
  }
  else
  {
    $desc = "";
    $add  = "desc";
  }
  // Устанавливаем соединение с базой данных
```

```
  require_once("config.php");

  // Выводим товарные позиции
  $query = "SELECT * FROM products
            ORDER BY $order $desc";
  $prd = mysql_query($query);
  if(!$prd) exit(mysql_error());
  // Если для текущего элемента каталога имеется хотя бы
  // одна товарная позиция, выводим ее
  if(mysql_num_rows($prd) > 0)
  {
    echo "<table border=1>
            <tr>
  <td><a href=$_SERVER[PHP_SELF]?order=name&add=$add>Название</a></td>
  <td><a href=$_SERVER[PHP_SELF]?order=price&add=$add>Цена</a></td>
  <td><a href=$_SERVER[PHP_SELF]?order=mark&add=$add>Оценка</a></td>
  <td><a href=$_SERVER[PHP_SELF]?order=count&add=$add>Количество</a></td>
            </tr>";
    while($product = mysql_fetch_array($prd))
    {
      echo "<tr>
              <td>$product[name]</td>
              <td>$product[price]</td>
              <td>$product[mark]</td>
              <td>$product[count]</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 появляется возможность при обращении к скрипту 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.

39.18. Двойной выпадающий список

При работе с элементами каталога часто возникает задача реализации двойного выпадающего списка. В первом списке выбирается раздел, а во второй список автоматически подставляются товарные позиции из данного раздела. Создадим двойной выпадающий список для разделов и товарных позиций в учебной базе данных (рис. 39.16).

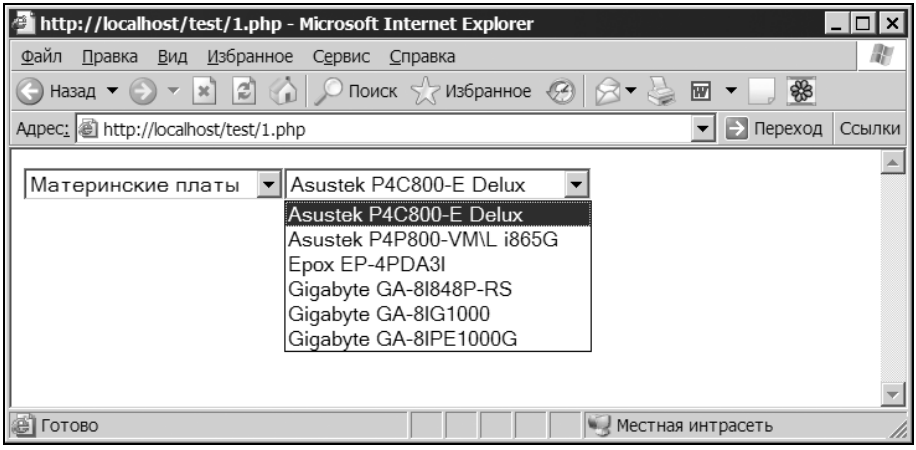


Рис. 39.16. Двойной выпадающий список

Для вывода первого списка необходимо сформировать HTML-форму с тегом `<select>` в цикле обработки результата SQL-запроса на выборку записей таблицы `catalogs`. Каждый элемент списка (`<option>`) будет представлять собой название позиции каталога, а атрибут `value` будет принимать значение первичного ключа текущего элемента каталога. Для того чтобы форма автоматически перезагружалась в тег `<select>`, следует добавить обработчик события `onchange` (выбор элемента из списка) следующего вида: `onchange='this.form.submit()'`. После перезагрузки страницы нужно отслеживать, не передан ли параметр `id_catalog` (именно так мы назовем первый выпадающий список). Параметр `id_catalog` следует проверить при помощи регулярного выражения на соответствие целому числу, т. к. злоумышленник через него может осуществить SQL-инъекцию (листинг 39.38).

Листинг 39.38. Двойной выпадающий список

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Начало HTML-формы
```

```
echo "<form method=post>";

// Формируем первый выпадающий список
$query = "SELECT * FROM catalogs
          ORDER BY name";
$cat = mysql_query($query);
if(!$cat) exit(mysql_error());
// Если имеется хотя бы одна запись,
// формируем выпадающий список
if(mysql_num_rows($cat) > 0)
{
  echo "<select name=id_catalog onchange='this.form.submit()'>";
  echo "<option value=0>Не имеет значения</option>";
  while($catalog = mysql_fetch_array($cat))
  {
    if($_POST['id_catalog'] == $catalog['id_catalog'])
    {
      $selected = "selected";
    }
    else $selected = "";
    echo "<option value=$catalog[id_catalog] $selected>
                $catalog[name]</option>";
  }
  echo "</select>";
}

// Проверяем, является ли параметр id_catalog числом
if(preg_match("|^[\d]+$|",$_POST['id_catalog']))
{
  // Формируем второй выпадающий список
  $query = "SELECT * FROM products
            WHERE id_catalog = $_POST[id_catalog]
            ORDER BY name";
  $prd = mysql_query($query);
  if(!$prd) exit(mysql_error());
  // Если для текущего элемента каталога имеется хотя бы
  // одна товарная позиция, формируем выпадающий список
  if(mysql_num_rows($prd) > 0)
  {
```

```
      echo "<select name=id_product onchange='this.form.submit()'>";
      while($product = mysql_fetch_array($prd))
      {
        echo "<option value=$product[id_product]>
                    $product[name]</option>";
      }
      echo "</select>";
    }
  }

  // Конец HTML-формы
  echo "</form>";
?>
```

К недостаткам метода, представленного в листинге 39.38, относится тот факт, что пользователь вынужден постоянно осуществлять перезагрузку страницы. Другим подходом, которым можно воспользоваться при решении этой задачи, является предварительная загрузка различных вариантов во втором выпадающем списке на страницу и отображение их средствами JavaScript при выборе элемента в первом выпадающем списке.

Для этого при формировании первого списка сформируем массив первичных ключей `$array_catalog` таблицы `catalogs`. Используя данный массив, сформируем массив JavaScript (листинг 39.39).

Листинг 39.39. Формируем массив JavaScript

```
<script language='JavaScript1.1' type='text/javascript'>
<!--
  var messageIdList = new Array(<?= implode(",", $array_catalog) ?>);
-->
</script>
```

В конечной HTML-странице массив будет выглядеть следующим образом:

```
var messageIdList = new Array(3,4,2,5,1);
```

Данный список нам понадобится для сокрытия и отображения различных вариантов второго выпадающего списка. Каждый вариант списка будет называться `product$i`, где `$i` пробегает значения из массива `messageIdList`. Кроме того, каждый выпадающий список получит атрибут `id`, значение которого будет совпадать с текущим номером `$i`, чтобы при операциях сокрытия и отображения к списку было удобнее обращаться из JavaScript. Для того чтобы скрыть выпадающий список, атрибуту `style` тега `<select>` необходимо присвоить значение `"display:none"`. Для того

чтобы отобразить его на странице, необходимо изменить значение атрибута `style` на `"display:block"`. Перед тем как отобразить выбранный список, все списки следует скрыть, чтобы в каждый момент времени на странице был только один вариант второго выпадающего списка. Эта операция будет осуществляться при помощи обработчика события `onchange` первого выпадающего списка (листинг 39.40).

Листинг 39.40. Обработчик события `onchange` первого выпадающего списка

```
<script language='JavaScript1.1' type='text/javascript'>
<!--
  var messageIdList = new Array(<?= implode(",", $array_catalog) ?>);
  function show(sel)
  {
    for (i = 0; i < messageIdList.length; i++)
    {
      document.getElementById(messageIdList[i]).style.display = "none";
    }
    Var selectedVal = sel.options[sel.selectedIndex].value;
    document.getElementById(selectedVal).style.display = "block";
  }
//-->
</script>
```

В цикле `for` все выпадающие списки скрываются, после чего выбранный список отображается. Сам PHP-код, формирующий HTML-форму, может выглядеть так, как это представлено в листинге 39.41.

Листинг 39.41. Двойной выпадающий список

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Начало HTML-формы
  echo "<form action=handler.php method=post>";

  // Формируем первый выпадающий список
  $query = "SELECT * FROM catalogs
            ORDER BY name";
  $cat = mysql_query($query);
```

```
if(!$cat) exit(mysql_error());
// Если имеется хотя бы одна запись,
// формируем выпадающий список
if(mysql_num_rows($cat) > 0)
{
  echo "<select name=id_catalog
         onchange='show(this.form.id_catalog)'>";
  echo "<option value=0>Не имеет значения</option>";
  while($catalog = mysql_fetch_array($cat))
  {
    if($_POST['id_catalog'] == $catalog['id_catalog'])
    {
      $selected = "selected";
    }
    else $selected = "";
    echo "<option value=$catalog[id_catalog] $selected>
                $catalog[name]</option>";

    // Формируем массив первичных ключей элементов каталога
    $array_catalog[] = $catalog['id_catalog'];
  }
  echo "</select>";
}

// Формируем второй выпадающий список
$query = "SELECT * FROM catalogs";
$cat = mysql_query($query);
if(!$cat) exit(mysql_error());
// Если имеется хотя бы одна запись,
// формируем выпадающий список
if(mysql_num_rows($cat) > 0)
{
  while($catalog = mysql_fetch_array($cat))
  {
    // Формируем скрытые списки
    $query = "SELECT * FROM products
              WHERE id_catalog = $catalog[id_catalog]
              ORDER BY name";
    $prd = mysql_query($query);
```

```
      if(!$prd) exit(mysql_error());
      // Если для текущего элемента каталога имеется хотя бы
      // одна товарная позиция, формируем выпадающий список
      if(mysql_num_rows($prd) > 0)
      {
        echo "<select id=$catalog[id_catalog]
           style=\"display:none\" name=product$catalog[id_catalog]>";
        while($product = mysql_fetch_array($prd))
        {
          if($_POST['id_product'] == $product['id_product'])
          {
            $selected = "selected";
          }
          else $selected = "";
          echo "<option value=$product[id_product] $selected>
                        $product[name]</option>";
        }
        echo "</select>";
      }
    }
  }
  echo "</br><input type=submit name=send value=Отправить>";

  // Конец HTML-формы
  echo "</form>";
?>
```

В качестве обработчика HTML-формы здесь выступает файл с именем handler.php. Несмотря на то, что выпадающие списки скрыты на HTML-странице, в суперглобальный массив `$_POST` попадут значения из всех пяти выпадающих списков. В этом легко убедиться, подставив в файл handler.php обработчик из листинга 39.42.

Листинг 39.42. Обработчик handler.php

```
<?php
  echo "<pre>";
  print_r($_POST);
  echo "</pre>";
?>
```

Результат может выглядеть так, как это представлено на рис. 39.17.

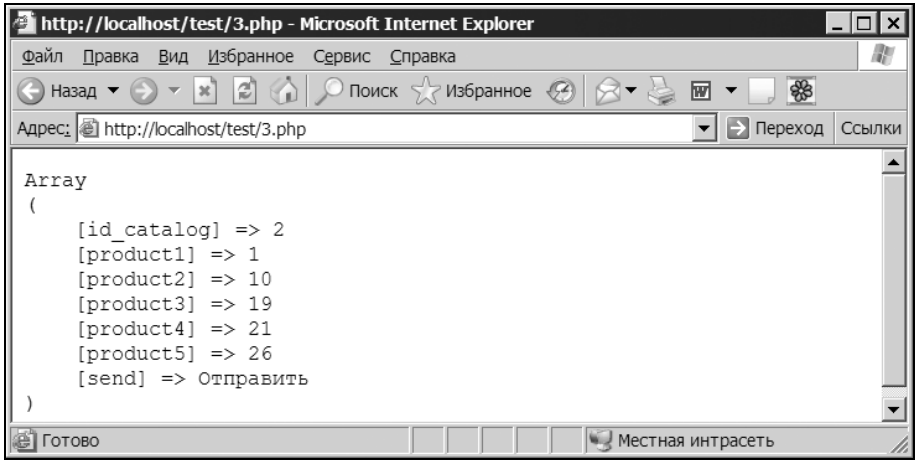


Рис. 39.17. Содержимое `POST`-массива, полученное из HTML-формы

Как видно из рис. 39.17, в первом выпадающем списке был выбран элемент, соответствующий первичному ключу таблицы `catalogs`, равному 2. Следовательно, можно доверять только списку `product2`, значение которого равно 10. Все остальные элементы массива `$_POST` не имеют осмысленного значения.

39.19. Удаление сразу нескольких позиций

Выведем список товарных позиций из таблицы `products` учебной базы данных `shop` в таблице. Напротив каждой товарной позиции отобразим флажок (checkbox). При нажатии кнопки **Удалить** все выбранные (отмеченные с помощью флажков) товарные позиции будут удаляться из базы данных (рис. 39.18).

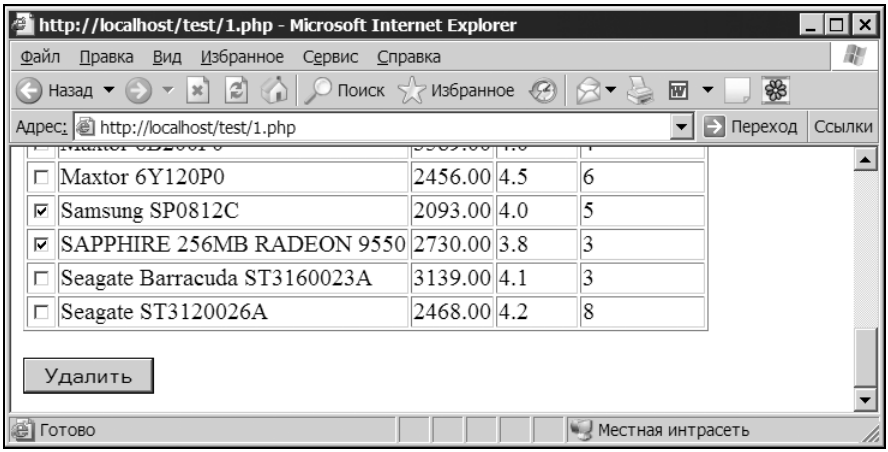


Рис. 39.18. Удаление сразу нескольких позиций

Для того чтобы удалить сразу несколько записей из таблицы `products`, необходимо сформировать SQL-запрос вида

```
DELETE FROM products WHERE id_product IN (4, 6, 8, ..., 20)
```

Данный запрос удалит все записи, в которых значение первичного ключа `id_product` совпадает с одним из значений в списке оператора `IN` (листинг 39.43).

Листинг 39.43. Удаление нескольких товарных позиций из таблицы `products`

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Выводим товарные позиции
  $query = "SELECT * FROM products
            ORDER BY name";
  $prd = mysql_query($query);
  if(!$prd) exit(mysql_error());
  // Если для текущего элемента каталога имеется хотя бы
  // одна товарная позиция, выводим ее
  if(mysql_num_rows($prd) > 0)
  {
    echo "<form method=post>";
    echo "<table border=1>
            <tr>
              <td> </td>
              <td>Название</td>
              <td>Цена</td>
              <td>Оценка</td>
              <td>Количество</td>
            </tr>";
    $i = 0;
    while($product = mysql_fetch_array($prd))
    {
      echo "<tr>
              <td><input type=checkbox name=product[]
value=$product[id_product]></td>
              <td>$product[name]</td>
              <td>$product[price]</td>
              <td>$product[mark]</td>
```

```
            <td>$product[count]</td>
          </tr>";
      $i++;
    }
    echo "</table>";
    echo "</br><input type=submit name=send value=Удалить>";
    echo "</form>";
  }

  // Если суперглобальный массив $_POST не пуст,
  // производим обработку запроса
  if(!empty($_POST))
  {
    // Для удаления товарных позиций необходимо
    // сформировать запрос вида
    // DELETE FROM products WHERE id_product IN (4, 6, 8, ..., 20),
    // где цифры в скобках являются элементами
    // массива $_POST['product'][]
    $temp = array();
    foreach($_POST['product'] as $id_product)
    {
      // Проверяем, является ли переменная $id_product числом
      if(preg_match("|^[\d]+$|",$id_product))
      {
        $temp[] = $id_product;
      }
    }
    // Формируем и выполняем запрос на удаление
    // нескольких позиций
    $query = "DELETE FROM products
              WHERE id_product IN (".implode(",",$temp).")";
    if(mysql_query($query))
    {
      echo "<HTML><HEAD>
         <META HTTP-EQUIV='Refresh' CONTENT='0; URL=$_SERVER[PHP_SELF]'>
            </HEAD></HTML>";
    }
  }
?>
```

39.20. Хранение MP3-файлов в базе данных

Часто на серверах хост-провайдеров место, отводимое под файлы виртуального хоста, тарифицируется, а место, отводимое под таблицы базы данных, не подвергается тарификации. Сохранение объемных файлов в базе данных позволяет избежать покупки более дорогих тарифных планов.

Еще одной причиной размещения файлов в базе данных является защита их от несанкционированной загрузки в обход системы авторизации или оплаты. При хранении файлов на диске у злоумышленника увеличиваются шансы загрузить файл в обход системы авторизации.

> ***Замечание***
>
> Если никаких причин хранить бинарные файлы в базе данных MySQL нет, лучше этого и не делать. СУБД MySQL проектировалась как очень быстрая база данных для работы с короткими текстами. Кроме того, размещая в таблице файлы, вы понижаете надежность Web-приложения по сравнению со случаем, когда файлы хранятся непосредственно на жестком диске в каталоге.

Создадим таблицу в базе данных и скрипт, позволяющий сохранять в ней MP3-файлы. В системе должен быть также скрипт, выводящий список доступных для загрузки MP3-файлов. Переход по ссылке должен приводить к загрузке файла с сервера на машину посетителя. При этом файл не должен сохраняться в промежуточные файлы на сервере, а передаваться непосредственно из базы данных клиенту (рис. 39.19).



Рис. 39.19. Размещение MP3-файлов в базе данных

Для хранения MP3-файлов в базе данных MySQL необходимо создать таблицу `mp3` (листинг 39.44), одно из полей которой будет производным от поля `BLOB`. Поля типа `BLOB` специально созданы для хранения бинарных данных и гарантируют, что поме-

щенная в них информация не будет испорчена кодировкой. Таблица `mp3` состоит из трех полей:

- `id_mp3` — первичный ключ, снабженный атрибутом `AUTO_INCREMENT`;
- `name` — имя файла;
- `content` — бинарное содержимое MP3-файла.

Листинг 39.44. Таблица `mp3`

```
CREATE TABLE mp3 (
  id_mp3 INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  content LONGBLOB NOT NULL,
  PRIMARY KEY (id_mp3)
) TYPE=MyISAM;
```

Замечание

MP3-файлы обычно обладают большим размером, поэтому для корректной работы с ними необходимо убедиться, что все серверные директивы позволяют работать с таким объемом данных. Директивы конфигурационного файла php.ini должны позволять передавать достаточный объем данных методом `POST` (директива `post_max_size`) и через загружаемый файл (директива `upload_max_filesize`). Кроме того, база данных MySQL должна позволять оперировать длинными SQL-запросами (директива `max_allowed_packet` файла my.ini).

Скрипт, осуществляющий загрузку MP3-файлов на сервер и размещение их в базе данных, представлен в листинге 39.45.

Листинг 39.45. Загрузка MP3-файлов в базу данных

```
<form enctype='multipart/form-data' method=post>
<input type="file" name="mp3"><br>
<input type=submit value='Загрузить'>
</form>
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Обработчик HTML-формы
  if(!empty($_FILES))
  {
    // Проверяем, является ли переданный файл MP3-файлом
    if($_FILES['mp3']['type'] == 'audio/mpeg')
```

```
    {
      // Читаем содержимое файла
      $content = file_get_contents($_FILES['mp3']['tmp_name']);
      // Уничтожаем файл во временном каталоге
      unlink($_FILES['mp3']['tmp_name']);

      // Экранируем специальные символы в бинарном содержимом файла
      $content = mysql_escape_string($content);

      // Формируем запрос на добавление файла в таблицу
      $query = "INSERT INTO mp3 VALUES
                (NULL, '".$_FILES['mp3']['name']."', '$content')";
      if(mysql_query($query))
      {
        // Осуществляем автоматическую перезагрузку страницы
        // для очистки POST-данных
        echo "<HTML><HEAD>
         <META HTTP-EQUIV='Refresh' CONTENT='0; URL=$_SERVER[PHP_SELF]'>
             </HEAD></HTML>";
      } else exit(mysql_error());
    }
  }

  // Выводим список файлов
  $query = "SELECT * FROM mp3";
  $mp = mysql_query($query);
  if(!$mp) exit(mysql_error());
  // Если имеется хотя бы одна запись,
  // выводим ее
  if(mysql_num_rows($mp) > 0)
  {
    while($mp3 = mysql_fetch_array($mp))
    {
      echo "<a href=get.php?id_mp3=$mp3[id_mp3]>$mp3[name]</a><br>";
    }
  }
?>
```

После того как MP3-файл загружен на сервер во временный каталог, проверяется, является ли он музыкальным файлом, при помощи условного оператора `if`:

```
if($_FILES['mp3']['type'] == 'audio/mpeg')
```

После этого содержимое файла переводится в переменную `$content`, а временный файл уничтожается при помощи функции `unlink()`. Перед помещением содержимого MP3-файла в базу данных оно подвергается действию функции `mysql_escape_string()`, которая экранирует спецсимволы, способные исказиться при передаче запроса по сети.

В конце скрипта выводится список MP3-файлов, уже помещенных в базу данных. Каждая позиция представляет собой гиперссылку на файл get.php, которому через параметр `id_mp3` передается первичный ключ записи, которая должна быть предоставлена пользователю для загрузки. Сам файл get.php может выглядеть так, как это представлено в листинге 39.46.

Листинг 39.46. Передача файла из базы данных пользователю

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Провряем, передан ли параметр id_mp3
  // и является ли он целым числом, чтобы
  // предотвратить SQL-инъекцию
  if(!preg_match("|^[\d]+$|",$_GET['id_mp3']))
  {
    exit("Недопустимый формат URL-запроса");
  }

  // Извлекаем MP3-файл из базы данных
  $query = "SELECT * FROM mp3
            WHERE id_mp3 = $_GET[id_mp3]";
  $mp3 = mysql_query($query);
  if(!$mp3) exit(mysql_error());
  $file = mysql_fetch_array($mp3);

  // Устнавливаем имя загружаемого файла
  header("Content-Disposition: attachment; filename=$file[name]");
  // Отсылаем заголовки на загрузку файла
  header("Content-type: application/octet-stream");
```

```
  // Отправляем файл пользователю
  echo $file['content'];
?>
```

После того как GET-параметр id_mp3 проверяется при помощи регулярного выражения на соответствие числу, из таблицы mp3 извлекается запись запрошенного MP3-файла. Далее браузеру отправляются HTTP-заголовки с названием файла и описанием содержимого, в котором сообщается, что в документе передаются бинарные данные. Отсутствие этих заголовков приведет к тому, что браузер посчитает файл за обычную страницу и выведет содержимое файла в своем окне. Завершает скрипт отправка бинарных данных при помощи конструкции echo.

> ***Замечание***
>
> После оператора `echo $file['content'];` не должно быть никакого вывода в окно браузера, в том числе не должно быть пробелов и переводов строк после тега ?>, иначе файл будет испорчен.

39.21. Хранение изображений в базе данных

Еще чаще, чем хранение музыкальных файлов, встает задача хранения в базе данных графических файлов. Разработаем систему хранения графических изображений в базе данных, причем организуем вывод не ссылок, а самих изображений, по три штуки на одной странице (рис. 39.20). При этом графические файлы не будут сохраняться в промежуточные файлы на сервере, а передаваться непосредственно из базы данных клиенту.

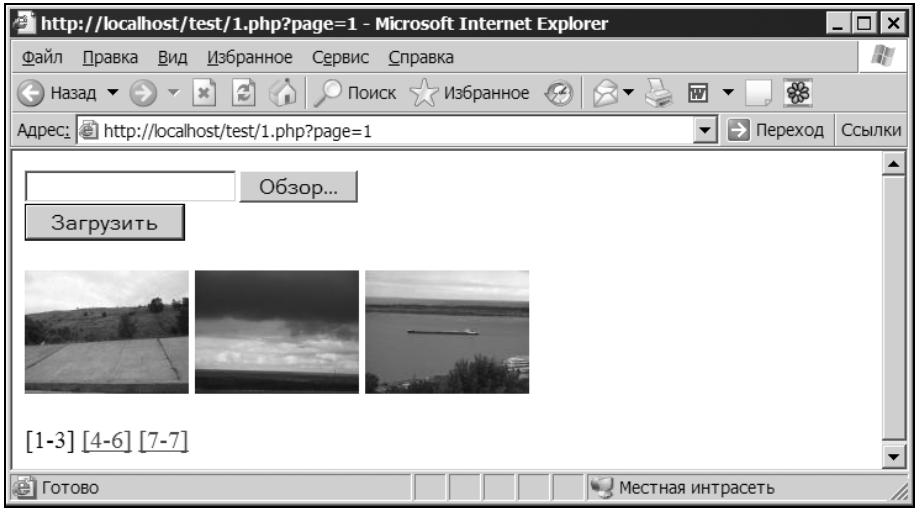


Рис. 39.20. Вывод изображений из базы данных

Для хранения изображений в базе данных MySQL необходимо создать таблицу `image` (листинг 39.47), одно из полей которой будет производным от поля `BLOB`. Таблица `image` состоит из трех полей:

- `id_image` — первичный ключ, снабженный атрибутом `AUTO_INCREMENT`;
- `name` — имя файла;
- `content` — бинарное содержимое графического файла.

Листинг 39.47. Таблица `image`

```
CREATE TABLE image (
  id_image INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  content LONGBLOB NOT NULL,
  PRIMARY KEY (id_image)
) TYPE=MyISAM;
```

Скрипт, осуществляющий загрузку графических файлов на сервер и размещение их в базе данных, представлен в листинге 39.48.

Листинг 39.48. Загрузка графических файлов в базу данных

```
<form enctype='multipart/form-data' method=post>
<input type="file" name="image"><br>
<input type=submit value='Загрузить'>
</form>
<?php
  // Число изображений на странице
  $pnumber = 3;

  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Обработчик HTML-формы
  if(!empty($_FILES))
  {
    // Проверяем, является ли переданный файл графическим
    if(substr($_FILES['image']['type'],0,5) == 'image')
    {
      // Читаем содержимое файла
      $content = file_get_contents($_FILES['image']['tmp_name']);
      // Уничтожаем файл во временном каталоге
```

```
    unlink($_FILES['image']['tmp_name']);

    // Экранируем специальные символы в бинарном содержимом файла
    $content = mysql_escape_string($content);

    // Формируем запрос на добавление файла в таблицу
    $query = "INSERT INTO image VALUES(NULL,
                                       '".$_FILES['image']['name']."',
                                       '$content')";
    if(mysql_query($query))
    {
      // Осуществляем автоматическую перезагрузку страницы
      echo "<HTML><HEAD>
        <META HTTP-EQUIV='Refresh' CONTENT='0; URL=$_SERVER[PHP_SELF]'>
           </HEAD></HTML>";
    } else exit(mysql_error());
  }
}

// Проверяем, передан ли номер текущей страницы
if(isset($_GET['page'])) $page = $_GET['page'];
else $page = 1;

// Начальная позиция
$start = (($page - 1)*$pnumber + 1);

// Выводим список файлов
$query = "SELECT * FROM image LIMIT $start, $pnumber";
$img = mysql_query($query);
if(!$img) exit(mysql_error());
// Если имеется хотя бы одна запись,
// выводим ее
if(mysql_num_rows($img) > 0)
{
  while($image = mysql_fetch_array($img))
  {
    echo "<img src=get.php?id_image=$image[id_image]> ";
  }
}
echo "<br><br>";

// Число страниц
```

```
  $query = "SELECT COUNT(*) FROM image";
  $tot = mysql_query($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 - 1)."]</a> ";
      }
    }
  }
?>
```

Для того чтобы пропустить все графические файлы, проверяется не все содержимое элемента суперглобального массива `$_FILES['image']['type']`, а лишь первые пять символов:

```
if(substr($_FILES['image']['type'],0,5) == 'image')
```

Если они равны подстроке 'image', мы имеем дело с графическим изображением. После этого логика скрипта совпадает со скриптом из листинга 39.45, однако в области под формой загрузки изображения выводятся не ссылки, а изображения. Атрибуту src тега <img> передается ссылка на файл get.php, который имеет следующее содержание (листинг 39.49).

Листинг 39.49. Вывод изображения из базы данных

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Проверяем, передан ли параметр id_image
  // и является ли он целым числом, чтобы
  // предотвратить SQL-инъекцию
  if(!preg_match("|^[\d]+$|",$_GET['id_image']))
  {
    exit("Недопустимый формат URL-запроса");
  }

  // Извлекаем графический файл из базы данных
  $query = "SELECT * FROM image
            WHERE id_image = $_GET[id_image]";
  $img = mysql_query($query);
  if(!$img) exit(mysql_error());
  $image = mysql_fetch_array($img);

  // Отсылаем заголовки на загрузку файла
  header("Content-type: image/*");
  // Отправляем файл пользователю
  echo $image['content'];
?>
```

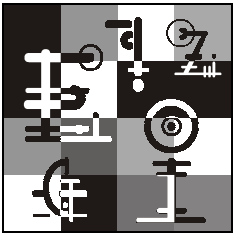
Здесь в качестве заголовка перед отправкой содержимого графического файла передается HTTP-заголовок "Content-type: image/*", который сообщает, что после него последует графический файл (формат файла не уточняется).

> ***Замечание***
>
> После оператора echo $image['content']; не должно быть никакого вывода в окно браузера, в том числе не должно быть пробелов и переводов строк после тега ?>, иначе изображение будет испорчено.

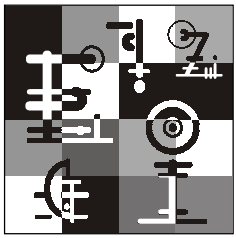
Заключение

Что бы ни 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, на котором вы можете задать свои вопросы **http://www.softtime.ru/forum/index.php?id_forum=3**. 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!



ПРИЛОЖЕНИЯ



Приложение 1

Пространственные расширения MySQL

MySQL поддерживает пространственные расширения, при помощи которых можно в удобной форме хранить и анализировать геометрические объекты. Данные расширения необходимы для построения ГИС-систем.

> ***Определение***
>
> *ГИС (Географическая информационная система)* называют информационную систему, обеспечивающую сбор, хранение, обработку, доступ, отображение и распространение пространственно-координированных данных (пространственных данных). В качестве наиболее типичной ГИС можно назвать интерактивную карту, предоставляющую сведения об объектах на карте. Примером ГИС-системы может служить сервис GoogleEarth (**http://earth.google.com/**).

MySQL реализует пространственные расширения согласно открытой спецификации консорциума OGS (Open GIS Consortium, **http://www.opengis.org**), в который входят более 300 компаний и университетов. Задачи консорциума достаточно разнообразны, и им подготовлено большое число документов. При реализации пространственных расширений разработчики MySQL ориентировались на документ "Спецификации простых функций OpenGIS для SQL", который доступен на сайте OGS по адресу **http://portal.opengeospatial.org/files/?artifact_id=13228**.

> ***Замечание***
>
> Пространственные расширения для версий MySQL, младше 5.0.16, доступны только для таблиц MyISAM, начиная с версии 5.0.16. Дополнительно к этому появилась возможность использовать пространственные расширения в таблицах InnoDB, NDB, BDB и ARCHIVE.

Пространственные расширения — это набор геометрических типов для столбцов и функций, которые ими оперируют. Геометрические типы используют для моделирования геометрических элементов (которые также называют *геопространственными элементами* или *геометриями*), которые могут иметь различные свойства:

- объект (например, гора, водоем, город);
- пространство (например, область, соответствующая району или лесному массиву);
- определенное местонахождение (например, перекресток как место пересечения двух улиц).

Определение

Географический элемент — это точка или совокупность точек, которые представляют любой объект в мире, имеющий определенное месторасположение.

П1.1. Геометрическая модель OpenGIS

Геометрическая модель является объектно-ориентированной. Это означает, что все объекты относятся к тому или иному географическому классу, которые выстроены в иерархию, в которой потомки наследуют свойства родительских классов (рис. П1.1). Наследование означает, что свойства класса `Geometry` доступны во всех классах иерархии, свойства класса `GeometryCollection` доступны для всех производных классов (`MultiPoint`, `MultiCurve`, `MultySurface`, `MultiLineString` и `MultiPolygon`). Для того чтобы свойства были доступны сразу большой группе производных классов, его достаточно добавить в их базовый класс.

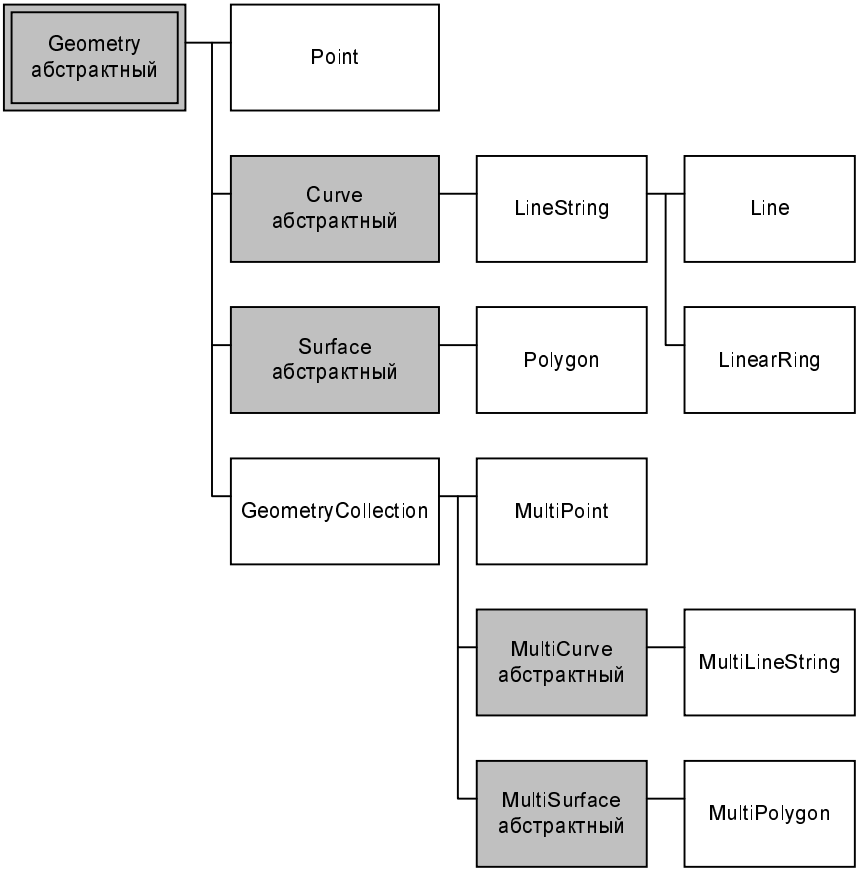


Рис. П1.1. Иерархия геометрических классов

Классы, которые помечены на рис. П1.1 серым цветом, являются абстрактными, т. е. для них невозможно создать объект, они предназначены лишь для того, чтобы задавать общие свойства производных классов. Для классов, помеченных на рис. П1.1 белым цветом, допускается создание объектов.

В начале иерархии помещен базовый абстрактный класс `Geometry`, все производные от него классы можно условно разделить как нуль-, одно- и двумерные:

- `Point` представляет нуль-мерные объекты, которые являются точками;
- `Curve` представляет класс одномерных объектов, которые изображают линию;
- `Surface` представляет класс двумерных объектов;
- `GeometryCollection` представляет собой коллекции для нуль-мерных (`MultiPoint`), одномерных (`MultiCurve`) и многомерных (`MultiSurface`) объектов.

Остановимся поподробнее на каждом из классов геометрической иерархии.

П1.1.1. Класс *Geometry*

Класс `Geometry` — это базовый класс в иерархии геометрических классов. Так как он является абстрактным, для него невозможно создать объект — его назначение снабжать каждый класс общими для всех классов свойствами, что позволяет даже разные объекты обрабатывать одинаково. Класс `Geometry` имеет следующие свойства:

- тип — данное свойство определяет, какому из классов геометрической иерархии принадлежит текущий объект;
- идентификатор пространственной системы координат (Spatial Reference Identifier, SRID) — идентифицирует пространственное значение координат, в котором задан объект;
- координаты — это числа с двойной точностью, т. е. числа, под которые отводится 8 байтов. Все не пустые геометрические объекты (если они не пустые) включают, как минимум, одну пару координат (x, y). Координаты зависят от идентификатора пространственной системы координат (SRID), т. к. представление координат на плоскости и сфере (например, если рассматривается земная поверхность) отличается;
- внутреннее пространство, границы и внешнее пространство — геометрический элемент может занимать определенное место в пространстве. Внешнее пространство — это все пространство, незанятое геометрическим объектом, внутреннее — это пространство, занимаемое геометрическим объектом. Границы — это раздел между внутренним и внешним пространствами. Ряд объектов, например, объекты класса `Point`, не имеет внутреннего пространства — в этом случае границы остаются пустыми;
- минимальный ограничивающий прямоугольник (Minimum Bounding Rectangle, MBR), или огибающая — координаты углов такого прямоугольника вычисляются по максимальному и минимальному значениям координат (x, y);

- измерение объекта — может принимать одно из следующих значений:
  - –1 — для пустого геометрического объекта;
  - 0 — для геометрического объекта, не имеющего длины или области (данное значение принимают объекты класса `Point`, `MultiPoint`);
  - 1 — для геометрического объекта с отличной от нуля длиной и равной нулю областью (данное значение принимают объекты класса `Line`, `LineString`, `MultiLineString`);
  - 2 — для геометрических объектов с ненулевой областью (данное значение принимают объекты класса `Poligon`, `MultiPoligon`);
- простота — объект может быть простым (simple) и непростым (non-simple). Для каждого класса геометрической иерархии существуют собственные критерии для данного свойства;
- замкнутость — объект может быть замкнутым (closed) и незамкнутым (not closed). Для каждого класса геометрической иерархии существуют собственные критерии для данного свойства;
- пустота — объект может быть пустым (empty) и непустым (not empty). Геометрический объект считается пустым, если у него нет ни одной точки. Границы, внешнее и внутреннее пространства пустого геометрического объекта не определены (принимают значение `NULL`).

П1.1.2. Класс *Point*

Класс `Point` определяет нуль-мерный геометрический объект, представляющий собой точку в пространстве координат (рис. П1.2).

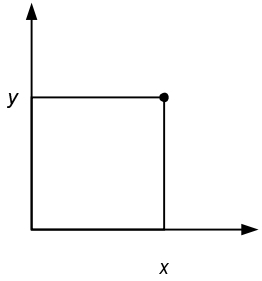


Рис. П1.2. Координаты объекта класса `Point`

Так как класс `Point` является производным от класса `Geometry`, он обладает всеми свойствами класса `Geometry`, добавляя ряд своих:

- значение координаты *x*;
- значение координаты *y*.

Замечание

Границы объекта являются пустыми, т. е. принимают значение `NULL`.

П1.1.3. Класс *Curve*

Абстрактный класс `Curve` определяет одномерный геометрический объект, представляющий собой кривую, которая строится как последовательность точек. Так как класс `Curve` является производным от класса `Geometry`, он обладает всеми свойствами класса `Geometry`, добавляя ряд своих:

- класс `Curve` определяет координаты точек, составляющих кривую;
- геометрический объект класса `Curve` считается простым, если кривая, которую он представляет, не проходит через одну и ту же точку дважды;
- геометрический объект класса `Curve` считается замкнутым, если начальная точка кривой совпадает с ее конечной точкой;
- границы замкнутой кривой всегда пустые;
- границы незамкнутой кривой состоят из двух ее конечных точек;
- простая и замкнутая кривая `Curve` представляет собой линейное кольцо, т. е. объект класса `LinearRing`.

П1.1.4. Класс *LineString*

Абстрактный класс `Curve` представляет собой абстрактную кривую, состоящую из точек. Класс `LineString` уточняет способ интерполяции между точками, а именно определяет линейную интерполяцию, или ломаную (рис. П1.3).

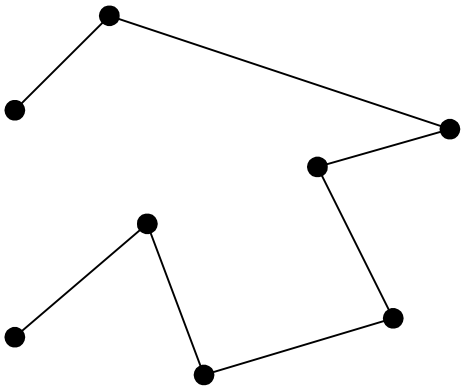


Рис. П1.3. Объект класса `LineString`

Класс `LineString` имеет следующие свойства:

- объект класса `LineString` представляет собой совокупность координат сегментов, которые определяются двумя точками (`Point`), задающими начало и конец сегмента;
- объект класса `LineString`, состоящий только из двух точек, эквивалентен объекту класса `Line`;
- объект `LineString` эквивалентен объекту класса `LinearRing`, если он является одновременно и замкнутым, и простым.

Следует отметить, что объект класса `LinearRing`, несмотря на то, что его название дословно можно перевести, как линейное кольцо, представляет широкий круг многоугольников. Так, в частности, квадрат относится к этому типу, как и круг.

П1.1.5. Класс *Surface*

Абстрактный класс `Surface` определяет объект, представляющий собой двумерную поверхность. Данный класс обладает следующими свойствами:

- объект класса `Surface` состоит из единственной внешней границы и нуля или более внутренних границ;
- внутренние и внешние границы простого объекта `Surface` обязательно должны быть замкнутыми.

П1.1.6. Класс *Polygon*

Объект класса `Polygon` представляет собой двумерную поверхность, которая имеет одну внешнюю границу и две или более внутренних границ (рис. П1.4).

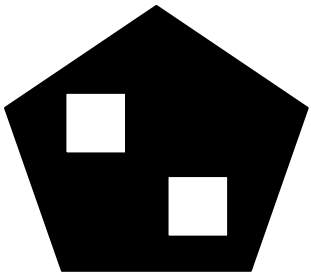


Рис. П1.4. Объект класса `Polygon`

На рис. П1.4 черная область соответствует точкам, которые принадлежат объекту класса `Polygon`.

Объект класса `Polygon` считается простым, если он удовлетворяет следующим условиям:

- границы простого объекта `Polygon` представляют собой множество простых объектов `LinearRing` и образуют его внешние и внутренние границы;
- простой объект класса `Polygon` не имеет пересекающихся колец. Кольца могут пересекаться лишь в одной точке (по касательной);
- простой объект не содержит отдельных линий, острых углов или разрывов границ;
- внутреннее пространство простого объекта `Polygon` представляет собой совокупность связанных точек;
- для простого объекта класса `Polygon` допускается наличие отверстий (рис. П1.4), однако отверстия не должны соприкасаться с внешними границами.

Если объект `Polygon` не удовлетворяет перечисленным выше условиям, он считается сложным.

П1.1.7. Класс *GeometryCollection*

Объект класса `GeometryCollection` — это геометрический элемент, представляющий собой совокупность из одного и большего числа геометрических объектов любого класса геометрической иерархии (в том числе и других коллекций). Для данного вида коллекций существует только одно ограничение — все геометрические элементы должны быть представлены в единой пространственной системе координат.

П1.1.8. Класс *MultiPoint*

Объект класса `MultiPoint` представляет собой совокупность точек (объектов `Point`). При этом точки никак не упорядочены и не связаны между собой. Объекты `MultiPoint` характеризуются следующими свойствами:

- объект `MultiPoint` — это нуль-мерный геометрический элемент;
- объект `MultiPoint` считается простым, если координаты ни одной из его точек не совпадают друг с другом;
- границы объекта `MultiPoint` пусты.

П1.1.9. Класс *MultiCurve*

Абстрактный класс `MultiCurve` представляет собой совокупность кривых (объектов `Curve`). Класс `MultiCurve` характеризуется следующими свойствами:

- объекты, наследующие классу `MultiCurve`, представляют собой одномерный геометрический объект;
- объект `MultiCurve` считается простым, если все его элементы (кривые) являются простыми, а любые два объекта пересекаются только в точках границ;

- объект `MultiCurve` является замкнутым, если все составляющие его элементы (кривые) являются замкнутыми;
- значения границ замкнутого объекта `MultiCurve` всегда пустые.

П1.1.10. Класс *MultiString*

Объект класса `MultiLineString` представляет собой совокупность ломаных линий (объектов класса `LineString`).

П1.1.11. Класс *MultiSurface*

Абстрактный класс `MultiSurface` представляет собой совокупность поверхностей (объектов класса `Surface`). Так как класс является абстрактным, для него отсутствует возможность создания объектов.

П1.1.12. Класс *MultiPolygon*

Объект класса `MultiPolygon` является двумерным и представляет собой совокупность поверхностей (объектов класса `Polygon`). Класс `MultiPolygon` характеризуется следующими свойствами:

- объект класса `MultiPolygon` не может содержать двух или более элементов `Polygon`, внутренние пространства которых пересекаются или соприкасаются в бесконечном количестве точек;
- объект класса `MultiPolygon` не может содержать острых выступов, обрезанных или оборванных границ;
- границы объекта `MultiPolygon` представляет собой множество замкнутых кривых (значения `LineString`), соответствующих границам его элементов `Polygon`.

П1.2. Форматы пространственных данных

До настоящего времени рассматривались структура и свойства иерархии геометрических классов. Каким образом осуществляется представление точек, кривых и поверхностей в базе данных? Они представляются в виде геометрических объектов, которые создаются при помощи классов. Для представления пространственных объектов используют два формата:

- WKT — стандартный текстовый формат (Well-Known Text);
- WKB — стандартный двоичный формат (Well-Known Binary).

Оба формата применяются для интерфейсных целей — ввода данных, представления результатов клиентам СУБД и т. д. СУБД MySQL имеет собственное внутреннее представление геометрических элементов.

Как следует из названия, формат WKT предназначен для обмена геометрической информацией в виде текста. Следует помнить, что элементы WKT — это строки, а не

конструкции и функции, т. е. их необходимо заключать в кавычки, если они упоминаются в SQL-запросе, например, `'Point(15 20)'`.

Формат WKB используется для представления двоичных данных в бинарном формате, который хранится в столбцах типа `BLOB`. Для представления данных используются одно-, четырех- и восьмибайтовые целые числа без знака (листинг П1.1).

Листинг П1.1. Используемые для представления геометрических данных типы целых чисел

```
// byte   : 8-bit unsigned integer (1 byte)
// uint32 : 32-bit unsigned integer (4 bytes)
// double : double precision number (8 bytes)
```

П1.2.1. Объект *Point*

Объект `Point` в WKT-формате представляется в виде `Point(X Y)`, где `X` — координата по оси абсцисс, а `Y` — координата по оси ординат. Следует иметь в виду, что значения `X` и `Y` разделяются пробелом, использование других разделителей, например, запятой не допускается. В качестве примера можно привести определение точки `Point(15 20)` (рис. П1.5).

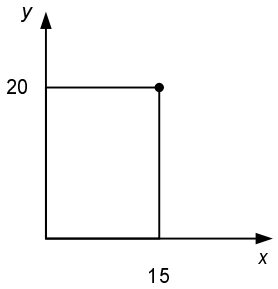


Рис. П1.5. Точка `Point(15 20)`

Представление объекта `Point` в WKB-формате выглядит как шестнадцатеричная последовательность. Например, `Point(1 1)` выглядит следующим образом:

```
0101000000000000000000F03F000000000000F03F
```

Данная последовательность состоит из следующих компонентов:

```
Порядок байтов: 01
       Тип WKB: 01000000
             X: 000000000000F03F
             Y: 000000000000F03F
```

Значение порядка байтов может принимать либо значение 0, либо 1 для указания сохранения байтов в прямом или обратном порядке. Тип WKB, код которого определяет геометрический тип, может принимать значения от 1 до 7 для обозначения объектов `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon` и `GeometryCollection`. Координаты `X` и `Y` представляются в виде целых чисел двойной точности (`double`). Как правило, нет надобности воспроизводить бинарные WKB-значения вручную, они получаются автоматически при объявлении типа данных `WKBPoint` (листинг П1.2).

Листинг П1.2. Тип `WKBPoint`

```
WKBPoint
{
  byte    byteOrder;
  uint32  wkbType;     // Для Point принимает значение 1
  Point   point;
}
```

Для работы со структурами данных необходимы вспомогательные множества `wkbGeometryType` и `wkbByteOrder` и тип `Point`, представленные в листинге П1.3.

Листинг П1.3. Вспомогательные структуры данных

```
enum wkbGeometryType
{
  wkbPoint              = 1,
  wkbLineString         = 2,
  wkbPolygon            = 3,
  wkbMultiPoint         = 4,
  wkbMultiLineString    = 5,
  wkbMultiPolygon       = 6,
  wkbGeometryCollection = 7
}
enum wkbByteOrder
{
  wkbXDR = 0,     // Big Endian
  wkbNDR = 1      // Little Endian
}
Point
{
  double  x;
  double  y;
}
```

П1.2.2. Объект *LineString*

Объект `LineString` в WKT-формате представляется в виде

```
LineString(X1 Y1, X2 Y2, ...)
```

и может включать две или более точек, образующих ломаную кривую. Например, объект `LineString(1 1, 2 3, 3 2, 4 4)` может выглядеть так, как это представлено на рис. П1.6.

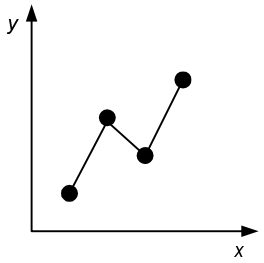


Рис. П1.6. Ломаная кривая `LineString(1 1, 2 3, 3 2, 4 4)`

Для формирования WKB-формата используют тип данных `WKBLineString` (листинг П1.4).

Листинг П1.4. Тип `WKBLineString`

```
WKBLineString
{
  byte    byteOrder;
  uint32  wkbType;           // Для LineString принимает значение 2
  uint32  numPoints;         // Количество точек в массиве points
  Point   points[numPoints]; // Массив координат
}
```

П1.2.3. Объект *Polygon*

Объект `Polygon` строится из одного или более линейных колец (замкнутых многоугольников) и представляется в WKT-формате следующим образом:

```
Polygon((X1 Y1, X2 Y2, X3 Y3, X1 Y1),(X1 Y1, X2 Y2, X3 Y3, X1 Y1),...)
```

Например, объект `Polygon((0 0, 10 0, 10 10, 0 10, 0 0), (5 5, 7 5, 7 7, 5 7, 5 5))` может выглядеть так, как это представлено на рис. П1.7.

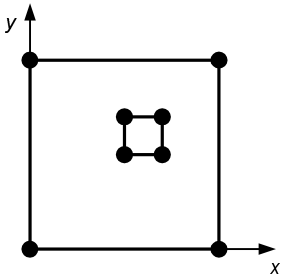


Рис. П1.7. Объект `Polygon((0 0, 10 0, 10 0, 0 10, 0 0), (5 5, 7 5, 7 7, 5 7, 5 5))`

Для формирования WKB-формата объекта `Polygon` используют тип данных `WKBPolygon` (листинг П1.5).

Листинг П1.5. Тип `WKBPolygon`

```
WKBPolygon
{
  byte       byteOrder;
  uint32     wkbType;          // Для Polygon принимает значение 3
  uint32     numRings;         // Количество точек в массиве rigngs
  LinearRing rings[numRings]; // Массив линейных колец
}
```

Для линейных колец, из которых строится объект `Polygon`, в прикладных библиотеках используется тип `LinearRing`, представляющий собой набор точек (листинг П1.6).

Листинг П1.6. Тип `LinearRing`

```
LinearRing
{
  uint32  numPoints;
  Point   points[numPoints];
}
```

П1.2.4. Объект *MultiPoint*

Коллекционный объект `MultiPoint`, предназначенный для хранения не связанных друг с другом точек, представляется в WKT-формате следующим образом: `MultiPoint(X1 Y1, X2 Y2, ...)`. Например, объект `MultiPoint(10 10, 20 20, 60 60)` может выглядеть так, как это представлено на рис. П1.8.

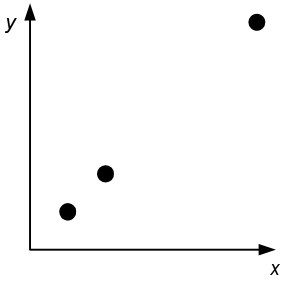


Рис. П1.8. Объект `MultiPoint(10 10, 20 20, 60 60)`

Для формирования WKB-формата объекта `MultiPoint` используют тип данных `WKBMultiPoint` (листинг П1.7).

Листинг П1.7. Тип `WKBMultiPoint`

```
WKBMultiPoint
{
  byte       byteOrder;
  uint32     wkbType;          // Для MultiPoint принимает значение 4
  uint32     num_wkbPoints;    // Количество точек в массиве WKBPoints
  WKBPoint   WKBPoints[num_wkbPoints]; // Массив точек
}
```

П1.2.5. Объект *MultiLineString*

Коллекционный объект `MultiLineString` предназначен для хранения совокупности ломаных кривых и представляется в WKT-формате следующим образом:

```
MultiLineString((X1 Y1, X2 Y2, ...), ( X1 Y1, X2 Y2, ...), ...)
```

Например, объект `MultiLineString((1 1, 2 3, 3 2, 4 4), (1 2, 2 4, 3 3, 4 5), (1 3, 2 5, 3 4, 4 6))` может выглядеть так, как это представлено на рис. П1.9.

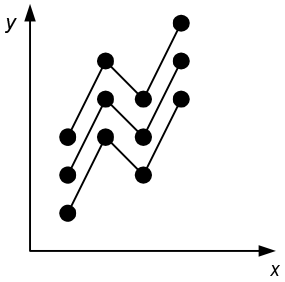


Рис. П1.9. Объект `MultiLineString((1 1, 2 3, 3 2, 4 4), (1 2, 2 4, 3 3, 4 5), (1 3, 2 5, 3 4, 4 6))`

Для формирования WKB-формата объекта `MultiLineString` используют тип данных `WKBMultiLineString` (листинг П1.8).

Листинг П1.8. Тип `WKBMultiLineString`

```
WKBMultiLineString
{
  byte          byteOrder;
  uint32        wkbType;  // Для MultiLineString принимает значение 5
  uint32        num_wkbLineStrings; // Количество точек в массиве
                                    // WKBLineStrings
  WKBLineString WKBLineStrings[num_wkbLineStrings]; // Массив кривых
}
```

П1.2.6. Объект *MultiPolygon*

Коллекционный объект `MultiPolygon` предназначен для хранения множества многоугольников и представляется в WKT-формате следующим образом:

```
MultiPolygon(((X1 Y1, X2 Y2, X3 Y3, X4 Y4, X1 Y1),
              (X1 Y1, X2 Y2, X3 Y3, X4 Y4, X1 Y1)),
              (X1 Y1, X2 Y2, X3 Y3, X4 Y4, X1 Y1),
              (X1 Y1, X2 Y2, X3 Y3, X4 Y4, X1 Y1))
```

Например, объект `MultiPolygon(((0 0, 10 0, 10 0, 0 10, 0 0)), ((5 5, 7 5, 7 7, 5 7, 5 5)))` может выглядеть так, как это представлено на рис. П1.7.

> ***Замечание***
>
> Следует иметь в виду, что даже если многоугольник в объекте `Polygon` один, его все равно нужно помещать в дополнительные круглые скобки.

Для формирования WKB-формата объекта `MultiPolygon` используют тип данных `WKBMultiPolygon` (листинг П1.9).

Листинг П1.9. Тип `WKBMultiPolygon`

```
WKBMultiPolygon
{
  byte       byteOrder;
  uint32     wkbType;    // Для MultiPolygon принимает значение 6
  uint32     num_wkbPolygons; // Количество точек в массиве
                              // wkbPolygons
  WKBPolygon wkbPolygons[num_wkbPolygons]; // Массив многоугольников
}
```

П1.2.7. Объект *GeometryCollection*

Коллекционный объект `GeometryCollection` предназначен для хранения произвольных геометрических объектов. Например, запись в WKT-формате `GeometryCollection(Point(10 10), Point(30 30), LIEARSTRING(15 15, 25 25))` хранит две точки и линию (рис. П1.10).

> ***Замечание***
>
> Важно отметить, что в отличие от других коллекционных типов, следует явно прописывать, к какому из классов относится элемент коллекции: точке (`Point`), ломаной кривой (`LINEARSTRING`) или многоугольникам (`Polygon`).

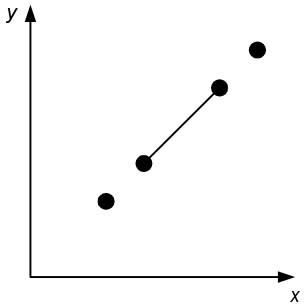


Рис. П1.10. Объект `GeometryCollection(Point(10 10), Point(30 30), LIEARSTRING(15 15, 25 25))`

Для формирования WKB-формата объекта `GeometryCollection` используют тип данных `WKBGeometryCollection` (листинг П1.10).

Листинг П1.10. Тип `WKBGeometryCollection`

```
WKBGeometryCollection
{
  byte       byte_order;
  uint32     wkbType;    // Для GeometryCollection принимает значение 7
  uint32     num_wkbGeometries; // Количество объектов в коллекции
  WKBGeometry wkbGeometries[num_wkbGeometries]; // Массив элементов
}
```

Тип `WKBGeometryCollection` строится аналогично другим типам и содержит массив элементов коллекции. Каждый элемент этого массива имеет тип `WKBGeometry` (листинг П1.11).

Листинг П1.11. Тип `WKBGeometry`

```
WKBGeometry
{
  union
  {
    WKBPoint                point;
    WKBLineString           linestring;
    WKBPolygon              polygon;
    WKBGeometryCollection   collection;
    WKBMultiPoint           mpoint;
    WKBMultiLineString      mlinestring;
    WKBMultiPolygon         mpolygon;
  }
}
```

П1.3. Работа с геометрическими элементами в MySQL

СУБД MySQL допускает в рамках пространственных расширений создание столбцов со следующими геометрическими типами:

- `GEOMETRY`;
- `Point`;
- `LineString`;
- `Polygon`;
- `MultiPoint`;
- `MultiLineString`;
- `MultiPolygon`;
- `GeometryCollection`.

Первые четыре типа списка позволяют хранить по одному объекту, причем тип `GEOMETRY` может хранить любой объект (точку, ломаную кривую или многоугольник). Последние четыре типа позволяют хранить коллекции элементов, причем в столбце `GeometryCollection` можно хранить смешанные коллекции. В листинге П1.12 создается таблица `geom`, содержащая два столбца, первый из которых принимает тип `GEOMETRY`, а второй — `Point`.

> ***Замечание***
>
> Удаление и редактирование столбцов с геометрическим типом данных можно проводить точно так же, как и для остальных столбцов — при помощи оператора `ALTER TABLE` *(см. главу 13)*.

Листинг П1.12. Создание таблицы с геометрическими типами

```
mysql> CREATE TABLE geom (g GEOMETRY, p Point);
mysql> DESCRIBE geom;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| g     | geometry | YES  |     | NULL    |       |
| p     | point    | YES  |     | NULL    |       |
+-------+----------+------+-----+---------+-------+
```

Как видно из листинга П1.12, при просмотре структуры таблицы при помощи оператора `DESCRIBE` (или `SHOW COLUMNS`) тип столбца отображается корректно. Однако просто поместить в столбцы объекты в форматах WKT или WKB не получится (листинг П1.13).

Листинг П1.13. Ошибочная вставка геометрических типов

```
mysql> INSERT INTO geom VALUES('Point(1 1)', 'Point(1 1)');
ERROR 1416: Cannot get geometry object from data you send to the GEOMETRY
field
```

В СУБД MySQL существуют специальные встроенные функции, которые преобразуют геометрические элементы в форматах WKT и WKB в их внутреннее представление.

> ***Замечание***
>
> Забегая вперед, отметим, что значения геометрических элементов также невозможно просмотреть непосредственно в результирующей таблице оператора `SELECT` — для этого потребуются специальные преобразования.

П1.3.1. Функция *PointFromText()*

Функция `PointFromText()` создает объект типа `Point` и имеет следующий синтаксис:

```
PointFromText(wkt[, srid])
```

Функция принимает в качестве первого параметра *`wkt`* строку с определением объекта `Point` в WKT-формате, а в качестве второго необязательного параметра *`srid`* — идентификатор пространственной системы координат (если он не задан, то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат). Например, вставка нового значения в таблицу `geom` (листинг П1.12) двух точек `Point(1 1)` и `Point(2 2)` может выглядеть так, как это представлено в листинге П1.14.

Замечание

Так как тип `GEOMETRY` может принимать любой не коллекционный геометрический элемент, в него можно помещать в том числе и объект точки (`Point`).

Листинг П1.14. Использование функции `PointFromText()`

```
mysql> INSERT INTO geom
    -> VALUES(PointFromText('Point(1 1)'), PointFromText('Point(1 1)'));
mysql> SELECT * FROM geom;
+--------------------------+--------------------------+
| g                        | p                        |
+--------------------------+--------------------------+
|                          |                          |
+--------------------------+--------------------------+
```

Как видно из листинга П1.14, извлечение записей из таблицы при помощи обычного оператора `SELECT` ни к чему не приводит. Для того чтобы отобразить геометрические элементы в текстовом формате WKT или бинарном формате WKB, следует преобразовать их из внутреннего представления при помощи функций `AsText()` и `AsBinary()`, соответственно (листинг П1.15).

Листинг П1.15. Использование функций `AsText()` и `AsBinary()`

```
mysql> SELECT AsText(g), AsText(p) FROM geom;
+------------+------------+
| AsText(g)  | AsText(p)  |
+------------+------------+
| Point(1 1) | Point(1 1) |
+------------+------------+
mysql> SELECT AsBinary(g), AsBinary(p) FROM geom;
+----------------------+----------------------+
| AsBinary(g)          | AsBinary(p)          |
+----------------------+----------------------+
| ☺☺                   | ☺☺                   |
+----------------------+----------------------+
```

Как видно из листинга П1.15, с бинарными данными не очень удобно работать в интерактивных клиентах, зато они очень удобны в прикладных C-программах, т. к. позволяют сохранять и извлекать геометрические объекты без перевода их в текстовый формат WKT.

В качестве координат `X` и `Y` в объекте `Point` и во всех остальных объектах могут выступать не только целые числа, но и дробные (листинг П1.16).

Листинг П1.16. Использование дробных чисел для построения объекта `Point`

```
mysql> INSERT INTO geom VALUES(PointFromText('Point(1.5 1.1)'),
    -> PointFromText('Point(1.0 1.1)'));
mysql> SELECT AsText(g), AsText(p) FROM geom;
+---------------+--------------+
| AsText(g)       | AsText(p)    |
+---------------+--------------+
| Point(1.5 1.1) | Point(1 1.1) |
+---------------+--------------+
```

П1.3.2. Функция *PointFromWKB()*

Функция `PointFromWKB()` создает объект типа `Point` и имеет следующий синтаксис:

```
PointFromWKB(wkb[, srid])
```

Функция принимает в качестве первого параметра *wkb* бинарную последовательность, определяющую объект `Point` в WKB-формате, а в качестве второго необязательного параметра *srid* — идентификатор пространственной системы координат (если он не задан, то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат). Например, вставка нового значения в таблицу `geom` (см. листинг П1.12) двух точек `Point(1 1)` и `Point(1 1)` может выглядеть так, как это представлено в листинге П1.17.

Листинг П1.17. Использование функции `PointFromWKB()`

```
mysql> INSERT INTO geom
VALUES(PointFromWKB(0x0101000000000000000000F03F000000000000F03F),
    -> PointFromWKB(0x0101000000000000000000F03F000000000000F03F));
mysql> SELECT AsText(g), AsText(p) FROM geom;
+------------+------------+
| AsText(g)   | AsText(p)   |
+------------+------------+
| Point(1 1) | Point(1 1) |
| Point(1 1) | Point(1 1) |
+------------+------------+
```

П1.3.3. Функция *Point()*

Функция `Point()` создает объект типа `Point` и имеет следующий синтаксис:

```
Point(x, y)
```

Функция принимает в качестве первого аргумента координату по оси абсцисс, а в качестве второго — координату по оси ординат. В результате функция возвращает `BLOB`-значение, содержащее объект `Point` в бинарном WKB-формате (листинг П1.18).

> ***Замечание***
>
> Функция `Point()` не входит в стандарт OpenGIS и является расширением MySQL. Это означает, что функция поддерживается в рамках СУБД MySQL, но SQL-код, созданный с использованием `Point()`, может быть несовместим с другими СУБД, реализующими пространственные расширения.

Листинг П1.18. Использование функции `Point()`

```
mysql> SELECT Point(0,0);
+----------------------+
| Point(0,0)           |
+----------------------+
| ☺☺                   |
+----------------------+
```

Результат функции `Point()` удобно использовать в качестве первого аргумента функции `PointFromWKB()` (листинг П1.19).

Листинг П1.19. Совместное использование функций `PointFromWKB()` и `Point()`

```
mysql> INSERT INTO geom
    -> VALUES(PointFromWKB(Point(1,1)), PointFromWKB(Point(3,3)));
mysql> SELECT AsText(g), AsText(p) FROM geom;
+------------+------------+
| AsText(g)  | AsText(p)  |
+------------+------------+
| Point(1 1) | Point(3 3) |
+------------+------------+
```

П1.3.4. Функция *LineFromText()*

Функция `LineFromText()` создает объект типа `LineString` и имеет следующий синтаксис:

```
LineFromText(wkt[, srid])
```

Функция принимает в качестве первого параметра *wkt* строку с определением объекта LineString в WKT-формате, а в качестве второго необязательного параметра *srid* — идентификатор пространственной системы координат (если он не задан, то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат). Так как первое значение таблицы geom (см. листинг П1.12) имеет тип GEOMETRY, в него можно вставить при помощи функции LineFromText() ломаную кривую (листинг 1.20).

> ***Замечание***
>
> Функция LineFromText() имеет синоним LineStringFromText().

Листинг П1.20. Использование функции LineFromText()

```
mysql> INSERT INTO geom VALUES(LineFromText('LineString(2 2, 3 4, 5 6)'),
                               PointFromText('Point(0 0)'));
mysql> SELECT AsText(g) FROM geom;
+-------------------------+
| AsText(g)               |
+-------------------------+
| LINESTRING(2 2,3 4,5 6) |
+-------------------------+
```

П1.3.5. Функция *LineFromWKB()*

Функция LineFromWKB() создает объект типа LineString и имеет следующий синтаксис:

```
LineFromWKB(wkb[, srid])
```

Функция принимает в качестве первого параметра *wkb* бинарную последовательность, определяющую объект LineString в WKB-формате, а в качестве второго необязательного параметра *srid* — идентификатор пространственной системы координат (если он не задан, то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат).

> ***Замечание***
>
> Функция LineFromWKB() имеет синоним LineStringFromWKB().

П1.3.6. Функция *LineString()*

Функция LineString() создает объект типа LineString и имеет следующий синтаксис:

```
LineString(p1, p2, ...)
```

Функция принимает один или более аргументов, в качестве которых выступают WKB-значения точек, которые удобно формировать при помощи функций `Point()` (см. листинг П1.18). В качестве результата функция возвращает `BLOB`-значение, содержащее объект `LineString` в бинарном WKB-формате (листинг П1.21).

> ***Замечание***
>
> Функция `LineString()` не входит в стандарт OpenGIS и является расширением MySQL. Это означает, что функция поддерживается в рамках СУБД MySQL, но SQL-код, созданный с использованием `LineString()`, может быть несовместим с другими СУБД, реализующими пространственные расширения.

Листинг П1.21. Использование функции `LineString()`

```
mysql> SELECT LineString(Point(1,1),Point(2,3),Point(3,2));
+-----------------------------------------------------------+
| LineString(Point(1,1),Point(2,3),Point(3,2))              |
+-----------------------------------------------------------+
| ☺☻                                                        |
+-----------------------------------------------------------+
```

Результат функции `LineString()` удобно использовать в качестве первого аргумента функции `LineFromWKB()` (листинг П1.22).

Листинг П1.22. Совместное использование функций `LineString` и `LineFromWKB`

```
mysql> INSERT INTO geom
    -> VALUES(LineFromWKB(LineString(Point(1,1),Point(2,3),Point(3,2))),
    ->        PointFromWKB(Point(3,3)));
mysql> SELECT AsText(g), AsText(p) FROM geom;
+------------------------+------------+
| AsText(g)              | AsText(p)  |
+------------------------+------------+
| LineString(1 1,2 3,3 2) | Point(3 3) |
+------------------------+------------+
```

Если хотя бы один из параметров функции `LineString()` не является WKB-значением объекта `Point`, функция возвращает `NULL` (листинг П1.23).

Листинг П1.23. Ошибочный список аргументов функции `LineString()`

```
mysql> SELECT LineString(Point(1,1),Point(2,3),45);
+-------------------------------------+
| LineString(Point(1,1),Point(2,3),45) |
+-------------------------------------+
| NULL                                |
+-------------------------------------+
```

П1.3.7. Функция *PolyFromText()*

Функция `PolyFromText()` создает объект типа `Polygon` и имеет следующий синтаксис:

```
PolyFromText(wkt[, srid])
```

Функция принимает в качестве первого параметра *wkt* строку с определением объекта `Polygon` в WKT-формате, а в качестве второго необязательного параметра *srid* — идентификатор пространственной системы координат (если он не задан, то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат). Так как первое значение таблицы `geom` (см. листинг П1.12) имеет тип `GEOMETRY`, в него можно вставить при помощи функции `PolyFromText()` многоугольник (листинг П1.24).

> ***Замечание***
>
> Функция `PolyFromText()` имеет синоним `PolygonFromText()`.

Листинг П1.24. Использование функции `PolyFromText()`

```
mysql> INSERT INTO geom
    -> VALUES(PolygonFromText('Polygon((0 0, 10 0, 10 10, 0 10, 0 0),
    ->                                 (5 5, 7 5, 7 7, 5 5))'),
    ->        PointFromText('Point(0 0)'));
mysql> SELECT AsText(g) FROM geom;
+-------------------------------------------------+
| AsText(g)                                       |
+-------------------------------------------------+
| Polygon((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 5)) |
+-------------------------------------------------+
```

П1.3.8. Функция *PolyFromWKB()*

Функция `PolyFromWKB()` создает объект типа `Polygon` и имеет следующий синтаксис:

```
PolyFromWKB(wkb[, srid])
```

Функция принимает в качестве первого параметра *wkb* бинарную последовательность, определяющую объект Polygon в WKB-формате, а в качестве второго необязательного параметра *srid* — идентификатор пространственной системы координат (если он не задан, то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат).

> ***Замечание***
>
> Функция PolyFromWKB() имеет синоним PolygonFromWKB().

П1.3.9. Функция *Polygon()*

Функция Polygon() создает объект типа Polygon и имеет следующий синтаксис:

```
Polygon(ls1, ls2,...)
```

Функция принимает один или более аргументов, в качестве которых выступают WKB-значения ломаных кривых, которые удобно формировать при помощи функций Polygon() (см. листинг П1.21). В результате функция возвращает BLOB-значение, содержащее объект Polygon в бинарном WKB-формате, которое удобно использовать в качестве первого аргумента функции PolyFromWKB() (листинг П1.25).

> ***Замечание***
>
> Функция Polygon() не входит в стандарт OpenGIS и является расширением MySQL. Это означает, что функция поддерживается в рамках СУБД MySQL, но SQL-код, созданный с использованием Polygon(), может быть несовместим с другими СУБД, реализующими пространственные расширения.

Листинг П1.25. Использование функции Polygon()

```
mysql> INSERT INTO geom
    -> VALUES(PolyFromWKB(Polygon(LineString(
    ->                                    Point(1,1),
    ->                                    Point(2,3),
    ->                                    Point(3,2),
    ->                                    Point(1,1)))),
    ->        PointFromWKB(Point(3,3)));
mysql> SELECT AsText(g), AsText(p) FROM geom;
+--------------------------+------------+
| AsText(g)                | AsText(p)  |
+--------------------------+------------+
| Polygon((1 1,2 3,3 2,1 1)) | Point(3 3) |
+--------------------------+------------+
```

Если хотя бы один из параметров функции `Polygon()` не является WKB-значением объекта `LineString`, функция возвращает `NULL`. Более того, объект `LineString` обязательно должен быть замкнут, в противном случае функция также возвращает `NULL` (листинг П1.26).

Листинг П1.26. Ошибочный список аргументов функции `Polygon()`

```
mysql> INSERT INTO geom
    -> VALUES(PolyFromWKB(Polygon(LineString(
    ->                                    Point(1,1),
    ->                                    Point(2,3),
    ->                                    Point(3,2)))),
    ->        PointFromWKB(Point(3,3)));
mysql> SELECT AsText(g), AsText(p) FROM geom;
+--------------------------+------------+
| AsText(g)                | AsText(p)  |
+--------------------------+------------+
| NULL                     | Point(3 3) |
+--------------------------+------------+
```

П1.3.10. Функция *GeomFromText()*

Функция `GeomFromText()` предназначена для создания любого из объектов `Point`, `LineString` или `Polygon` и имеет следующий синтаксис:

```
GeomFromText(wkt[, srid])
```

Функция принимает в качестве первого параметра *wkt* строку с определением геометрического объекта в WKT-формате, а в качестве второго необязательного параметра *srid* — идентификатор пространственной системы координат (если он не задан, то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат). В листинге П1.27 в таблицу `geom` (см. листинг П1.12) при помощи функции `GeomFromText()` вставляется запись, первое значение которой содержит ломаную кривую, а второе — точку.

> ***Замечание***
>
> Функция `GeomFromText()` имеет синоним `GeometryStringFromText()`.

Листинг П1.27. Использование функции `GeomFromText()`

```
mysql> INSERT INTO geom
    -> VALUES(GeomFromText('LineString(1 1, 2 3, 3 5, 4 1)'),
    ->        GeomFromText('Point(0 0)'));
```

```
mysql> SELECT AsText(g), AsText(p) FROM geom;
+-------------------------------------------------+------------+
| AsText(g)                                       | AsText(p)  |
+-------------------------------------------------+------------+
| LineString(1 1,2 3,3 5,4 1)                     | Point(0 0) |
+-------------------------------------------------+------------+
```

П1.3.11. Функция *GeomFromWKB()*

Функция `GeomFromWKB()` предназначена для создания любого из объектов `Point`, `LineString` или `Polygon` и имеет следующий синтаксис:

`GeomFromWKB(`*`wkb`*`[, `*`srid`*`])`

Функция принимает в качестве первого параметра *`wkb`* бинарную последовательность, определяющую геометрический объект в WKB-формате, а в качестве второго необязательного параметра *`srid`* — идентификатор пространственной системы координат (если он не задан, то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат).

> ***Замечание***
>
> Функция `GeomFromWKB()` имеет синоним `GeometryFromWKB()`.

П1.3.12. Функция *MPointFromText()*

Функция `MPointFromText()` создает объект типа `MultiPoint` и имеет следующий синтаксис:

`MPointFromText(`*`wkt`*`[, `*`srid`*`])`

Функция принимает в качестве первого параметра *`wkt`* строку с определением объекта `MultiPoint` в WKT-формате, а в качестве второго необязательного параметра *`srid`* — идентификатор пространственной системы координат (если он не задан, то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат).

Создадим таблицу `multigeom`, содержащую единственное поле `GeometryCollection`, которое позволяет хранить как смешанные коллекции, так и коллекции `MultiPoint`, `MultiLineString` и `MultiPolygon` (листинг П1.28).

Листинг П1.28. Создание таблицы `multigeom`

```
mysql> CREATE TABLE multigeom (coll GeometryCollection);
mysql> DESCRIBE multigeom;
+-------+------------------+------+-----+---------+-------+
| Field | Type             | Null | Key | Default | Extra |
```

```
+-------+--------------------+------+-----+---------+-------+
| coll  | geometrycollection | YES  |     | NULL    |       |
+-------+--------------------+------+-----+---------+-------+
```

При помощи функции `MPointFromText()` в таблицу `multigeom` можно вставить объект `MultiPoint`, который представляет собой последовательность несвязанных точек (листинг П1.29).

> ***Замечание***
>
> Функция `MPointFromText()` имеет синоним `MultiPointFromText()`.

Листинг П1.29. Использование функции `MPointFromText()`

```
mysql> INSERT INTO multigeom
    -> VALUES(MPointFromText('MultiPoint(1 1, 2 2, 3 3, 4 4)'));
mysql> SELECT AsText(coll) FROM multigeom;
+----------------------------+
| AsText(coll)               |
+----------------------------+
| MultiPoint(1 1,2 2,3 3,4 4) |
+----------------------------+
```

П1.3.13. Функция *MPointFromWKB()*

Функция `MPointFromWKB()` создает объект типа `MultiPoint` и имеет следующий синтаксис:

```
MPointFromWKB(wkb[, srid])
```

Функция принимает в качестве первого параметра *wkb* бинарную последовательность, определяющую объект `MultiPoint` в WKB-формате, а в качестве второго необязательного параметра *srid* — идентификатор пространственной системы координат (если он не задан, то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат).

> ***Замечание***
>
> Функция `MPointFromWKB()` имеет синоним `MultiPointFromWKB()`.

П1.3.14. Функция *MultiPoint()*

Функция `MultiPoint()` создает объект типа `MultiPoint` и имеет следующий синтаксис:

```
MultiPoint(p1, p2, ...)
```

Функция принимает один или более аргументов, в качестве которых выступают WKB-значения точек, которые удобно формировать при помощи функций `Point()` (см. листинг П1.18). В качестве результата функция возвращает `BLOB`-значение, содержащее объект `MultiPoint` в бинарном WKB-формате, которое удобно использовать в качестве первого аргумента функции `MPointFromWKB()` (листинг П1.30).

> ***Замечание***
>
> Функция `MultiPoint()` не входит в стандарт OpenGIS и является расширением MySQL. Это означает, что функция поддерживается в рамках СУБД MySQL, но SQL-код, созданный с использованием `MultiPoint()`, может быть несовместим с другими СУБД, реализующими пространственные расширения.

Листинг П1.30. Использование функции `MultiPoint()`

```
mysql> INSERT INTO multigeom
    -> VALUES(MPointFromWKB(MultiPoint(
    ->                                 Point(1,1),
    ->                                 Point(2,3),
    ->                                 Point(3,2))));
mysql> SELECT AsText(coll) FROM multigeom;
+-------------------------+
| AsText(coll)            |
+-------------------------+
| MultiPoint(1 1,2 3,3 2) |
+-------------------------+
```

Если хотя бы один из параметров функции `MultiPoint()` не является WKB-значением объекта `Point`, функция возвращает `NULL` (листинг П1.31).

Листинг П1.31. Ошибочный список аргументов функции `MultiPoint()`

```
mysql> SELECT MultiPoint(Point(1,1),Point(2,3),'Point(1 1)');
+-----------------------------------------------+
| MultiPoint(Point(1,1),Point(2,3),'Point(1 1)') |
+-----------------------------------------------+
| NULL                                          |
+-----------------------------------------------+
```

П1.3.15. Функция *MLineFromText()*

Функция `MLineFromText()` создает объект типа `MultiLineString` и имеет следующий синтаксис:

```
MLineFromText(wkt[, srid])
```

Функция принимает в качестве первого параметра *wkt* строку с определением объекта MultiLineString в WKT-формате, а в качестве второго необязательного параметра *srid* — идентификатор пространственной системы координат (если он не задан, то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат).

В листинге П1.32 демонстрируется вставка коллекции ломаных кривых в таблицу multigeom.

> ***Замечание***
>
> Функция MLineFromText() имеет синоним MultiLineStringFromText().

Листинг П1.32. Использование функции MLineFromText()

```
mysql> INSERT INTO multigeom
    -> VALUES(MLineFromText('MultiLineString((1 1, 2 2, 3 3, 4 4),
                                             (0 1,0 6))'));
mysql> SELECT AsText(coll) FROM multigeom;
+-------------------------------------------+
| AsText(coll)                              |
+-------------------------------------------+
| MultiLineString((1 1,2 2,3 3,4 4),(0 1,0 6)) |
+-------------------------------------------+
```

П1.3.16. Функция *MLineFromWKB()*

Функция MLineFromWKB() создает объект типа MultiLineString и имеет следующий синтаксис:

```
MPointFromWKB(wkb[, srid])
```

Функция принимает в качестве первого параметра *wkb* бинарную последовательность, определяющую объект MultiLineString в WKB-формате, а в качестве второго необязательного параметра *srid* — идентификатор пространственной системы координат (если он не задан, то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат).

> ***Замечание***
>
> Функция MLineFromWKB() имеет синоним MultiLineStringFromWKB().

П1.3.17. Функция *MultiLineString()*

Функция MultiLineString() создает объект типа MultiLineString и имеет следующий синтаксис:

```
MultiLineString(ls1, ls2,...)
```

Функция принимает один или более аргументов, в качестве которых выступают WKB-значения ломаных кривых, которые удобно формировать при помощи функций `LineString()` (см. листинг П1.21). В результате функция возвращает `BLOB`-значение, содержащее объект `MultiLineString` в бинарном WKB-формате, которое удобно использовать в качестве первого аргумента функции `MLineFromWKB()` (листинг П1.33).

> ***Замечание***
>
> Функция `MultiLineString()` не входит в стандарт OpenGIS и является расширением MySQL. Это означает, что функция поддерживается в рамках СУБД MySQL, но SQL-код, созданный с использованием `MultiLineString()`, может быть несовместим с другими СУБД, реализующими пространственные расширения.

Листинг П1.33. Использование функции `MultiLineString()`

```
mysql> INSERT INTO multigeom
    -> VALUES(MLineFromWKB(
    ->              MultiLineString(
    ->                      LineString(Point(1,1),Point(2,3),Point(3,2)),
    ->                      LineString(Point(10,10),Point(20,20)))));
mysql> SELECT AsText(coll) FROM multigeom;
+-----------------------------------------+
| AsText(coll)                            |
+-----------------------------------------+
| MultiLineString((1 1,2 3,3 2),(10 10,20 20)) |
+-----------------------------------------+
```

Если хотя бы один из параметров функции `MultiLineString()` не является WKB-значением объекта `LineString`, функция возвращает `NULL`.

П1.3.18. Функция *MPolyFromText()*

Функция `MPolyFromText()` создает объект типа `MultiPolygon` и имеет следующий синтаксис:

```
MPolyFromText(wkt[, srid])
```

Функция принимает в качестве первого параметра *`wkt`* строку с определением объекта `MultiPolygon` в WKT-формате, а в качестве второго необязательного параметра *`srid`* — идентификатор пространственной системы координат (если он не задан, то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат).

В листинге П1.34 демонстрируется вставка коллекции многоугольников в таблицу `multigeom`.

Замечание

Функция `MPolyFromText()` имеет синоним `MultiPolygonFromText()`.

Листинг П1.34. Использование функции `MPolyFromText()`

```
mysql> INSERT INTO multigeom
    -> VALUES(
    -> MPolyFromText('MultiPolygon(((1 1, 2 4, 3 3, 4 4, 1 1)))'));
mysql> SELECT AsText(coll) FROM multigeom;
+-------------------------------------------+
| AsText(coll)                              |
+-------------------------------------------+
| MultiPolygon(((1 1,2 4,3 3,4 4,1 1)))     |
+-------------------------------------------+
```

П1.3.19. Функция *MPolyFromWKB()*

Функция `MPolyFromWKB()` создает объект типа `MultiPolygon` и имеет следующий синтаксис:

```
MPolyFromWKB(wkb[, srid])
```

Функция принимает в качестве первого параметра *wkb* бинарную последовательность, определяющую объект `MultiPolygon` в WKB-формате, а в качестве второго необязательного параметра *srid* — идентификатор пространственной системы координат (если он не задан, то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат).

Замечание

Функция `MPolyFromWKB()` имеет синоним `MultiPolygonFromWKB()`.

П1.3.20. Функция *MultiPolygon()*

Функция `MultiPolygon()` создает объект типа `MultiPolygon` и имеет следующий синтаксис:

```
MultiPolygon(ls1, ls2,...)
```

Функция принимает один или более аргументов, в качестве которых выступают WKB-значения многоугольников, которые удобно формировать при помощи функций `Polygon()` (см. листинг П1.25). В результате функция возвращает `BLOB`-значение, содержащее объект `MultiPolygon` в бинарном WKB-формате, которое удобно использовать в качестве первого аргумента функции `MPolyFromWKB()` (листинг П1.35).

Замечание

Функция `MultiPolygon()` не входит в стандарт OpenGIS и является расширением MySQL. Это означает, что функция поддерживается в рамках СУБД MySQL, но SQL-код, созданный с использованием `MultiPolygon()`, может быть несовместим с другими СУБД, реализующими пространственные расширения.

Листинг П1.35. Использование функции `MultiPolygon()`

```
mysql> SET @poly1 = Polygon(LineString(
    ->                                Point(1,1),
    ->                                Point(2,3),
    ->                                Point(3,2),
    ->                                Point(1,1)));
mysql> SET @poly2 = Polygon(LineString(
    ->                                Point(10,10),
    ->                                Point(20,30),
    ->                                Point(30,20),
    ->                                Point(10,10)));
mysql> INSERT INTO multigeom
    -> VALUES(MLineFromWKB(MultiPolygon(@poly1,@poly2)));
mysql> SELECT AsText(coll) FROM multigeom;
+-------------------------------------------------------------+
| AsText(coll)                                                |
+-------------------------------------------------------------+
| MultiPolygon(((1 1,2 3,3 2,1 1)),((10 10,20 30,30 20,10 10))) |
+-------------------------------------------------------------+
```

В листинге П1.35 объекты `Polygon` сохраняются в пользовательские переменные `@poly1` и `@poly2` для того, чтобы конечный `INSERT`-оператор был менее громоздким. Если хотя бы один из параметров функции `MultiPolygon()` не является WKB-значением объекта `Polygon`, функция возвращает `NULL`.

П1.3.21. Функция *GeomCollFromText()*

Функция `GeomCollFromText()` создает объект типа `GeometryCollection` и имеет следующий синтаксис:

`GeomCollFromText(`*`wkt`*`[, `*`srid`*`])`

Функция принимает в качестве первого параметра *`wkt`* строку с определением произвольной коллекции в WKT-формате, а в качестве второго необязательного параметра *`srid`* — идентификатор пространственной системы координат (если он не задан,

то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат).

В листинге П1.36 демонстрируется вставка коллекции из точек и ломаных кривых в таблицу `multigeom`.

Замечание

Функция `GeomCollFromText()` имеет синоним `GeometryCollectionFromText()`.

Листинг П1.36. Использование функции `GeomCollFromText()`

```
mysql> INSERT INTO multigeom
    -> VALUES(GeomCollFromText('GeometryCollection(Point(1 1),
    ->                          Point(3 3),
    ->                          LineString(1 1, 2 5, 3 3))'));
mysql> SELECT AsText(coll) FROM multigeom;
+-------------------------------------------------------------------+
| AsText(coll)                                                      |
+-------------------------------------------------------------------+
| GeometryCollection(Point(1 1),Point(3 3),LineString(1 1,2 5,3 3)) |
+-------------------------------------------------------------------+
```

П1.3.22. Функция *GeomCollFromWKB()*

Функция `GeomCollFromWKB()` создает объект типа `GeometryCollection` и имеет следующий синтаксис:

```
GeomCollFromWKB(wkb[, srid])
```

Функция принимает в качестве первого параметра *wkb* бинарную последовательность, определяющую объект `GeometryCollection` в WKB-формате, а в качестве второго необязательного параметра *srid* — идентификатор пространственной системы координат (если он не задан, то по умолчанию параметр принимает значение 0, соответствующее декартовой системе координат).

Замечание

Функция `GeomCollFromWKB()` имеет синоним `GeometryCollectionFromWKB()`.

П1.3.23. Функция *GeometryCollection()*

Функция `GeometryCollection()` создает объект типа `GeometryCollection` и имеет следующий синтаксис:

```
GeometryCollection(g1, g2,...)
```

Функция принимает один или более аргументов, в качестве которых выступают WKB-значения геометрических элементов (как отдельных, так и коллекций). В результате функция возвращает BLOB-значение, содержащее объект GeometryCollection в бинарном WKB-формате, которое удобно использовать в качестве первого аргумента функции GeomCollFromWKB() (листинг П1.37).

> ***Замечание***
>
> Функция GeometryCollection() не входит в стандарт OpenGIS и является расширением MySQL. Это означает, что функция поддерживается в рамках СУБД MySQL, но SQL-код, созданный с использованием GeometryCollection(), может быть несовместим с другими СУБД, реализующими пространственные расширения.

Листинг П1.37. Использование функции GeometryCollection()

```
mysql> SET @g1 = Point(1,1);
mysql> SET @g2 = Point(2,4);
mysql> SET @g3 = Point(3,2);
mysql> SET @g4 =
    -> Polygon(LineString(Point(1,1),Point(2,3),Point(3,2),Point(1,1)));
mysql> INSERT INTO multigeom
    -> VALUES(GeomCollFromWKB(GeometryCollection(@g1,@g2,@g3,@g4)));
mysql> SELECT AsText(coll) FROM multigeom;
+-------------------------------------------------------------------+
| AsText(coll)                                                      |
+-------------------------------------------------------------------+
| GeometryCollection(Point(1 1),Point(2 4),Point(3 2),Polygon((1 1,2 3,3
2,1 1))) |
+-------------------------------------------------------------------+
```

В листинге П1.37 геометрические объекты сохраняются в пользовательские переменные @g1, @g2, @g3 и @g4 для того, чтобы конечный INSERT-оператор был менее громоздким. Если хотя бы один из параметров функции GeometryCollection() не является WKB-значением, функция возвращает NULL.

П1.4. Общие функции геометрических объектов

Помимо функций, при помощи которых создаются геометрические объекты, СУБД MySQL содержит широкий набор функций, позволяющих возвращать свойства геометрических объектов. Именно им и посвящен данных раздел.

Замечание

MySQL не реализует не полный набор функций пространственных расширений, которые определяет спецификация OpenGIS. Разработчики обещают исправить эту ситуацию в ближайших версиях.

П1.4.1. Функция *Dimension()*

Функция `Dimension()` возвращает измерение геометрического объекта и имеет следующий синтаксис:

`Dimension(`*`geometry`*`)`

Функция принимает геометрический объект типа *`geometry`* и возвращает одно из следующих значений:

- –1 — для пустого геометрического объекта;
- 0 — для геометрического объекта, не имеющего длины или области (данное значение принимают объекты классов `Point`, `MultiPoint`);
- 1 — для геометрического объекта с отличной от нуля длиной и равной нулю областью (данное значение принимают объекты классов `Line`, `LineString`, `MultiLineString`);
- 2 — для геометрических объектов с ненулевой областью (данное значение принимают объекты классов `Poligon`, `MultiPoligon`).

Пример использования функции `Dimension()` приводится в листинге П1.38.

Замечание

Если в качестве аргумента функции `Dimension()` передается значение, не являющееся геометрическим объектом, функция возвращает `NULL`. Если в качестве аргумента функции выступает коллекция `GeometryCollection`, функция `Dimension()` возвращает максимальное измерение, которое встречается среди объектов коллекции.

Листинг П1.38. Использование функции `Dimension()`

```
mysql> SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+-----------------------------------------------+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+
```

П1.4.2. Функция *Envelope()*

Функция `Envelope()` возвращает минимальный ограничивающий прямоугольник (MBR) для геометрического объекта и имеет следующий синтаксис:

`Envelope(`*`geometry`*`)`

Функция принимает геометрический объект типа *geometry* и возвращает ограничивающий прямоугольник в виде объекта `Polygon`. Например, для линии `LineString(1 1, 2 3, 3 2, 4 4)` результат работы функции `Envelope()` может выглядеть так, как это представлено в листинге П1.39.

> ***Замечание***
>
> Если в качестве аргумента функции `Envelope()` передается значение, не являющееся геометрическим объектом, функция возвращает `NULL`.

Листинг П1.39. Использование функции `Envelope()`

```
mysql> SELECT AsText(Envelope(GeomFromText(
                                    'LineString(1 1, 2 3, 3 2, 4 4)')));
+-------------------------------------------------------------------+
| AsText(Envelope(GeomFromText('LineString(1 1, 2 3, 3 2, 4 4)'))) |
+-------------------------------------------------------------------+
| Polygon((1 1,4 1,4 4,1 4,1 1))                                    |
+-------------------------------------------------------------------+
```

Графически результат из листинга П1.39 можно представить так, как это изображено на рис. П1.11.

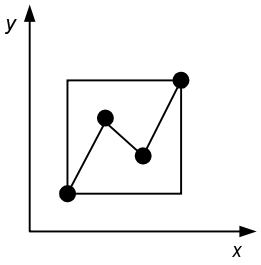


Рис. П1.11. Ограничивающий прямоугольник для ломаной кривой `LineString(1 1, 2 3, 3 2, 4 4)`

Функция `Envelope()` допускает в качестве аргумента и точки — в этом случае координаты углов ограничивающего прямоугольника будут одинаковы и равны координатам точки (листинг П1.40).

Листинг П1.40. Применение функции `Envelope()` к точке

```
mysql> SELECT AsText(Envelope(GeomFromText('Point(1 1)')));
+-------------------------------------------+
| AsText(Envelope(GeomFromText('Point(1 1)'))) |
+-------------------------------------------+
| Polygon((1 1,1 1,1 1,1 1,1 1))              |
+-------------------------------------------+
```

Наиболее интересных результатов можно добиться при определении ограничивающего прямоугольника для геометрической коллекции (листинг П1.41). Данная функция позволяет в любой момент вычислить границы карты, не зависимо от того, сколько элементов было перед этим на ней размещено и в какие координаты.

Листинг П1.41. Применение функции `Envelope()` к геометрической коллекции

```
mysql> SET @g1 = Point(1,1);
mysql> SET @g2 = Point(2,4);
mysql> SET @g3 = Point(3,3);
mysql> SET @g4 =
    -> Polygon(LineString(Point(1,1),Point(2,3),Point(3,2),Point(1,1)));
mysql> SELECT AsText(GeomCollFromWKB(
    ->                          GeometryCollection(@g1,@g2,@g3,@g4)));
+-----------------------------------------------------------------------+
| AsText(GeomCollFromWKB(GeometryCollection(@g1,@g2,@g3,@g4)))          |
+-----------------------------------------------------------------------+
| GeometryCollection(Point(1 1),Point(2 4),Point(3 3),Polygon((1 1,2 3,3
2,1 1))) |
+-----------------------------------------------------------------------+
mysql> SELECT AsText(Envelope(
    ->    GeomCollFromWKB(GeometryCollection(@g1,@g2,@g3,@g4)) )) AS pol;
+------------------------------+
| pol                          |
+------------------------------+
| Polygon((1 1,3 1,3 4,1 4,1 1)) |
+------------------------------+
```

Графически результат из листинга П1.41 можно представить так, как это изображено на рис. П1.12.

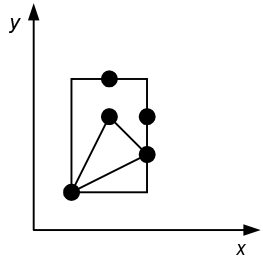


Рис. П1.12. Ограничивающий прямоугольник для ломаной кривой `GeometryCollection(Point(1 1), Point(2 4), Point(3 3), Polygon((1 1, 2 3, 3 2, 1 1)))`

П1.4.3. Функция *GeometryType()*

Функция `GeometryType()` возвращает название геометрического типа и имеет следующий синтаксис:

`GeometryType(`*geometry*`)`

Функция `GeometryType()` принимает в качестве единственного аргумента геометрический объект *geometry* и возвращает строку с названием геометрического типа объекта (листинг П1.42).

> ***Замечание***
>
> Если в качестве аргумента функции `GeometryType()` передается значение, не являющееся геометрическим объектом, функция возвращает `NULL`.

Листинг П1.42. Использование функции `GeometryType()`

```
mysql> SELECT GeometryType(GeomFromText('Point(1 1)'));
+---------------------------------------+
| GeometryType(GeomFromText('Point(1 1)')) |
+---------------------------------------+
| Point                                 |
+---------------------------------------+
mysql> SET @g1 = Point(1,1);
mysql> SET @g2 = Point(2,4);
mysql> SET @g3 = Point(3,3);
mysql> SET @g4 =
    -> Polygon(LineString(Point(1,1),Point(2,3),Point(3,2),Point(1,1)));
mysql> SELECT GeometryType(GeomCollFromWKB(
```

```
                                    GeometryCollection(@g1,@g2,@g3,@g4)));
+-----------------------------------------------------------------------+
| GeometryType(GeomCollFromWKB(GeometryCollection(@g1,@g2,@g3,@g4))) |
+-----------------------------------------------------------------------+
| GeometryCollection                                                    |
+-----------------------------------------------------------------------+
```

П1.4.4. Функция *SRID()*

Функция `SRID()` возвращает идентификатор системы координат геометрического объекта и имеет следующий синтаксис:

`SRID(`*`geometry`*`)`

Функция `SRID()` принимает в качестве единственного аргумента геометрический объект *`geometry`* и возвращает целое число, обозначающее систему координат (листинг П1.43).

> ***Замечание***
>
> Если в качестве аргумента функции `SRID()` передается значение, не являющееся геометрическим объектом, функция возвращает `NULL`.

> ***Замечание***
>
> В MySQL SRID-значение — это лишь формальность, все вычисления выполняются в декартовых координатах.

Листинг П1.43. Использование функции `SRID()`

```
mysql> SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
+---------------------------------------------+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+---------------------------------------------+
|                                         101 |
+---------------------------------------------+
```

П1.5. Функции для работы с объектом *Point*

П1.5.1. Функция *X()*

Функция `X()` возвращает координату по оси абсцисс для объекта типа `Point` и имеет следующий синтаксис:

`X(`*`point`*`)`

Функция принимает в качестве единственного параметра точку *point* и возвращает координату в виде числа с двойной точностью (листинг П1.44).

> ***Замечание***
>
> Если в качестве аргумента функции `X()` передается значение, не являющееся объектом типа `Point`, функция возвращает `NULL`.

Листинг П1.44. Использование функции `X()`

```
mysql> SELECT X(GeomFromText('Point(10.3 10.6)')) AS fst,
    ->        X(GeomFromText('Point(7 5)')) AS snd;
+------+-----+
| fst  | snd |
+------+-----+
| 10.3 |   7 |
+------+-----+
```

П1.5.2. Функция *Y()*

Функция `Y()` возвращает координату по оси ординат для объекта типа `Point` и имеет следующий синтаксис:

`Y(`*point*`)`

Функция принимает в качестве единственного параметра точку *point* и возвращает координату в виде числа с двойной точностью (листинг П1.45).

> ***Замечание***
>
> Если в качестве аргумента функции `Y()` передается значение, не являющееся объектом типа `Point`, функция возвращает `NULL`.

Листинг П1.45. Использование функции `Y()`

```
mysql> SELECT Y(GeomFromText('Point(10.3 10.6)')) AS fst,
    ->        Y(GeomFromText('Point(7 5)')) AS snd;
+------+-----+
| fst  | snd |
+------+-----+
| 10.6 |   5 |
+------+-----+
```

П1.6. Функции для работы с объектом *LineString*

Объект ломаной кривой `LineString` состоит из точек (`Point`). При помощи функций данной группы можно соединять определенные точки, подсчитывать их количество или получать длину объекта `LineString`.

П1.6.1. Функция *EndPoint()*

Функция `EndPoint()` возвращает последнюю точку ломаной кривой и имеет следующий синтаксис:

`EndPoint(`*`linestring`*`)`

Функция принимает в качестве единственного параметра ломаную кривую *`linestring`* и возвращает точку, которая является конечной для данной ломаной кривой. Пример использования функции `EndPoint()` приведен в листинге П1.46.

> ***Замечание***
>
> Если в качестве аргумента функции `EndPoint()` передается значение, не являющееся объектом типа `LineString`, функция возвращает `NULL`.

Листинг П1.46. Использование функции `EndPoint()`

```
mysql> SELECT AsText(EndPoint(GeomFromText('LineString(1 1,2 2,3 3)')));
+-------------------------------------------------------+
| AsText(EndPoint(GeomFromText('LineString(1 1,2 2,3 3)'))) |
+-------------------------------------------------------+
| Point(3 3)                                            |
+-------------------------------------------------------+
```

П1.6.2. Функция *StratPoint()*

Функция `StartPoint()` возвращает первую точку ломаной кривой и имеет следующий синтаксис:

`StartPoint(`*`linestring`*`)`

Функция принимает в качестве единственного параметра ломаную кривую *`linestring`* и возвращает точку, которая является начальной для данной ломаной кривой. Пример использования функции `StartPoint()` приведен в листинге П1.47.

> ***Замечание***
>
> Если в качестве аргумента функции `StartPoint()` передается значение, не являющееся объектом типа `LineString`, функция возвращает `NULL`.

Листинг П1.47. Использование функции `StartPoint()`

```
mysql> SELECT AsText(StartPoint(GeomFromText('LineString(1 1,2 2)')));
+-----------------------------------------------------+
| AsText(StartPoint(GeomFromText('LineString(1 1,2 2)'))) |
+-----------------------------------------------------+
| Point(1 1)                                          |
+-----------------------------------------------------+
```

П1.6.3. Функция *PointN()*

Функция `PointN()` возвращает заданную точку ломаной кривой и имеет следующий синтаксис:

`PointN(`*`linestring`*`, `*`n`*`)`

Функция принимает в качестве первого параметра ломаную кривую *`linestring`*, а в качестве второго — номер точки, который следует возвратить. При этом нумерация точек начинается с 1. Пример использования функции `StartPoint()` приведен в листинге П1.48.

> ***Замечание***
>
> Если в качестве первого аргумента функции `PointN()` передается значение, не являющееся объектом типа `LineString`, функция возвращает `NULL`.

Листинг П1.48. Использование функции `PointN()`

```
mysql> SELECT AsText(PointN(GeomFromText('LineString(1 1,2 2,3 3)'), 2));
+-----------------------------------------------------------+
| AsText(PointN(GeomFromText('LineString(1 1,2 2,3 3)'), 2)) |
+-----------------------------------------------------------+
| Point(2 2)                                                |
+-----------------------------------------------------------+
```

Если номер точки *`n`* больше максимального числа точек в ломаной кривой *`linestring`* или меньше нуля, функция `PointN()` возвращает `NULL` (листинг П1.49).

Листинг П1.49. Недопустимое значение второго параметра функции `PointN()`

```
mysql> SELECT AsText(PointN(GeomFromText('LineString(1 1,2 2,3 3)'), 7));
+-----------------------------------------------------------+
| AsText(PointN(GeomFromText('LineString(1 1,2 2,3 3)'), 7)) |
+-----------------------------------------------------------+
```

```
| NULL                                                          |
+---------------------------------------------------------------+
mysql> SELECT AsText(PointN(GeomFromText('LineString(1 1,2 2,3 3)'),-1));
+---------------------------------------------------------------+
| AsText(PointN(GeomFromText('LineString(1 1,2 2,3 3)'),-1)) |
+---------------------------------------------------------------+
| NULL                                                          |
+---------------------------------------------------------------+
```

П1.6.4. Функция *NumPoints()*

Функция `NumPoints()` возвращает число точек в ломаной кривой и имеет следующий синтаксис:

`NumPoints(`*`linestring`*`)`

Функция принимает в качестве единственного параметра ломаную кривую *`linestring`* и возвращает число точек в кривой. Пример использования функции `NumPoints()` приведен в листинге П1.50.

> ***Замечание***
>
> Если в качестве аргумента функции `NumPoints()` передается значение, не являющееся объектом типа `LineString`, функция возвращает `NULL`.

Листинг П1.50. Использование функции `NumPoints()`

```
mysql> SELECT NumPoints(GeomFromText('LineString(1 1,2 2,3 3)'));
+-------------------------------------------------+
| NumPoints(GeomFromText('LineString(1 1,2 2,3 3)')) |
+-------------------------------------------------+
|                                               3 |
+-------------------------------------------------+
```

П1.6.5. Функция *GLength()*

Функция `GLength()` возвращает длину ломаной кривой и имеет следующий синтаксис:

`GLength(`*`linestring`*`)`

Функция принимает в качестве единственного параметра ломаную кривую *`linestring`* и возвращает ее длину в виде числа двойной точности. Пример использования функции `GLenght()` приведен в листинге П1.51.

> ***Замечание***
>
> Если в качестве аргумента функции `GLength()` передается значение, не являющееся объектом типа `LineString`, функция возвращает `NULL`.

Замечание

В спецификации OpenGIS функция для вычисления длины ломаной кривой определяется как `Length()`, однако для того чтобы не возникал конфликт с одноименной встроенной функцией MySQL для подсчета количества символов в строке, она названа `GLenght()`.

Листинг П1.51. Использование функции `GLenght()`

```
mysql> SELECT GLength(GeomFromText('LineString(1 1,2 1,3 1)'));
+-------------------------------------------------+
| GLength(GeomFromText('LineString(1 1,2 1,3 1)')) |
+-------------------------------------------------+
|                                               2 |
+-------------------------------------------------+
mysql> SELECT GLength(GeomFromText('LineString(1 1,2 2,3 3)'));
+-------------------------------------------------+
| GLength(GeomFromText('LineString(1 1,2 2,3 3)')) |
+-------------------------------------------------+
|                                2.8284271247462 |
+-------------------------------------------------+
```

П1.6.6. Функция *IsClosed()*

Функция `IsClosed()` позволяет определить, является ли ломаная линия замкнутой или нет. Функция имеет следующий синтаксис:

`IsClosed(`*`linestring`*`)`

Функция принимает в качестве единственного параметра ломаную линию *`linestring`* и возвращает 1, если кривая является замкнутой, 0, если кривая не является замкнутой, и –1, если *`linestring`* равно значению `NULL`. Пример использования функции `IsClosed()` приведен в листинге П1.52.

Замечание

Если в качестве аргумента функции `IsClosed()` передается значение, не являющееся объектом типа `LineString`, функция возвращает `NULL`.

Листинг П1.52. Использование функции `IsClosed()`

```
mysql> SELECT IsClosed(GeomFromText('LineString(1 1,2 2,3 3)'));
+-------------------------------------------------+
| IsClosed(GeomFromText('LineString(1 1,2 2,3 3)')) |
+-------------------------------------------------+
```

```
|                                                 0 |
+---------------------------------------------------+
mysql> SELECT IsClosed(GeomFromText('LineString(1 1,2 2,3 3,1 1)'));
+-----------------------------------------------------+
| IsClosed(GeomFromText('LineString(1 1,2 2,3 3,1 1)')) |
+-----------------------------------------------------+
|                                                   1 |
+-----------------------------------------------------+
mysql> SELECT IsClosed(GeomFromText('Point(1 1)'));
+-------------------------------------+
| IsClosed(GeomFromText('Point(1 1)')) |
+-------------------------------------+
|                                NULL |
+-------------------------------------+
```

П1.7. Функции для работы с объектом *MultiLineString*

Функции для работы с коллекциями ломаных кривых `MultiLineString` дублируют функции для работы с единичными ломаными кривыми `LineString`. Если ломаные кривые помещены в коллекционный объект `MultiLineString`, при помощи функций данной группы можно обработать их всех сразу за один вызов.

П1.7.1. Функция *GLenght()*

Функция `GLength()` возвращает общую длину ломаных кривых в коллекции `MultiLineString` и имеет следующий синтаксис:

`GLength(`*`multilinestring`*`)`

Функция принимает в качестве единственного параметра коллекцию *`multilinestring`* и возвращает ее длину в виде числа с удвоенной точностью Пример использования функции `GLenght()` приведен в листинге П1.53.

> ***Замечание***
>
> Если в качестве аргумента функции `GLength()` передается значение, не являющееся объектом типа `MultiLineString`, функция возвращает `NULL`.

> ***Замечание***
>
> В спецификации OpenGIS данная функция называется `Length()`, однако для того чтобы не возникал конфликт с одноименной встроенной функцией MySQL для подсчета числа символов в строке, она названа `GLenght()`.

Листинг П1.53. Использование функции `GLenght()`

```
mysql> SELECT GLength(GeomFromText('MultiLineString((1 1,2 2,3 3),
    ->                                              (4 4,5 5))')) AS len;
+-----------------+
| len             |
+-----------------+
| 4.2426406871193 |
+-----------------+
```

П1.7.2. Функция *IsClosed()*

Функция `IsClosed()` позволяет определить, является ли коллекция `MultiLineString` замкнутой или нет. Функция имеет следующий синтаксис:

`IsClosed(`*`multilinestring`*`)`

Функция принимает в качестве единственного параметра коллекцию ломаных кривых *`multilinestring`* и возвращает 1, если коллекция является замкнутой, 0, если коллекция не является замкнутой, и –1, если *`multilinestring`* равно `NULL`. Пример использования функции `IsClosed()` приведен в листинге П1.54.

> ***Замечание***
>
> Коллекция считается замкнутой, если каждый элемент коллекции является замкнутым.

> ***Замечание***
>
> Если в качестве аргумента функции `IsClosed()` передается значение, не являющееся объектом типа `MultiLineString`, функция возвращает `NULL`.

Листинг П1.54. Использование функции `IsClosed()`

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT IsClosed(GeomFromText(@mls));
+----------------------------+
| IsClosed(GeomFromText(@mls)) |
+----------------------------+
| 0                          |
+----------------------------+
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3,1 1),(4 4,5 5,4 4))';
mysql> SELECT IsClosed(GeomFromText(@mls));
+----------------------------+
| IsClosed(GeomFromText(@mls)) |
+----------------------------+
| 1                          |
+----------------------------+
```

П1.8. Функции для работы с объектом *Polygon*

П1.8.1. Функция *Area()*

Функция `Area()` возвращает площадь объекта `Polygon` и имеет следующий синтаксис:

`Area(`*polygon*`)`

Функция принимает в качестве единственного параметра многоугольник *polygon* и возвращает его площадь в виде числа двойной точности. В листинге П1.55 приводится пример использования функции для вычисления площади многоугольника.

> ***Замечание***
>
> Если в качестве аргумента функции `Area()` передается значение, не являющееся объектом типа `Polygon`, функция возвращает `NULL`.

Листинг П1.55. Использование функции `Area()`

```
mysql> SET @poly = 'Polygon((0 0, 10 0, 10 10, 0 10, 0 0))';
mysql> SELECT Area(GeomFromText(@poly));
+--------------------------+
| Area(GeomFromText(@poly)) |
+--------------------------+
| 100                      |
+--------------------------+
```

Если многоугольник содержит отверстия, то площадь отверстий вычитается из суммарной площади многоугольника. Так например, для квадрата со стороной 10, содержащим отверстие со стороной 2 (рис. П1.13), функция `Area()` вернет площадь 96 (листинг П1.56).

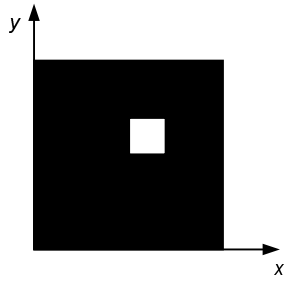


Рис. П1.13. Визуальное представление объекта `Polygon((0 0, 10 0, 10 10, 0 10, 0 0), (5 5, 7 5, 7 7, 5 7, 5 5))`

Листинг П1.56. Вычисление площади многоугольников с отверстиями

```
mysql> SET @poly = 'Polygon((0 0, 10 0, 10 10, 0 10, 0 0),
    ->                      (5 5, 7 5, 7 7, 5 7, 5 5))';
mysql> SELECT Area(GeomFromText(@poly));
+---------------------------+
| Area(GeomFromText(@poly)) |
+---------------------------+
| 96                        |
+---------------------------+
```

Вообще с функцией `Area()` следует быть достаточно аккуратным, т. к. если объект `Polygon` не является простым, результат может отличаться от ожидаемого (листинг П1.57).

Листинг П1.57. Вычисление площади сложного объекта `Polygon`

```
mysql> SET @poly = 'Polygon((0 0, 10 0, 10 10, 0 10, 0 0),
    ->                      (10 10, 20 10, 20 20, 10 20, 10 10))';
mysql> SELECT Area(GeomFromText(@poly));
+---------------------------+
| Area(GeomFromText(@poly)) |
+---------------------------+
| 0                         |
+---------------------------+
mysql> SET @poly = 'Polygon((0 0, 10 0, 10 10, 0 10, 0 0),
    ->                      (15 15, 20 15, 20 20, 15 20, 15 15))';
mysql> SELECT Area(GeomFromText(@poly));
+---------------------------+
| Area(GeomFromText(@poly)) |
+---------------------------+
| 75                        |
+---------------------------+
```

Как видно из листинга П1.57, первый многоугольник представляет собой два квадрата 10×10, которые касаются друг друга лишь в одной точке. Однако не следует думать, что площадь здесь вычисляется как пересечение. Второй объект `Polygon` в листинге П1.57 состоит из двух квадратов 10×10 и 5×5, которые вообще не пересекаются, однако результат представляет собой разность площадей многоугольников. Функция `Area()` вычисляется по следующему алгоритму: из площади первого

многоугольника вычитается площадь всех остальных многоугольников (отверстий), даже если они входят в состав первого. Результат берется по модулю, т. к. площадь не может принимать отрицательное значение.

П1.8.2. Функция *ExteriorRing()*

Функция `ExteriorRing()` возвращает внешнее кольцо объекта `Polygon` и имеет следующий синтаксис:

`ExteriorRing(`*polygon*`)`

Функция принимает в качестве единственного параметра многоугольник *polygon* и возвращает его внешнее кольцо в виде объекта `LineString`. В листинге П1.58 приводится пример использования функции.

> ***Замечание***
>
> Если в качестве аргумента функции `ExteriorRing()` передается значение, не являющееся объектом типа `Polygon`, функция возвращает `NULL`.

Листинг П1.58. Использование функции `ExteriorRing()`

```
mysql> SET @poly = 'Polygon((0 0,0 3,3 3,3 0,0 0),
    ->                      (1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT AsText(ExteriorRing(GeomFromText(@poly)));
+-----------------------------------------+
| AsText(ExteriorRing(GeomFromText(@poly))) |
+-----------------------------------------+
| LineString(0 0,0 3,3 3,3 0,0 0)           |
+-----------------------------------------+
```

Функция ориентируется на первый многоугольник в объекте `Polygon`, поэтому если один из последующих многоугольников (отверстий) выходит за границы самого первого многоугольника — это не учитывается в результате (листинг П1.59).

Листинг П1.59. Применение функции `ExteriorRing()` к сложному объекту

```
mysql> SET @poly = 'Polygon((0 0,0 3,3 3,3 0,0 0),(2 2,2 5,5 5,5 2,2
2))';
mysql> SELECT AsText(ExteriorRing(GeomFromText(@poly)));
+-----------------------------------------+
| AsText(ExteriorRing(GeomFromText(@poly))) |
+-----------------------------------------+
| LineString(0 0,0 3,3 3,3 0,0 0)           |
+-----------------------------------------+
```

В листинге П1.59 приводится пример обработки объекта `Polygon`, который визуально выглядит так, как это представлено на рис. П1.14. Несмотря на то, что второй многоугольник выходит за пределы первого, в качестве результата возвращаются границы первого многоугольника.

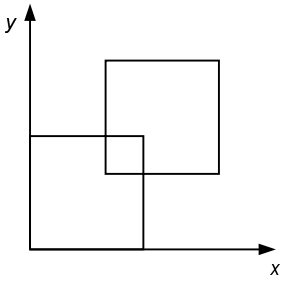


Рис. П1.14. Объект
`Polygon((0 0,0 3,3 3,3 0,0 0),(2 2,2 5,5 5,5 2,2 2))`

П1.8.3. Функция *InteriorRingN()*

Функция `InteriorRingN()` возвращает *N*-е внутреннее кольцо объекта `Polygon` и имеет следующий синтаксис:

`InteriorRingN(`*polygon*`, `*n*`)`

Функция принимает в качестве первого параметра многоугольник *polygon*, а в качестве второго — номер *n* внутреннего кольца. В результате возвращается *N*-ое внутреннее кольцо в виде объекта `LineString`. Под внутренними кольцами понимаются все кольца, кроме первого. Нумерация начинается с 1. В листинге П1.60 приводится пример использования функции.

> ***Замечание***
>
> Если в качестве аргумента функции `InteriorRingN()` передается значение, не являющееся объектом типа `Polygon`, функция возвращает `NULL`. Если номер точки *n* больше максимального количества внутренних колец или меньше нуля, функция возвращает `NULL`.

Листинг П1.60. Использование функции `InteriorRingN()`

```
mysql> SET @poly = 'Polygon((0 0,0 3,3 3,3 0,0 0),
    ->                      (1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT AsText(InteriorRingN(GeomFromText(@poly),1));
+-------------------------------------------+
```

```
| AsText(InteriorRingN(GeomFromText(@poly),1)) |
+----------------------------------------------+
| LineString(1 1,1 2,2 2,2 1,1 1)              |
+----------------------------------------------+
```

П1.8.4. Функция *NumInteriorRings()*

Функция `NumInteriorRings()` возвращает количество внутренних колец (колец, отличных от первого) в объекте `Polygon` и имеет следующий синтаксис:

`NumInteriorRings(`*`polygon`*`)`

Функция принимает в качестве единственного параметра многоугольник *`polygon`* и возвращает число внутренних колец (это число всегда на единицу меньше, чем число колец в объекте `Polygon`). В листинге П1.61 приводится пример использования функции `NumInteriorRings()`.

> ***Замечание***
>
> Если в качестве аргумента функции `NumInteriorRings()` передается значение, не являющееся объектом типа `Polygon`, функция возвращает `NULL`.

Листинг П1.61. Использование функции `NumInteriorRings()`

```
mysql> SET @poly = 'Polygon((0 0,0 3,3 3,3 0,0 0),
    ->                      (1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT NumInteriorRings(GeomFromText(@poly));
+--------------------------------------+
| NumInteriorRings(GeomFromText(@poly)) |
+--------------------------------------+
| 1                                    |
+--------------------------------------+
```

П1.9. Функции для работы с объектом *MultiPolygon*

В настоящий момент в данной группе содержится только одна функция — `Area()`.

П1.9.1. Функция *Area()*

Функция `Area()` возвращает площадь объекта `MultiPolygon` и имеет следующий синтаксис:

`Area(`*`multipolygon`*`)`

Функция принимает в качестве единственного параметра коллекцию многоугольников *multipolygon* и возвращает сумму их площадей в виде числа двойной точности. В листинге П1.62 приводится пример использования функции.

> ***Замечание***
>
> Если в качестве аргумента функции `Area()` передается значение, не являющееся объектом типа `MultiPolygon`, функция возвращает `NULL`.

Листинг П1.62. Использование функции `Area()`

```
mysql> SET @mpoly = 'MultiPolygon(((0 0,0 3,3 3,3 0,0 0),
    ->                             (1 1,1 2,2 2,2 1,1 1)))';
mysql> SELECT Area(GeomFromText(@mpoly));
+---------------------------+
| Area(GeomFromText(@mpoly)) |
+---------------------------+
| 8                         |
+---------------------------+
mysql> SET @mpoly = 'MultiPolygon(((0 0,0 3,3 3,3 0,0 0),
    ->                             (1 1,1 2,2 2,2 1,1 1)),
    ->                            ((0 0,0 3,3 3,3 0,0 0),
    ->                             (1 1,1 2,2 2,2 1,1 1)))';
mysql> SELECT Area(GeomFromText(@mpoly));
+---------------------------+
| Area(GeomFromText(@mpoly)) |
+---------------------------+
| 16                        |
+---------------------------+
```

П1.10. Функции для работы с объектом *GeometryCollection*

П1.10.1. Функция *GeometryN()*

Функция `GeometryN()` возвращает заданный объект из коллекционного объекта `GeometryCollection` и имеет следующий синтаксис:

`GeometryN(`*gc*`, `*n*`)`

Функция принимает в качестве первого параметра коллекцию *gc*, а в качестве второго — номер *n* элемента коллекции. В результате возвращается *N*-ный геометрический

элемент коллекции. Нумерация начинается с 1. В листинге П1.63 приводится пример использования функции.

Замечание

Если номер точки *n* больше максимального количества геометрических элементов коллекции или меньше нуля, функция возвращает пустую строку.

Листинг П1.63. Использование функции `GeometryN()`

```
mysql> SELECT AsText(GeometryN(GeomFromText(@gc),1));
+--------------------------------------+
| AsText(GeometryN(GeomFromText(@gc),1)) |
+--------------------------------------+
| Point(1 1)                           |
+--------------------------------------+
mysql> SELECT AsText(GeometryN(GeomFromText(@gc),4));
+--------------------------------------+
| AsText(GeometryN(GeomFromText(@gc),4)) |
+--------------------------------------+
|                                      |
+--------------------------------------+
```

П1.10.2. Функция *NumGeometries()*

Функция `NumGeometries()` возвращает количество геометрических объектов в коллекционном объекте `GeometryCollection` и имеет следующий синтаксис:

`NumGeometries(`*gc*`)`

Функция принимает в качестве единственного параметра коллекцию геометрических объектов *gc* и возвращает количество объектов в коллекции. В листинге П1.64 приводится пример использования функции.

Листинг П1.64. Использование функции `NumGeometries()`

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT NumGeometries(GeomFromText(@gc));
+---------------------------------+
| NumGeometries(GeomFromText(@gc)) |
+---------------------------------+
| 2                               |
+---------------------------------+
```

П1.11. Функции для проверки отношения минимальных ограничивающих прямоугольников

Функции данной группы проверяют отношения между минимальными ограничивающими прямоугольниками (Minimum Bounding Rectangle, MBR) двух геометрических объектов.

П1.11.1. Функция *MBRContains()*

Функция `MBRContains()` определяет, содержит ли минимальный ограничивающий прямоугольник одного геометрического объекта минимальный ограничивающий прямоугольник другого геометрического объекта. Функция имеет следующий синтаксис:

```
MBRContains(g1, g2)
```

Функция принимает в качестве аргументов два геометрических объекта *g1* и *g2*. В результате возвращается 1, если минимальный ограничивающий прямоугольник геометрического объекта *g1* содержит минимальный ограничивающий прямоугольник *g2*, и 0 в противном случае. Пример использования функции `MBRContains()` приводится в листинге П1.65.

Листинг П1.65. Использование функции `MBRContains()`

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Point(1 1)');
mysql> SELECT MBRContains(@g1,@g2), MBRContains(@g2,@g1);
+---------------------+---------------------+
| MBRContains(@g1,@g2) | MBRContains(@g2,@g1) |
+---------------------+---------------------+
| 1                   | 0                   |
+---------------------+---------------------+
```

П1.11.2. Функция *MBRDisjoint()*

Функция `MBRDisjoint()` определяет, пересекаются ли минимальные ограничивающие прямоугольники двух геометрических объектов, и имеет следующий синтаксис:

```
MBRDisjoint(g1, g2)
```

Функция принимает в качестве аргументов два геометрических объекта *g1* и *g2*. В результате возвращается 1, если минимальные ограничивающие прямоугольники

геометрических объектов не пересекаются, и 0, если пересечение имеет место. Пример использования функции `MBRDisjoint()` приводится в листинге П1.66.

Листинг П1.66. Использование функции `MBRDisjoint()`

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((2 2,2 5,5 5,5 2,2 2))');
mysql> SELECT MBRDisjoint(@g1, @g2);
+----------------------+
| MBRDisjoint(@g1, @g2) |
+----------------------+
| 0                    |
+----------------------+
mysql> SET @g3 = GeomFromText('Point(5 5)');
mysql> SELECT MBRDisjoint(@g1, @g3);
+----------------------+
| MBRDisjoint(@g1, @g3) |
+----------------------+
| 1                    |
+----------------------+
```

Как видно из листинга П1.66, в первом случае минимальные ограничивающие прямоугольники геометрических объектов пересекаются (см. рис. П1.14), а во втором — нет.

Следует отметить, что если ограничивающие прямоугольники совпадают хотя бы в одной или более точек, считается, что они пересекаются (листинг П1.67).

Листинг П1.67. Касание тоже считается пересечением

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((3 0,3 3,6 3,6 0,3 0))');
mysql> SELECT MBRDisjoint(@g1,@g2);
+---------------------+
| MBRDisjoint(@g1,@g2) |
+---------------------+
| 0                   |
+---------------------+
```

Визуально объекты `@g1` и `@g2` в листинге П1.67 представляют собой квадраты, которые пересекаются по одной из граней (рис. П1.15).

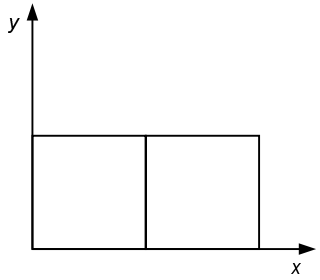


Рис. П1.15. Касание в одной или более точках тоже считается пересечением

П1.11.3. Функция *MBREqual()*

Функция `MBREqual()` определяет, совпадают ли минимальные ограничивающие прямоугольники геометрических объектов или нет. Функция имеет следующий синтаксис:

`MBREqual(`*g1*`, `*g2*`)`

Функция принимает в качестве аргументов два геометрических объекта *g1* и *g2*. В результате возвращается 1, если минимальные ограничивающие прямоугольники геометрических объектов совпадают, и 0 в противном случае. Пример использования функции `MBREqual()` приводится в листинге П1.68.

Листинг П1.68. Использование функции `MBREqual()`

```
mysql> SET @g1 = GeomFromText('Polygon((2 2,2 5,5 5,5 2,2 2))');
mysql> SET @g2 = GeomFromText('Polygon((2 2,2 5,5 5,2 2))');
mysql> SELECT MBREqual(@g1,@g2);
+------------------+
| MBREqual(@g1,@g2) |
+------------------+
| 1                |
+------------------+
mysql> SET @g1 = GeomFromText('Polygon((2 2,2 5,5 5,5 2,2 2))');
mysql> SET @g2 = GeomFromText('Polygon((6 2,6 5,9 5,6 2))');
mysql> SELECT MBREqual(@g1,@g2);
+------------------+
| MBREqual(@g1,@g2) |
+------------------+
| 0                |
+------------------+
```

В первом случае минимальный ограничивающий прямоугольник квадрата совпадает с минимальным ограничивающим прямоугольником треугольника (рис. П1.16). Поэтому функция `MBREqual()` возвращает 1.

Во втором случае минимальный ограничивающий прямоугольник квадрата не совпадает с минимальным ограничивающим прямоугольником треугольника (рис. П1.17). Поэтому функция `MBREqual()` возвращает 0.

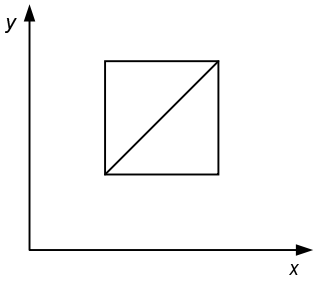


Рис. П1.16. Совпадение минимальных ограничивающих прямоугольников квадрата и треугольника

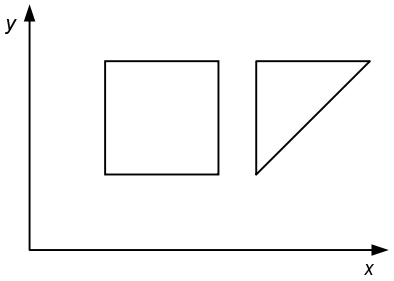


Рис. П1.17. Минимальные ограничивающие прямоугольники квадрата и треугольника не совпадают

П1.11.4. Функция *MBRIntersects()*

Функция `MBRIntersects()` определяет, пересекаются ли минимальные ограничивающие прямоугольники двух геометрических объектов. Функция имеет следующий синтаксис:

```
MBRIntersects(g1, g2)
```

Функция принимает в качестве аргументов два геометрических объекта *g1* и *g2*. В результате возвращается 1, если минимальные ограничивающие прямоугольники геометрических объектов пересекаются, и 0 в противном случае. Пример использования функции `MBRIntersects()` приводится в листинге П1.69.

Листинг П1.69. Использование функции `MBRIntersects()`

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((2 2,2 5,5 5,5 2,2 2))');
mysql> SELECT MBRIntersects(@g1, @g2);
+-------------------------+
| MBRIntersects(@g1, @g2) |
+-------------------------+
| 1                       |
+-------------------------+
```

```
mysql> SET @g3 = GeomFromText('Point(5 5)');
mysql> SELECT MBRIntersects(@g1, @g3);
+------------------------+
| MBRIntersects(@g1, @g3) |
+------------------------+
| 0                      |
+------------------------+
```

В целом, функция `MBRIntersects()` аналогична функции `MBRDisjoint()`, различается лишь трактовка: если одна функция возвращает 0, то другая функция в этой ситуации возвращает 1, и наоборот (листинг П1.70).

Листинг П1.70. Сравнение функций `MBRIntersects()` и `MBRDisjoint()`

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((3 0,3 3,6 3,6 0,3 0))');
mysql> SELECT MBRDisjoint(@g1,@g2);
+---------------------+
| MBRDisjoint(@g1,@g2) |
+---------------------+
| 0                   |
+---------------------+
mysql> SELECT MBRIntersects(@g1,@g2);
+-----------------------+
| MBRIntersects(@g1,@g2) |
+-----------------------+
| 1                     |
+-----------------------+
```

П1.11.5. Функция *MBROverlaps()*

Функция `MBROverlaps()` определяет, перекрываются ли минимальные ограничивающие прямоугольники двух геометрических объектов. Функция имеет следующий синтаксис:

`MBROverlaps(`*g1*`, `*g2*`)`

Функция принимает в качестве аргументов два геометрических объекта *g1* и *g2*. В результате возвращается 1, если минимальные ограничивающие прямоугольники геометрических объектов перекрываются, и 0 в противном случае. Пример использования функции `MBROverlaps()` приводится в листинге П1.71.

Листинг П1.71. Использование функции `MBROverlaps()`

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((2 2,2 5,5 5,5 2,2 2))');
mysql> SELECT MBROverlaps(@g1, @g2);
+----------------------+
| MBRDisjoint(@g1, @g2) |
+----------------------+
| 1                    |
+----------------------+
mysql> SET @g3 = GeomFromText('Point(5 5)');
mysql> SELECT MBROverlaps(@g1, @g3);
+----------------------+
| MBRDisjoint(@g1, @g3) |
+----------------------+
| 0                    |
+----------------------+
```

Функция `MBROverlaps()` полностью аналогична функции `MBRIntersects()`, за исключением того факта, что касание в одной или более точках не считается пересечением (листинг П1.72).

Листинг П1.72. Сравнение функций `MBROverlaps()` и `MBRIntersects()`

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((3 0,3 3,6 3,6 0,3 0))');
mysql> SELECT MBRIntersects(@g1,@g2);
+-----------------------+
| MBRIntersects(@g1,@g2) |
+-----------------------+
| 1                     |
+-----------------------+
mysql> SELECT MBROverlaps(@g1,@g2);
+----------------------+
| MBROverlaps(@g1,@g2) |
+----------------------+
| 0                    |
+----------------------+
```

П1.11.6. Функция *MBRTouches()*

Функция `MBRTouches()` определяет, соприкасаются ли минимальные ограничивающие прямоугольники двух геометрических объектов в какой-либо точке, и имеет следующий синтаксис:

`MBRTouches(`*g1*`, `*g2*`)`

Функция принимает в качестве аргументов два геометрических объекта *g1* и *g2*. В результате возвращается 1, если минимальные ограничивающие прямоугольники геометрических объектов соприкасаются в одной или более точках, и 0 в противном случае. Пример использования функции `MBRTouches()` приводится в листинге П1.73.

Листинг П1.73. Использование функции `MBRTouches()`

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((3 0,3 3,6 3,6 0,3 0))');
mysql> SELECT MBRTouches(@g1,@g2);
+--------------------+
| MBRTouches(@g1,@g2) |
+--------------------+
| 1                  |
+--------------------+
```

Причем функция возвращает 1 только в том случае, если соприкосновение происходит по касательной минимального ограничивающего прямоугольника. При пересечении прямоугольников функция вернет 0 (листинг П1.74).

Листинг П1.74. При пересечении функция `MBRTouches()` возвращает 0

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((2 2,2 5,5 5,5 2,2 2))');
mysql> SELECT MBRTouches(@g1,@g2);
+--------------------+
| MBRTouches(@g1,@g2) |
+--------------------+
| 0                  |
+--------------------+
```

П1.11.7. Функция *MBRWithin()*

Функция `MBRWithin()` позволяет определить, находится ли минимальный ограничивающий прямоугольник одного геометрического объекта внутри минимального огра-

ничивающего прямоугольника другого геометрического объекта. Функция имеет следующий синтаксис:

```
MBRWithin(g1, g2)
```

Функция принимает в качестве аргументов два геометрических объекта *g1* и *g2*. В результате возвращается 1, если минимальный ограничивающий прямоугольник геометрического объекта *g1* находится внутри минимального ограничивающего прямоугольника *g2*. В противном случае возвращается 0. Пример использования функции `MBRWithin()` приводится в листинге П1.75.

Листинг П1.75. Использование функции `MBRWithin()`

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))');
mysql> SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
+-------------------+-------------------+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
+-------------------+-------------------+
| 1                 | 0                 |
+-------------------+-------------------+
```

П1.12. Пространственные индексы

Столбцы с геометрическими объектами могут подвергаться индексации при помощи обычных индексов *(см. главу 5)*. Однако для геометрических объектов в MySQL имеется возможность создания пространственного индекса.

> ***Замечание***
>
> Построение пространственного индекса выполняется при помощи минимального ограничивающего прямоугольника объекта (MBR).

Синтаксис создания пространственного индекса аналогичен созданию обычных индексов — отличие заключается в добавлении ключевого слова `SPATIAL`. В листинге П1.76 приводятся варианты создания пространственного индекса.

> ***Замечание***
>
> Индексируемые столбцы должны быть объявлены с атрибутом `NOT NULL`.

Листинг П1.76. Варианты создания пространственного индекса

```
mysql> CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g));
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
mysql> CREATE SPATIAL INDEX sp_index ON geom (g);
```

Удалить уже созданный индекс можно либо при помощи оператора `ALTER TABLE`, либо при помощи оператора `DROP INDEX` (листинг П1.77).

> ***Замечание***
>
> Подробно синтаксис операторов `DROP INDEX` и `ALTER TABLE` рассматривается в *главах 5* и *13*, соответственно.

Листинг П1.77. Удаление пространственного индекса

```
mysql> ALTER TABLE geom DROP INDEX g;
mysql> DROP INDEX sp_index ON geom;
```

Предметный указатель

A

ACID 482
ActivePerl 839
Advanced Encryption
  Standard (AES) 352
ASCII-код 229

C

Chroot-окружение 513
CorelLab 816
Cron 616

D, F

dbExpress 816
DBI 625, 839
Foreign key (FK) 13

G, I

gcc 782
GoogleEarth 911
IP-адрес 433, 490

M

MAC-адрес 437
make 785
MBR 913, 964, 971
Message-Digest
  Algorithm (MD5) 357

O, P, R

Open GIS Consortium 911
Perl 625
Relay logs 632

S

Secure Hash Algorithm (SHA1) 359
SRID 913
SSL 505, 529, 600
Structured Query
  Language (SQL) 6, 20
    инъекция 868

T, U

Triple-DES (3DES) 355
Universal Unique
  Identifier (UUID) 436

W

Web-браузер 17, 19
Web-сервер 17
WKB 918
WKT 918

А

Атрибут:
- AUTO_INCREMENT 66, 97, 103, 105—109, 187, 189, 190, 199—201, 425, 520, 521, 542, 569, 570, 577, 603, 794, 863
- CHARACTER SET 200, 201
- COLLATE 200, 204
- COMMENT 200, 201, 204
- DEFAULT 80, 87, 107, 199, 430, 571
- DEFAULT CHARACTER SET 72
- FOREIGN KEY 200
- FULLTEXT 205
- INDEX 98, 200
- KEY 98, 200
- NO_AUTO_VALUE_ON_ZERO 542
- NOT NULL 87, 89, 92, 199, 201, 971
- NULL 87, 91, 98, 199, 201
- PRIMARY 588
- PRIMARY KEY 94, 95, 173, 181, 200, 201
- UNIQUE 98, 100, 173, 181, 200
- UNIQUE INDEX 98
- UNIQUE KEY 98
- UNSIGNED 77, 79, 542
- ZEROFILL 78

Б

База данных:
- information_schema 70, 742
- mysql 70, 358, 491, 703, 741
- test 70
- восстановление 613
- дамп 63
- денормализация 14
- иерархическая 6
- изменение 73
- имя 69
- нормализация схемы 13
- привилегии 503
- реляционная 9, 12
- сетевая 8
- система управления 6, 20
- создание 69
- текущая 423
- удаление 70
- целостность 481

Библиотека:
- dbexpmys.dll 816
- dbexpmysql.dll 816
- libmysql 816
- mysqlclient 784
- zlib 325, 528

Бинарная регистрация 570
Бинарное сравнение 229
Блокировка:
- получение 432
- проверка 434
- снятие 432, 436

В

Время бездействия 594

Г

Геометрическая модель 912
ГИС 911

Д

Дамп 63, 71, 207
Деление на ноль 243
Длина строки 335
Доменное имя 491

Ж

Журнал:
- бинарный 514, 523, 529, 531, 544, 546, 647
- запросов 515, 546
- медленных запросов 515, 531, 532, 551
- общий 531
- общих запросов 544
- ошибок 515, 531, 544, 545
- ретрансляции 533, 632, 636, 644

З

Заголовочный файл:
crypt.h 783
errmsg.h 800
math.h 783
my_global.h 783
mysql.h 783
mysqld_error.h 800
stdio.h 783
Запись:
вставка 103
максимальное число 210
в таблице 207
минимальное число 210
обновление 173
редактирование 173
средняя длина 207
удаление 167
Запрос:
вложенный 23, 443, 446, 734, 736
выборка данных 118
двухтабличный 148
коррелированный 460, 461
левое объединение 160, 734
многотабличный 147, 448
перекрестное объединение 149, 734
подчиненный 446
правое объединение 160, 734
самообъединение таблицы 153
строчный 461
трехтабличный 157
четырехтабличный 158

И

Идентификационный номер 593
Именованное условие 712
Индекс:
BTREE 101
FOREIGN KEY, внешний ключ 13, 99, 192, 200, 473, 769
FULLTEXT, многостолбцовый 96, 381, 382, 527
HASH 101
PRIMARY KEY, первичный ключ 11, 94, 95, 104, 200, 218, 473, 588, 769
SPATIAL 971
UNIQUE, уникальный 98, 200, 218, 219, 588, 769
анализ распределения 616
вторичный ключ 476
изменение 101
имя 588
логический ключ 11
максимальное число 186
обычный 98, 219
повреждение 615
полнотекстовый 94
пространственный 971
сжатие 210
создание 100, 218
суррогатный ключ 11
удаление 100, 219
Информационная схема 23, 741
Информационное сообщение 614

К

Каскадное удаление или обновление связанных записей 479
Каталог данных 41, 63, 70, 185, 194, 209, 513, 519, 525
Класс:
Curve 915
Geometry 912, 913
GeometryCollection 912, 917
LinearRing 915, 916
LineString 915
MultiCurve 912, 917
MultiLineString 912, 918
MultiPoint 912, 917
MultiPolygon 912, 918
MultiSurface 912, 918
Point 914
Polygon 916
Surface 916
Кластер 192
Кластерная модель вычислений 19

Клиент-серверная архитектура 16
Кодировка 23, 52, 63, 72, 200, 223, 229, 237, 324, 380, 512, 524, 575, 611, 743
Команда chcp 63
Команда mysql:
  \! 60
  \# 60
  \. 60
  ? 59
  \c 59
  \d 59
  \e 59
  \g 59
  \h 59
  \n 59
  \p 60
  \q 59, 60
  \r 59, 60
  \s 60
  \t 59, 60
  \u 60
  CLEAR 59
  CONNECT 59
  DELIMITER 49, 59
  EDIT 59
  EGO 59
  EXIT 59
  GO 59
  HELP 59
  NOPAGER 59
  NOTEE 59
  PAGER 60
  PRINT 60
  PROMT 60
  QUIT 60
  REHASH 60
  SOURCE 60
  STATUS 60
  SYSTEM 60
  TEE 60
  USE 60, 73
Командная строка 45
Комментарий 86, 579, 668
Компьютерная сеть 14, 194, 491
Конструкция:
  AGAINST 384, 388, 395
  ALL 456
  ANY 454, 456
  AS 399, 464, 467
  BETWEEN 251
  BINARY 245, 362, 372
  CONNECTION 554
  DISTINCT 397, 405, 406, 408, 409, 412, 449
  ELSE 417
  ENGINE 206
  EXISTS 456, 458
  FOREIGN KEY 476, 477
    CASCADE 476
    NO ACTION 476
    REFERENCES 476
    RESTRICT 476
    SET DEFAULT 476
    SET NULL 476
  FROM 461, 463, 576, 586
  GROUP BY 397, 398, 404, 449, 538, 543
    WITH ROLLUP 413
  HAVING 245, 451
  IN 254, 453, 463
  IN BOOLEAN MODE 388
  IN NATURAL LANGUAGE
    MODE 388, 395
    WITH QUERY
      EXPANSION 388
  INTERVAL 280
  IS NOT NULL 251
  IS NULL 250
  ISNULL() 250
  LIKE 265
  LIMIT 275, 424, 431, 449, 571, 584, 762
  MATCH 383
  MATCH (…) AGAINST (…) 383, 387
  NOT BETWEEN 252
  NOT EXISTS 456, 458
  NOT IN 256, 453
  ON 451
(окончание рубрики см. на стр. 979)

Конструкция *(окончание)*:
ORDER BY 202, 275, 399, 407, 413, 449, 537, 538
QUERY 554
REGEXP 265
ROW 462
SEPARATOR 406
SOME 454
SQL_CALC_FOUND_ROWS 424
THEN 417
WHEN 417
WHERE 245, 361, 383, 404, 449, 451, 471, 571
WITH QUERY EXPANSION 388, 395, 396
Корень квадратный 277
Коэффициент релевантности 385
Курсоры 23, 714

Л

Лицензии 531
GPL 22
Логический режим 388, 395
" 394
* 394
+ 389
< 391
> 391

М

Мэйнфрейм 15

Н

Нарушение ссылочной целостности 474
Натуральный логарифм 272

О

Обработчик ошибок 23, 708
CONTINUE 709
EXIT 709
NOT FOUND 709
SQLEXCEPTION 709
SQLSTATE 709
SQLWARNING 709
UNDO 709
Обратные кавычки 571
Общеупотребительные слова 384
Объект:
GeometryCollection 920, 925, 943, 962, 963
Line 945
LineString 920, 921, 931, 935, 936, 945, 951, 955
MultiLineString 920, 923, 939, 945, 955, 956
MultiPoint 920, 922, 936, 937, 945
MultiPolygon 920, 924, 940, 941, 945, 961
Point 919, 920, 927, 935, 936, 945, 951
Polygon 920, 921, 934—936, 945, 946, 957, 959—961
Объем страниц памяти 514
Округление 275
Операнд 240
Оператор 240
! 257, 265
!= 246, 456
% 244
& 259
&& 258, 685
* 109, 240, 242
/ 240, 242
:= 265, 438, 441
^ 261
| 260
|| 258, 685
~ 262
+ 240, 265, 283
< 189, 247, 685
<< 263
<= 248, 685
<=> 189, 246
<> 246, 456, 685
= 189, 245, 362, 441, 456, 685
> 121, 189, 248, 685
>= 249, 685
>> 263
(продолжение рубрики см. на стр. 980)

Оператор *(продолжение)*:
ALTER DATABASE 73, 486
ALTER FUNCTION 707
ALTER PROCEDURE 707
ALTER TABLE 179, 196, 213, 218, 220, 383, 476, 486, 515, 534, 552, 569, 582, 595, 596, 618, 926
ADD 214
ADD FULLTEXT 218
ADD INDEX 218
ADD PRIMARY KEY 218
ADD UNIQUE 218
AFTER 215, 216
COLLATE 223
COLUMN 214
DISABLE KEYS 224
DROP [COLUMN] 214
DROP INDEX 219
DROP PRIMARY KEY 219
ENABLE KEYS 224
FIRST 215, 216
MODIFY 216, 217
ORDER BY 221
RENAME 220
ALTER TABLE, CHANGE 217
ALTER VIEW 739
ANALYZE TABLE 515, 552, 588, 596, 616, 627
ANALYZE TABLES 616
AND 121, 258, 379, 387
BACKUP TABLE 622, 625
BACKUP TABLE, TO 622
BACKUP TABLES 622
BEGIN 486
BEGIN … END 664, 666, 677, 678, 684, 691, 719
BEGIN WORK 486
CACHE INDEX 552
CALL 666, 683, 733
CASE 688, 719
CHANGE MASTER 500
CHANGE MASTER TO 649
CHECK TABLE 613, 627
CHANGED 615
EXTENDED 615
FAST 615
MEDIUM 615
QUICK 615
CHECK TABLES 613
CHECKSUM TABLE 617
EXTENDED 618
QUICK 618
CLOSE 715
COMMIT 484, 486, 525, 568, 598
CREATE DATABASE 63, 65, 67, 69, 486, 579, 792
IF NOT EXISTS 71
CREATE FUNCTION 662, 679, 681
CREATE INDEX 100, 101, 383, 486, 499
CREATE PROCEDURE 662, 679, 681
IN 663, 669
INOUT 663, 671
OUT 663, 670
CREATE ROUTINE:
DETERMINISTIC 531
NO SQL 531
READS SQL DATA 531
CREATE TABLE 74, 89, 91, 98, 147, 185, 186, 188, 190—192, 196, 198, 201, 206, 381, 442, 466, 468, 470, 476, 486, 499, 533, 534, 582, 618, 832
CHARACTER SET 234
IF NOT EXISTS 198, 199
TEMPORARY 198
DEFAULT CHARACTER SET 579
DELAY_KEY_WRITE 525
CREATE TEMPORARY TABLE 442
CREATE TRIGGER 718, 719
CREATE USER 46, 493, 495, 499
IDENTIFIED BY 494
PASSWORD 494
CREATE VIEW 499, 724, 735, 739
DECLARE 676, 719
DECLARE ... CONDITION FOR 709, 712, 713
DECLARE ... HANDLER FOR 708
DECLARE CURSOR 714, 715
DELAYED 598

DELETE 54, 167, 168, 175, 193, 196, 224, 381, 428, 446, 473, 480, 499, 501, 508, 515, 532, 569, 571, 598, 619, 667, 718, 734, 792, 805, 832, 842, 860
AS 171
LIMIT 168, 169
ORDER BY 168, 169
WHERE 168
многотабличный 169
DELIMITER 665
DESCRIBE 76, 88, 95, 97, 99, 446, 577, 748, 752, 755, 761, 763, 765, 773, 927
DIV 240, 243
DROP DATABASE 65, 70, 486
DROP FUNCTION 707
DROP INDEX 100, 101, 486, 499
DROP PROCEDURE 706
IF EXISTS 706
DROP TABLE 75, 196, 211, 215, 443, 486, 499
CASCADE 212
IF EXISTS 212
RESTRICT 212
TEMPORARY 212
DROP TRIGGER 722
DROP USER 493, 495, 499, 508
DROP VIEW 738
IF EXISTS 738
ELSEIF 687
FETCH 714, 715
FLUSH 500, 553, 598
FLUSH HOSTS 532
FLUSH LOGS 544, 545, 547
FLUSH PRIVILEGES 493
FLUSH TABLES 594, 596
GET_LOCK() 595
GOTO 698
GRANT 498, 516, 542
LIKE 503
MAX_CONNECTIONS_PER_HOUR 505
MAX_QUERIES_PER_HOUR 505
MAX_UPDATES_PER_HOUR 505
MAX_USER_CONNECTIONS 505
ON 502
REQUIRE 506
REQUIRE CIPHER 506
REQUIRE ISSUER 506
REQUIRE SSL 506
REQUIRE SUBJECT 506
REQUIRE X 506
GRANT ALL, IDENTIFIED BY 503
IF 684, 686, 687, 690, 719
INSERT 23, 97, 106, 110, 145, 181, 193, 196, 224, 336, 381, 428, 446, 469, 471, 475, 482, 499, 501, 515, 524, 525, 532, 541, 569, 571, 598, 667, 718, 721, 736, 792, 805, 832, 860, 866
DELAYED 111, 526, 532
DELAYED KEY 513
IGNORE 105, 471
REPLACE 471
SELECT 524
SET 110
VALUES 103, 110, 471
многострочный 110
однострочный 103
отложенная вставка 111
ITERATE 695, 696, 719
JOIN 159, 162, 449
ON 159, 165
USING 161, 165
KILL 500, 554, 593
KILL CONNECTION 554
KILL QUERY 554
LABEL 698
LEAVE 693, 694, 696, 697, 719
LEFT JOIN 160
LEFT OUTER JOIN 161
LIKE 361
LOAD DATA 111, 114, 115, 144, 499, 524, 531, 534, 537, 550, 646
ENCLOSED BY 114
ESCAPED BY 114
FIELDS 144
FROM MASTER 631, 651, 652

(продолжение рубрики см. на стр. 982)

Оператор *(продолжение)*:
IGNORE 112
INTO TABLE 112
LINES 113, 114, 144
LOCAL 112, 531
REPLACE 112
STARTING BY 114
TERMINATED BY 114
LOAD DATA INFILE 53, 143, 144
ENCLOSED BY 145
ESCAPED BY 145
FIELDS 145
LINES 145
STARTING BY 144
TERMINATED BY 144, 145
LOAD INDEX INTO CACHE 555
LOAD MASTER DATA 486
LOAD TABLE FROM MASTER 652
LOCK TABLE 65, 487, 500, 595, 625
LOW_PRIORITY 489
READ 487, 532
WRITE 487, 532
LOCK TABLES 486, 487, 623
LOOP 697, 698, 719
MOD 244
NOT 257, 387
NOT LIKE 366
NOT REGEXP 379
NOT RLIKE 379
OPEN 714, 715
OPTIMIZE TABLE 515, 552, 596, 619, 627
OPTIMIZE TABLES 619
OR 121, 258, 379, 387, 543
PURGE MASTER LOGS 500, 647
REGEXP 367
RENAME 486
DATABASE 72
TABLE 221, 486, 596
USER 497, 499
REPAIR TABLE 534, 596, 620, 627
REPAIR TABLEEXTENDED 621
REPAIR TABLEQUICK 621
REPAIR TABLES 620
REPAIR TABLEUSE_FRM 621
REPEAT 696, 698, 719
REPLACE 181, 193, 619, 667
RESET 555
MASTER 547, 647
SLAVE 642, 653
RESTORE TABLE 623, 625
RESTORE TABLES 623
RETURN 663, 679
RETURNS 679
REVOKE 493, 498, 508
ALL PRIVILEGES 499
RIGHT JOIN 160
RIGHT OUTER JOIN 161
RLIKE 367
ROLLBACK 484, 486, 525, 568, 599
TO SAVEPOINT 486
SAVEPOINT 486
SELECT 118, 120, 135, 138, 140, 148, 149, 193, 194, 196, 222, 383, 404, 424, 438, 443, 446, 450, 500, 501, 515, 525, 532, 533, 568, 570, 571, 584, 595, 667, 703, 705, 715, 734, 786, 805, 860
ALL 135
AS 131, 152, 340, 829
BETWEEN 122
BINARY 226
COLLATE 238
DISTINCT 134, 135, 239, 595, 734
DISTINCTROW 134
FROM 118, 129, 144, 148, 159, 734
GROUP BY 135, 136, 137, 152, 153, 165, 238, 340, 734, 883
HAVING 136, 137, 165, 239, 734
IGNORE 469
IN 122
INTO 499
INTO ... FROM 672, 674, 714
INTO OUTFILE 112, 144
LIMIT 128, 129
NOT BETWEEN 122
NOT IN 123
ORDER BY 124, 126, 131, 132, 145, 153, 222, 238

ORDER BY ... DESC 127
ORDER BY ... ASC 128
REPLACE 469
SQL_NO_CACHE 536
WHERE 121, 131, 136, 137, 152, 153, 159, 239, 533, 734
SET 426, 441, 445, 479, 484, 560, 670, 719
CHARACTER SET 236, 743, 568
DEFAULT 562, 569, 570
LOCAL 562
NAMES 235, 568, 805, 842, 846
PASSWORD 492
SESSION 562
SQL_BIN_LOG 647
TRANSACTION 486
SET GLOBAL 500, 561
SQL_SLAVE_SKIP_COUNTER 653
SHOW 446, 563, 741
BDB LOGS 583
BINLOG EVENTS 648
CHARACTER SET 202, 321, 569, 570, 574, 742, 744
COLLATION 232, 233, 575, 745
COLLATION LIKE 232
COLUMNS 576, 751, 927
CREATE DATABASE 579
CREATE FUNCTION 703
CREATE PROCEDURE 702, 703
CREATE TABLE 200, 208, 542, 571, 579, 580
CREATE VIEW 500, 739
DATABASE 537, 652
DATABASES 70, 500, 503, 518, 580, 742, 763
ENGINE 582
ENGINES 581
ENGINES STORAGE 582
ERROR 533
ERRORS 583
FIELDS 577
FULL COLUMNS 200, 204, 750
FULL PROCESSLIST 500
FUNCTION STATUS 701, 702, 757
GRANTS 496, 585
INDEX 586, 765
INNODB STATUS 583
KEYS 586
MASTER LOGS 648
MASTER STATUS 648
PLUGIN 589
PRIVILEGES 502, 589
PROCEDURE STATUS 700, 701, 702, 757
PROCESSLIST 554, 593, 632, 633
SLAVE HOSTS 646, 649
SLAVE STATUS 636, 646, 653
STATUS 596, 772
TABLE STATUS 208, 601
TABLE TYPES 582
TABLES 74, 503, 604
VARIABLES 519, 606
VARIABLES, GLOBAL 606
VARIABLES, LOCAL 606
VARIABLES, SESSION 606
WARNINGS 533, 583, 607, 738
SOUNDS LIKE 367
START SLAVE 656
START TRANSACTION 485, 486
STOP SLAVE 657
TRUNCATE TABLE 168, 428, 486
UNION 138, 139, 140, 449, 734
ALL 140
DISTINCT 141
UNLOCK TABLE 65, 487, 500
UNLOCK TABLES 487
UPDATE 23, 54, 173, 176, 180, 193, 196, 241, 336, 428, 446, 480, 493, 501, 515, 532, 540, 541, 569, 571, 619, 667, 718, 734, 736, 792, 805, 832, 842, 860
AS 180
IGNORE 178
LIMIT 175, 180
ORDER BY 175, 180
SET 173
VALUES 181
WHERE 173, 241
многотабличный 178

(окончание рубрики см. на стр. 984)

Оператор *(окончание)*:
USE 73, 601, 673
WHILE 691, 698, 719
XOR 259
битовый 259
логический 121, 257
приоритет 265
Ошибка операции 614

П

Пакетная загрузка данных 111
Параметр:
--character-sets-dir 52
--compress 52
--debug 52
--default-character-set 52
--des-key-file 356
--help 52
--host 52
--no-pager 54
--password 52
--port 52
--set-variable 52
--silent 52
--socket 52
--sql-mode 320
--user 52
--version 52
-# 52
-C 52
-f 66
-h 52
-I 52
-O 52
-p 47, 52
–p 47
-P 52
-q 67
-r 68
-s 52
-S 52
-T 53
-u 46, 52
Параметр mysql:
--batch 53
--database 53
--debug-info 53
--delimiter 53
--delimiter=name 49
--execute 53
--force 53
--html 53
--i-am-a-dummy 54
--ignore-space 53
--local-infile 53
--named-commands 53, 58
--no-auto-rehash 53
--no-beep 53
--no-named-commands 54, 58
--no-tee 54
--one-database 54
--pager 54
--prompt 54, 60
--protocol 54
--quick 54
--raw 54
--reconnect 54
--safe-updates 54
--secure-auth 54
--skip-column-names 55
--skip-line-numbers 55
--table 55
--tee 54, 55
--unbuffered 55
--vertical 55, 578, 579, 744
--wait 55
--xml 55
-A 53
-b 53
-B 53
-D 53
-e 53
-E 55, 578, 744
-f 53
-g 54
-G 53
-H 53
-i 53

-L 55
-n 55
-N 55
-q 54
-r 54
-t 55
-U 54
-W 55
Параметр mysqlbinlog:
--base64-output 549
--character-sets-dir 549
--database 549
--debug 549
--disable-log-bin 549
--force-read 549
--help 549
--hexdump 550
--host 550
--local-load 550
--offset 550
--password 550
--port 550
--position 550
--protocol 550
--read-from-remote-server 550, 551
--result-file 550
--server-id 550
--short-form 550
--socket 550
--start-datetime 550
--start-position 550, 551
--stop-datetime 551
--stop-position 551
--to-last-log 551
--user 550, 551
--version 551
-# 549
-? 549
-d 549
-D 549
-f 549
-h 550
-H 550
-j 550
-l 550
-o 550
-p 550
-P 550
-r 550
-R 550
-s 550
-S 550
-t 551
-u 551
-V 551
Параметр mysqld 512
--ansi 512
--basedir 512, 522
--bdb-logdir 523
--bdb-shared-data 523
--bdb-tmpdir 523
--bind-address 512
--bin-log 642
--binlog-do-db 547, 548
--binlog-ignore-db 548
--character-set-client-handshake 512
--character-set-filesystem 512
--character-sets-dir 512
--character-set-server 513
--chroot 513
--collation-server 513
--console 513, 545
--core-file 513
--datadir 513, 525
--debug 513
--default-character-set 234
--default-collation 234
--default-storage-engine 513
--default-table-type 513
--default-time-zone 513
--delay-key-write 513
--delimiter 665
--des-key-file 513
--enable-locking 514
--enable-named-pipe 513
--event-scheduler 514
--exit-info 514
--external-locking 514
--flush 514, 527
(окончание рубрики см. на стр. 986)

Параметр mysqld *(окончание)*:
--ft_min_word_len 386
--help 512
--init-file 514, 530
--language 514
--large-pages 514
--log 514, 545
--log-bin 514, 523, 547, 631
--log-bin-index 514, 547
--log-bin-trust-function-creators 514
--log-error 515, 545
--log-isam 515
--log-output 515, 545
--log-queries-not-using-indexes 515, 551
--log-short-format 515
--log-slave-updates 642
--log-slow-admin-statements 515, 552
--log-slow-queries 515, 532, 545, 551
--log-warnings 515, 642
--low-priority-updates 515
--master-connect-retry 631, 642, 646
--master-host 643
--master-info-file 643
--master-password 643
--master-port 643
--master-retry-count 643
--master-ssl 643
--master-ssl-ca 643
--master-ssl-capath 643
--master-ssl-cert 643
--master-ssl-cipher 643
--master-user 643
--max-relay-log-size 643
--memlock 515, 531
--myisam-recover 516
--myisam-rocover 534
--old-passwords 516
--old-protocol 802
--one-thread 516
--open-files-limit 516
--pid-file 516, 535
--port 516
--port-open-timeout 516
--read-only 643
--relay-log 636
--relay-log-index 636, 644
--relay-log-info-file 644
--relay-log-purge 644
--relay-log-space-limit 644
--replicate-do-db 644
--replicate-do-table 644
--replicate-ignore-db 645
--replicate-ignore-table 645
--replicate-rewrite-db 645
--replicate-wild-do-table 645
--replicate-wild-ignore-table 645
--report-host 645, 649
--report-port 646
--safe 525
--safe-mode 516
--safe-user-create 516
--secure-auth 517, 537
--server-id 537
--shared-memory 517
--shared-memory-base-name 517
--skip-bdb 517, 523, 528
--skip-character-set-client-handshake 512
--skip-concurrent-insert 517
--skip-external-locking 517, 595
--skip-grant-tables 517
--skip-host-cache 517
--skip-innodb 517
--skip-log-warnings 515, 642
--skip-name-resolve 517
--skip-networking 517
--skip-new 525
--skip-safemalloc 517
--skip-show-database 518
--skip-slave-start 646
--skip-stack-trace 518
--skip-symbolic-links 517
--skip-thread-priority 518
--slave_compressed_protocol 646
--slave-load-tmpdir 646
--slave-net-timeout 646
--slave-skip-errors 646
--socket 518
--sql-mode 266, 279, 518, 540
--standalone 517
--symbolic-links 517

--temp-pool 518
--tmpdir 518
--transaction-isolation 518
--user 518
--version 518
--with-max-indexes 186
-? 512
-b 512
-C 513
-h 513
-L 514
-P 516
-t 518
-u 518
-V 518
Параметр mysqldump:
--add-drop-database 65
--add-drop-table 65, 67
--add-locks 65, 67
--all-databases 64, 65
--allow-keywords 65
--comments 65
--compact 65
--compatible 65
--complete-insert 66
--create-options 66, 67
--databases 64, 66
--delayed 66
--delete-master-logs 66
--disable-keys 66, 67
--extended-insert 66, 67
--fields-enclosed-by 66
--fields-escaped-by 66
--fields-optionally-enclosed-by 66
--fields-terminated-by 66
--force 66
--hex-blob 66
--ignore-table 66
--insert-ignore 67
--lines-terminated-by 66
--lock-all-tables 67
--lock-tables 67
--no-autocommit 67
--no-create-db 67
--no-data 67
--opt 67
--order-by-primary 67
--protocol 67
--quick 67
--quote-names 67
--replace 68
--result-file 68
--routines 68
--set-charset 67
--single-transaction 68
--skip-comments 68
--skip-opt 67
--skip-triggers 68
--tab 68
--tables 68
--triggers 68
--tz-utc 68
--verbose 68
--where 68
--xml 68
-A 64, 65
-B 64, 66
-c 66
-d 67
-e 66
-i 65
-K 66
-l 67
-n 67
-Q 67
-R 68
-T 68
-u 64
-v 68
-w 68
-x 67
Параметр mysqlimport:
--columns 116
--delete 116
--fields-enclosed-by 116
--fields-escaped-by 116
--fields-optionally-enclosed-by 116
--fields-terminated-by 116
--force 116
(окончание рубрики см. на стр. 988)

Параметр mysqlimport *(окончание)*:
--ignore 116
--ignore-line 116
--lines-terminated-by 116
--local 116
--lock-tables 116
--replace 117
-c 116
-D 116
-f 116
-i 116
-l 116
-L 116
-r 117
Параметр таблицы:
AUTO_INCREMENT 207, 222, 542
AVG_ROW_LENGTH 207
CHARACTER SET 207, 208
CHECKSUM 208, 618
COMMENT 208
COMPRESSED 211
CONNECTION 208
CREATE TABLE 189
DATA DIRECTORY 209
DEFAULT 211
DEFAULT CHARACTER SET 207, 223
DELAY_KEY_WRITE 209
DYNAMIC 211
ENGINE 186, 188, 189, 190, 191, 195, 206, 542
ENGINE 192, 196
FIRST 188
FIXED 211
INDEX DIRECTORY 209
INSERT_METHOD 188, 209
LAST 188
MAX_ROWS 210, 533
MIN_ROWS 210
NO 188
PACK_KEYS 210
PASSWORD 210
ROW_FORMAT 211
TYPE 186, 188, 206
UNION 211
Переменная:
SESSION_USER() 429
SYSTEM_USER() 429
пользовательская 438, 447, 560, 670, 736
сеансовая 561
серверная 560, 736
системная 519, 561
Переменная состояния:
Aborted_clients 597
Aborted_connects 597
Binlog_chache_disk_use 597
Binlog_chache_use 597
Bytes_received 597
Bytes_sent 597
Com_xxx 598
Connections 598
Created_tmp_disk_tables 598
Created_tmp_files 598
Created_tmp_tables 598
Delayed_errors 598
Delayed_insert_threads 598
Delayed_writes 598
Flush_commands 598
Handler_commit 598
Handler_delete 598
Handler_read_first 598
Handler_read_key 598
Handler_read_next 598
Handler_read_prev 598
Handler_read_rnd 598
Handler_read_rnd_next 599
Handler_rollback 599
Handler_update 599
Handler_write 599
Key_blocks_used 599
Key_read_requests 599
Key_reads 599
Key_write_requests 599
Key_writes 599
Max_used_connections 599
Not_flushed_delayed_rows 599
Not_flushed_key_blocks 599
Open_files 599
Open_streams 599

Open_tables 599
Opened_tables 599
Qcache_free_blocks 599
Qcache_free_memory 599
Qcache_inserts 599
Qcache_lowmem_prunes 599
Qcache_not_cached 599
Qcache_queries_in_cache 599
Questions 599
Rpl_status 599
Select_full_join 599
Select_full_range_join 600
Select_range 600
Select_range_check 600
Select_scan 600
Slave_open_temp_tables 600
Slave_running 600
Slow_launch_threads 600
Slow_queries 600
Sort_merge_passes 600
Sort_range 600
Sort_rows 600
Sort_scan 600
Ssl_xxx 600
Table_locks_immediate 600
Table_locks_waited 600
Threads_cached 600
Threads_connected 600
Threads_created 600
Threads_running 600
Uptime 600
Плагин 535, 589
Планировщик заданий 514, 526
Полнотекстовый поиск 186, 205, 381, 527
Пользователь:
root 46, 490
SSL-доступ 505
REQUIRE CIPHER 506
REQUIRE ISSUER 506
REQUIRE SSL 506
REQUIRE SUBJECT 506
REQUIRE X509 506
анонимный 491
изменение 497
имя 46, 196, 423, 429, 518, 594, 797, 845
максимальная длина 494
ограничение 505
MAX_CONNECTIONS_PER_HOUR 505
MAX_QUERIES_PER_HOUR 505
MAX_UPDATES_PER_HOUR 505
MAX_USER_CONNECTIONS 505
пароль 46, 196, 352, 535, 797, 845
привилегии 495, 498
редактирование 497
создание 46, 493
удаление 495, 508
Порт 34, 52, 196, 516, 535, 550, 594, 643, 646, 797
Последовательность:
#! 840
$ 69, 74
% 362, 363, 503
* 119, 131, 151
.* 171
/* 668
@ 438, 670
@@ 445
_ 69, 74, 362, 363, 503
® 395
\G 579, 744
\n 114, 144
\r\n 114, 115, 144
\t 114, 145
бинарная 696
двоичная 226
Постраничная навигация 424
Представитель 237
Представление 23, 723, 741
ALGORITHM 732
CASCADED 736
CHARACTER_SETS 742
COLLATION_CHARACTER_SET_APPLICABILITY 747
COLLATIONS 745, 747
COLUMN_PRIVILEGES 748
(окончание рубрики см. на стр. 990)

Представление *(окончание)*:
COLUMNS 751
KEY_COLUMN_USAGE 755
LOCAL 736
MERGE 732
OR REPLACE 735
ROUTINES 757
SCHEMA_PRIVILEGES 761
SCHEMATA 763
STATISTICS 765
TABLE_CONSTRAINTS 769
TABLE_PRIVILEGES 770
TABLES 772
TEMPTABLE 732
UNDEFINED 732
USER_PRIVILEGES 775
VIEWS 776
WITH CHECK OPTION 736
вертикальное 728
горизонтальное 729
редактирование 739
смешанное 732
создание 724
удаление 738
Предупреждение 614
Привилегия 500, 652
ALL 499
ALTER 499
ROUTINE 499, 531, 662
CREATE 499
ROUTINE 499, 531, 662
TEMPORARY TABLES 499
USER 499
VIEW 499, 735, 739
DELETE 499, 735, 739
DROP 499
EVENT 499
EXECUTE 499, 662
FILE 499
GRANT OPTION 500, 516
INDEX 499
INSERT 499, 516, 749
LOCK TABLES 500
PROCESS 500
REFERENCES 749
RELOAD 500, 638, 652
REPLICATION:
CLIENT 500
SLAVE 500, 638, 643
SELECT 500, 638, 735, 739, 742, 749
SHOW DATABASES 500, 518, 537
SHOW VIEW 500
SHUTDOWN 500
SUPER 500, 531, 536, 548, 570, 593, 638, 652, 719
TRIGGER 500
UPDATE 500, 749
USAGE 500
USEAGE 501
WITH GRANT OPTION 504
имя 592
область действия 592
отключение 517
уровень 502
Протокол соединения 196
версия 535

P

Расширенное имя 75
Регулярное выражение 367
$ 368
* 375
. 379
? 375
[[:<:]] 370
[[:>:]] 370
\\ 373
\\ 370
^ 368, 373
{} 376
| 370
+ 375
e-mail 378
\f 373
\n 373
\r 373
\t 373
формат ###.## 377

Режим SQL-сервера 538
ALLOW_INVALID_DATES 540
ANSI 543
ANSI_QUOTES 540
DB2 543
ERROR_FOR_DIVISION_BY_ZERO 541
HIGH_NOT_PRECEDENCE 541
IGNORE_SPACE 541
MAXDB 543
MSSQL 543
MYSQL323 543
MYSQL40 544
NO_AUTO_CREATE_USER 542
NO_AUTO_VALUE_ON_ZERO 542
NO_BACKSLASH_ESCAPES 542
NO_DIR_IN_CREATE 542
NO_ENGINE_SUBSTITUTION 542
NO_FIELD_OPTIONS 542
NO_KEY_OPTIONS 542
NO_TABLE_OPTIONS 542
NO_UNSIGNED_SUBTRACTION 542
NO_ZERO_DATE 542, 543
NO_ZERO_IN_DATE 542, 543
ONLY_FULL_GROUP_BY 543
ORACLE 544
PIPES_AS_CONCAT 543
POSTGRESQL 544
REAL_AS_FLOAT 543
STRICT_ALL_TABLES 543
STRICT_TRANS_TABLES 543
TRADITIONAL 544
Репликация 529—531, 533, 537, 630
главный сервер 531
подчиненный сервер 531

С

Сегментирование 529
Секретный ключ 352
Сервер:
MySQL 511
версия 429
встроенный 22
Сервис 34, 38
Системная переменная 532
auto_increment_increment 520
auto_increment_offset 521
autocommit 484
AUTOCOMMIT 568
back_log 522
basedir 522
bdb_cache_parts 523
bdb_cache_size 523
bdb_home 523
bdb_log_buffer_size 523
bdb_logdir 523
bdb_max_lock 523
bdb_region_size 523
bdb_shared_data 523
bdb_tmpdir 523
BIG_TABLES 444, 569
binlog_cache_size 523
bulk_insert_buffer_size 524
CHARACTER SET 569, 570
character_set_client 524
character_set_connection 236, 524
character_set_database 524
character_set_filesystem 524
character_set_results 237, 524
character_set_server 524
character_set_system 524
character_sets 524
character_sets_dir 524
collation_connection 524
collation_database 525
collation_server 525
completion_type 525
concurrent_insert 525
connect_timeout 525
datadir 525
date_format 525
datetime_format 525
default_week_format 525
delay_key_write 525
delayed_insert_limit 526
delayed_insert_timeout 526
delayed_queue_size 526
div_precision_increment 526

(окончание рубрики см. на стр. 992)

Системная переменная *(окончание)*:
event_scheduler 526
expire_logs_days 527
flush 527
flush_time 527
foreign_key_checks 479
FOREIGN_KEY_CHECKS 569
ft_boolean_syntax 527
ft_max_word_len 527
ft_min_word_len 387, 394, 527
ft_query_expansion_limit 527
ft_stopword_file 528
group_concat_max_len 528
have_archive 528
have_bdb 528
have_blackhole_engine 528
have_compress 528
have_crypt 528
have_csv 528
have_example_engine 528
have_federated_engine 528
have_geometry 528
have_innodb 529
have_ndbcluster 529
have_openssl 529
have_partitioning 529
have_query_cache 529
have_row_based_replication 529
have_rtree_keys 529
have_symlink 529
IDENTITY 426, 569
init_connect 529
init_file 530
init_slave 530
INSERT_ID 569
interactive_timeout 530
join_buffer_size 530
key_buffer_size 530
key_cache_age_threshold 530
key_cache_block_size 530
key_cache_division_limit 530
language 531
large_file_support 531
large_pages 531
LAST_INSERT_ID 426, 569
license 531
local_infile 531
locked_in_memory 531
log 531
log_bin 531
log_bin_trust_function_creators 531
log_error 531
log_slave_updates 531
log_slow_queries 531
log_warnings 531
long_query_time 515, 531
low_priority_updates 532
lower_case_file_system 532
lower_case_table_names 532
max_allowed_packet 532, 534
max_binlog_cache_size 532
max_binlog_size 532, 547
max_connect_errors 532
max_connections 516, 532
max_error_count 533
max_heap_table_size 533
max_insert_delayed_threads 533
max_join_size 533
max_relay_log_size 533
max_seeks_for_key 533
max_sort_length 533
max_tmp_tables 533
max_user_connections 533
myisam_data_pointer_size 533
myisam_max_sort_file_size 533
myisam_recover_options 534
myisam_repair_threads 534
myisam_sort_buffer_size 534
myisam_stats_method 534
myisam_use_mmap 534
named_pipe 534
NAMES 569
net_buffer_length 534
net_read_timeout 534, 652
net_retry_count 534
net_write_timeout 534, 652
new 534
old_passwords 535
ONE_SHOT 570
open_files_limit 535

optimizer_prune_level 535
optimizer_search_depth 535
pid_file 535
plugin_dir 535
port 535
preload_buffer_size 535
protocol_version 535
query_alloc_block_size 535
query_cache_limit 535
query_cache_min_res_unit 536
query_cache_size 536
query_cache_type 536
query_cache_wlock_invalidate 536
query_prealloc_size 536
range_alloc_block_size 536
read_buffer_size 536
read_only 536
read_rnd_buffer_size 537
relay_log_purge 536
secure_auth 537
server_id 537
shared_memory 537
shared_memory_base_name 537
skip_external_locking 537
skip_networking 537
skip_show_database 537
slave_compressed_protocol 537
slave_load_tmpdir 537
slave_net_timeout 538
slave_skip_errors 538
slow_launch_time 538
slow_queries 531
socket 538
sort_buffer_size 538
SQL_AUTO_IS_NULL 427, 570
SQL_BIG_SELECTS 570
SQL_BUFFER_RESULT 570
SQL_LOG_BIN 570
SQL_LOG_OFF 570
SQL_LOG_UPDATE 570
sql_mode 538
SQL_QUOTE_SHOW_CREATE 571, 580
SQL_SAFE_UPDATES 571
SQL_SELECT_LIMIT 571
sql_slave_skip_counter 538
SQL_WARNINGS 571
storage_engine 538
sync_binlog 538
sync_frm 538
system_time_zone 513, 538
table_cache 516, 538
table_definition_cache 539
table_open_cache 538, 539
table_type 538, 539
thread_cache_size 539
thread_concurrency 539
thread_stack 539
time_format 539
time_zone 539
TIMESTAMP 571
tmp_table_size 539
tmpdir 539
transaction_alloc_block_size 539
transaction_prealloc_size 539
tx_isolation 539
UNIQUE_CHECKS 571
updatable_views_with_limit 540
version 540
version_bdb 540
version_comment 540
version_compile_machine 540
version_compile_os 540
wait_timeout 540
Случайное число 274
Соединение с удаленным хостом 55
Сообщения об ошибках 514
Сортировка 575, 745
Ссылка:
внешняя 460, 461
коррелирующая 461
Стандарт времени:
ISO 297
американский 297
европейский 297
интернациональный 297
японский, индустриальный 297
Стандарт языка SQL 21
Статус операции 614

Статус соединения:
Checking table 594
Closing tables 594
Connect Out 594
Copying to tmp table on disk 594
Creating tmp table 594
Deleting from main table 594
Deleting from reference tables 594
Flushing tables 594
Killed 594
Opening tables 595
Query 594
Removing duplicates 595
Reopen table 595
Repair by sorting 595
Repair with keycache 595
Searching rows for update 595
Sending data 595
Sleep 595
Sorting for group 595
Sorting for order 595
System lock 595
Updating 595
Upgrading lock 595
User Lock 595
Waiting for handler insert 596
Waiting for tables 595
Столбец:
группировка 134
добавление 213
дополнительные сведения 577
значение по умолчанию 577
изменение 216
имя 74, 577
индексация 577
кодировка 200, 202, 208
комментарий 588
максимальное значение 131
минимальное значение 131
полное имя 150
привилегии 503
псевдоним 131, 153
сортировка 200
тип 577
удаление 214
уникальные значения 134
Сценарий:
mysql.server 43
mysql_install_db 512
mysql_intall_db 41
mysql_multi 43
mysqld_safe 43
mysqlhotcopy 625

T

Таблица:
ALTER TABLE 483
ARCHIVE 23, 111, 192, 193, 528, 625, 911
BDB 190, 191, 481, 483, 517, 523, 528, 539, 540, 583, 613, 616, 619, 911
BLACKHOLE 196, 528
CREATE TABLE 533
CSV 23, 193, 528
DUAL 130
EXAMPLE 23, 189, 190, 528
FEDERATED 23, 194, 195, 196, 528
COMMENT 195
CONNECTION 195
HEAP 189, 210, 481, 533, 604
InnoDB 22, 32, 63, 101, 190—192, 200, 475, 476, 480, 481, 483, 486, 517, 529, 539, 569, 571, 583, 604, 616, 911
ISAM 185, 481
MEMORY 101, 111, 188, 189, 533, 604
MERGE 186, 187, 188, 209, 211
MRG_MYISAM 188
MyISAM 22, 32, 63, 101, 111, 185, 186, 195, 207, 209—211, 381, 480, 481, 483, 487, 511, 517, 525, 530, 533, 534, 536, 539, 613, 616, 619, 620, 622, 623, 625, 631, 652, 911
NDB 911
NDB Cluster 23, 192, 529
блокировка 487
восстановление 516, 620
из резервной копии 623

временная 198, 212, 442, 533, 570, 736
время создания 603
дефрагментация 619
изменение структуры 213
имя 74, 603
количество записей 603
комментарий 582, 604
контрольная сумма 208, 604, 617
объединение 138
очистка 168
переименование 220
повреждение 615
привилегии 503
проверка 613
псевдоним 153
резервное копирование 622
результирующая 118, 120
самообъединение 153
связь 147, 473
создание 73
сравнение имен 532
средняя длина записи 603
структура 199
тип 582, 603
уникальные значения 134
число записей 130
Текущая дата 279
Текущее время суток 279
Тип данных:
BIGINT 77, 96, 108, 240, 243, 259, 400—402, 433
BINARY 66, 81, 226
BIT 78
BLOB 81, 189, 199, 211, 326, 336, 355, 533
BOOLEAN 78
CHAR 81, 210, 226, 326, 381
CSV 545
DATE 83, 226, 279, 540
DATETIME 83, 168, 226, 279, 540, 551
DEC 78
DECIMAL 78
DOUBLE 78, 543
ENUM 82, 94
FLOAT 78, 543
GEOMETRY 926, 931, 945, 946
GEOMETRYCOLLECTION 926, 936, 942, 943, 945
INT 77, 96, 108, 240
INTEGER 77
LINESTRING 926, 930, 931, 935, 940
MEDIUMINT 77
MULTILINESTRING 926, 936, 939
MultiPoint 926, 936, 937
MultiPolygon 926, 936, 940, 941
NULL 86, 94
NUMERIC 78
Point 926, 929, 930, 932, 938, 949, 950
Polygon 926, 933, 934, 942
PRECISION 79
REAL 79, 543
SET 82, 94, 332
SIGNED INTEGER 226
SMALLINT 77
TEXT 81, 189, 199, 211, 326, 381, 533
TIME 83, 227, 279
TIMESTAMP 23, 83, 85, 200, 245, 279, 540, 551
TINYINT 77, 108
UNSIGNED INT 240
UNSIGNED INTEGER 227
VARBINARY 66, 81
VARCHAR 81, 210, 211, 326, 381
YEAR 83, 279
временной, форматирование 288
календарный 83
приближенный 77
приведение 225
строковый 80, 104
точечный 77
числовой 77
Тонкий клиент 18
Транзакция 481, 525, 532
атомарность 482
длительность 482
изоляция 482
режим автоматического завершения 483
целостность 482

Трехуровневая архитектура 18
Триггер 23, 718, 736
  AFTER 718
  BEFORE 718
  DELETE 719
  INSERT 719
  NEW 719
  OLD 719
  UPDATE 719
  создание 718
  удаление 722

У

Удаленный хост 55, 194, 196, 797
Универсальный уникальный идентификатор 436
Уникальный идентификатор соединения 422
Утечка памяти 809
Утилита:
  myisamchk 627
  myisampack 44
  mysql 41, 43, 44, 422
  MySQL Administrator 44
  MySQL Command Line Client 36
  MySQL Control Center 44
  MySQL Query Browser 44
  MySQL Server Instance Configuration Wizard 32
  mysqladmin 43, 500, 544, 556
  mysqlbinlog 44, 549
  mysqlcheck 44, 627
  mysqldump 44, 63, 145, 191
  mysqlhotcopy 44
  mysqlimport 44, 115
  mysqlshow 44, 610
  perror 44
  replace 44
Учебная база данных 63, 75, 89
Учетная запись 490
  % 491
  root 36, 38
  блокировка 537
  ограничения 505
  пароль 492, 516
  привилегии 492, 495, 498
  редактирование 497
  создание 36, 493
  удаление 495
  удаление привилегий 508
  форма 490

Ф

Файл:
  .arm 193
  .arn 193
  .arz 193
  .csv 193
  .db 190, 191
  .frm 185, 187, 189, 190, 192—194, 210, 603, 622, 624, 740
  .mrg 187
  .myd 186, 194, 209, 622, 624
  .myi 186, 187, 209, 621, 622, 624
  db.opt 70, 73
  dbxconnections.ini 824
  dbxdrivers.ini 817, 820
  DES 356
  frm 621
  master.info 636, 642, 650
  my.cnf 234, 386, 511, 519, 520, 529, 561
  my.ini 60, 61, 73, 234, 236, 266, 320, 386, 511, 519, 520, 529, 561
  php.ini 898
  relay-log.info 636
  trg 721
  временный 518
  длина файла:
    индексного 603
    данных 603
  журнальный 514, 527, 531, 536, 544
  максимальная длина файла данных 603
  сохранение результатов во внешний файл 143
  текстовый 112, 113
Форматирование даты 288

Функция:
ABS() 266
ACOS() 267
ADDDATE() 279
ADDTIME() 285
AES_DECRYPT() 352
AES_ENCRYPT() 352
Area() 957, 961
AsBinary() 928
ASCII() 320
ASIN() 267
AsText() 928
ATAN() 267
ATAN2() 268
AVG() 397
BENCHMARK() 422
BIN() 321
BIT_AND() 400
BIT_COUNT() 264
BIT_LENGTH() 321
BIT_OR() 401, 402
CASE() 417
CAST() 226
CEIL() 268
CEILING() 268
CHAR() 321
CHAR_LENGTH() 323
CHARACTER_LENGTH() 323
CHARSET() 324
COALESCE() 253
COLLATION() 325
COMPRESS() 325, 528
CONCAT() 327, 464, 696
CONCAT_WS() 328
CONNECTION_ID() 422
CONV() 329
CONVERT() 228
CONVERT_TZ() 286
COS() 269
COT() 270
COUNT() 130, 131, 134, 135, 165, 191, 403, 861
CRC32() 270
CURDATE() 286
CURRENT_DATE 286
CURRENT_DATE() 286
CURRENT_TIME() 287
CURRENT_USER() 423, 586
CURTIME() 287
DATABASE() 423
DATE() 287
DATE_ADD() 279
DATE_FORMAT() 288, 297
DATE_SUB() 307
DATEDIFF() 288
DAY() 291
DAYNAME() 291
DAYOFMONTH() 292
DAYOFWEEK() 293
DAYOFYEAR() 293
DECODE() 355
DEFAULT() 430
DEGREES() 270
DES_DECRYPT() 355, 513
DES_ENCRYPT() 355, 513
Dimension() 945
ELT() 329
ENCODE() 355
ENCRYPT() 356, 528
EndPoint() 951
Envelope() 945
EXP() 271
EXPORT_SET() 330
ExteriorRing() 959
EXTRACT() 293
FIELD() 331
FIND_IN_SET() 331
FLOOR() 271
FORMAT() 332
FOUND_ROWS() 424
FROM_DAYS() 295
FROM_UNIXTIME() 295
GeomCollFromText() 942
GeomCollFromWKB() 943, 944
GeometryCollection() 943
GeometryCollectionFromText() 943
GeometryCollectionFromWKB() 943
GeometryFromWKB() 936
GeometryN() 962
(окончание рубрики см. на стр. 998)

Функция *(окончание)*:
GeometryStringFromText() 935
GeometryType() 948
GeomFromText() 935
GeomFromWKB() 936
GET_FORMAT() 297
GET_LOCK() 432
GLength() 953, 955
GREATEST() 254
GROUP_CONCAT() 406
GROUP_CONTACT() 528
HEX() 333
HOUR() 298
IF() 418
IFNULL() 420
INET_ATON() 433
INET_NTOA() 433
INSERT() 333
INSTR() 334
InteriorRingN() 960
INTERVAL() 256
IS_FREE_LOCK() 434
IS_USED_LOCK() 435
IsClosed() 954, 956
LAST_DAY() 299
LAST_INSERT_ID() 107, 425, 569
LCASE() 337
LEAST() 254
LEFT() 335
LENGTH() 335
LineFromText() 930
LineFromWKB() 931, 932
LineString() 931
LineStringFromText() 931
LineStringFromWKB() 931
LN() 272
LOAD_FILE() 335
LOCALTIME() 303
LOCALTIMESTAMP() 303
LOCATE() 336
LOG() 272
LOG10() 273
LOG2() 272
LOWER() 337
LPAD() 337
LTRIM() 338
MAKE_SET() 338
MAKEDATE() 300
MAKETIME() 301
MASTER_POS_WAIT() 652
MAX() 131, 132, 409
MBRContains() 964
MBRDisjoint() 964
MBREqual() 966
MBRIntersects() 967
MBROverlaps() 968
MBRTouches() 970
MBRWithin() 970
MD5() 357
MICROSECOND() 301
MID() 338
MIN() 131, 132, 408
MINUTE() 302
MLineFromWKB() 939
MOD() 244, 273
MONTH() 302
MONTHNAME() 302
MPointFromText() 936
MPointFromWKB() 937
MPolyFromText() 940
MPolyFromWKB() 941
MultiLineString() 939
MultiLineStringFromWKB() 939
MultiPointFromText() 937
MultiPointFromWKB() 937
MultiPolygon() 941, 942
MultiPolygonFromText() 941
MultiPolygonFromWKB() 941
mysql_fetch_array() 855
NAME_CONST() 435
NOW() 303
NULLIF() 421
NumGeometries() 963
NumInteriorRings() 961
NumPoints() 953
OCT() 340
OLD_PASSWORD() 359
ORD() 340
PASSWORD() 358, 493, 494
PERIOD_ADD() 304

PERIOD_DIFF() 304
PI() 273
Point() 930, 932
PointFromText() 927
PointFromWKB() 929, 930
PointN() 952
PolyFromText() 933
PolyFromWKB() 933, 934
Polygon() 934
PolygonFromText() 933
PolygonFromWKB() 934
POSITION() 341
POW() 273
POWER() 273
QUARTER() 305
QUOTE() 341
RADIANS() 274
RAND() 274
RELEASE_LOCK() 432, 436
REPEAT() 341
REPLACE() 342
REVERSE() 344
RIGHT() 344
ROUND() 275
RPAD() 345
RTRIM() 345
SEC_TO_TIME() 305
SECOND() 305
SHA() 360
SHA1() 359
SIGN() 276
SIN() 277
SLEEP() 436
SOUNDEX() 346
SQRT() 277
SRID() 949
StartPoint() 951
STD() 410
STDDEV() 410
STDDEV_POP() 410
STDDEV_SAMP() 411
STR_TO_DATE() 297, 306
STRCMP() 380
SUBDATE() 307
SUBSTRING() 339, 346, 464
SUBSTRING_INDEX() 348
SUBTIME() 308
SUM() 412
SYSDATE() 303
TAN() 277
TIME() 308
TIME_FORMAT() 312
TIME_TO_SEC() 312
TIMEDIFF() 309
TIMESTAMP() 309
TIMESTAMPADD() 310
TIMESTAMPDIFF() 311
TO_DAYS() 156, 295, 313
TRIM 348
TRUNCATE() 278
UCASE() 351
UNCOMPRESS() 349, 528
UNCOMPRESSED_LENGTH() 350
UNHEX() 351
UNIX_TIMESTAMP() 296, 314
UPPER() 351, 451
USER() 429, 541
UTC_DATE() 314
UTC_TIME() 315
UTC_TIMESTAMP() 315
UUID() 436
VAR_POP() 412
VAR_SAMP() 413
VARIANCE() 412
VERSION() 129, 130, 429
WEEK() 316, 525
WEEKDAY() 317
WEEKOFYEAR() 317
X() 949
Y() 950
YEAR() 318
YEARWEEK() 318
агрегатная 397, 734
временная 279
встроенная 129
информационная 421
использование 129
математическая 266
строковая 320
суммирующая 397
хранимая 23, 679
шифрования 352

Х

Хост 594
  локальный 490
Хранимая процедура 21, 23, 661, 736
  COMMENT 683, 707
  DETERMINISTIC 682
  LANGUAGE SQL 682
  NOT DETERMINISTIC 682
  SQL SECURITY 682, 707, 733
  групповые характеристики 681
  изменение 707
  имя 665, 701, 704
  метка 664
  параметр 663, 668
  создание 662
  тело 664
  тип 704
  удаление 706

Ц

Централизованная архитектура 15

Ч

Часовой пояс 513, 538, 539

Ш

Шифрование:
  необратимое 352, 356
  симметричное 352

Я

Язык C++:
  Builder 816
  CR_COMMANDS_OUT_OF_SYNC 803
  CR_CONN_HOST_ERROR 802
  CR_CONNECTION_ERROR 802
  CR_IPSOCK_ERROR 802
  CR_NAMEDPIPEOPEN_ERROR 802
  CR_NAMEDPIPESETSTATE_ERROR 802
  CR_NAMEDPIPEWAIT_ERROR 802
  CR_OUT_OF_MEMORY 802
  CR_SERVER_GONE_ERROR 803
  CR_SERVER_LOST 802, 803
  CR_SOCKET_CREATE_ERROR 802
  CR_UNKNOWN_ERROR 803
  CR_UNKNOWN_HOST 802
  CR_VERSION_ERROR 802
  dbExpress 816
  my_ulonglong 787
  MYSQL 786, 808
  mysql_affected_rows() 787, 792, 805
  mysql_autocommit() 792
  mysql_change_user() 792
  mysql_charset_name() 792
  mysql_close() 783, 792, 796, 799
  mysql_commit() 792
  mysql_connect() 792, 794
  mysql_create_db() 792
  mysql_data_seek() 792
  mysql_debug() 792
  mysql_drop_db() 792
  mysql_dump_debug_info() 792
  mysql_eof() 792
  mysql_errno() 792, 800
  mysql_error() 783, 792, 800
  mysql_escape_string() 793
  mysql_fetch_field() 786, 793
  mysql_fetch_field_direct() 793
  mysql_fetch_fields() 793
  mysql_fetch_lengths() 793, 812
  mysql_fetch_row() 786, 793, 806, 812
  MYSQL_FIELD 786
  mysql_field_count() 793
  MYSQL_FIELD_OFFSET 786
  mysql_field_seek() 786, 793

mysql_field_tell() 793
mysql_free_result() 793, 809
mysql_get_client_info() 793
mysql_get_client_version() 793
mysql_get_host_info() 793
mysql_get_proto_info() 793
mysql_get_server_info() 793
mysql_get_server_version() 793, 809
mysql_info() 793
mysql_init() 783, 793, 796, 797
mysql_insert_id() 787, 794
mysql_kill() 794
mysql_library_end() 794
mysql_library_init() 794
mysql_list_dbs() 794
mysql_list_fields() 794
mysql_list_processes() 794
mysql_list_tables() 794
mysql_more_results() 794
mysql_next_result() 794
mysql_num_fields() 794
mysql_num_rows() 787, 794, 810, 812
mysql_options() 794
mysql_ping() 794
mysql_query() 794, 802, 805
mysql_real_connect() 783, 792, 794, 797, 802
mysql_real_escape_string() 794
mysql_real_query() 795, 802, 805
mysql_refresh(795
mysql_reload() 795
MYSQL_RES 786, 808
mysql_rollback() 795
MYSQL_ROW 786, 806, 808
mysql_row_seek() 795
mysql_row_tell() 795
mysql_select_db() 795
mysql_server_end() 795
mysql_server_init() 795
mysql_set_server_option() 795
mysql_shutdown() 795
mysql_sqlstate() 795
mysql_stat() 795
mysql_store_result() 795, 806, 808
mysql_thread_id() 795
mysql_thread_safe() 795
mysql_use_result() 795
mysql_warning_count() 795
SQLClientDataSet 817
SQLConnection 817, 821, 823, 828
SQLDataSet 817, 827, 828, 832
SQLMonitor 817
SQLQuery 817
SQLStoredProc 817
SQLTable 817
WKBByteOrder 920
WKBGeometryCollection 925
WKBGeometryType 920
WKBLineString 921
WKBMultiLineString 924
WKBMultiPoint 923
WKBMultiPolygon 924
WKBPoint 920
WKBPolygon 922
дескриптор:
  запроса 786
  соединения 783, 786, 808
Язык Perl:
connect() 841
disconnect() 843
do() 842
execute() 842
fetchrow_array() 843
finish() 843
ppm 839
prepare() 842
Язык PHP 844
$_SESSION 871
exit() 851, 875
get_magic_quotes_gpc() 866
include_once() 848
list() 852
(окончание рубрики см. на стр. 1002)

Язык PHP *(окончание)*:
MYSQL_CLIENT_
COMPRESS 845
MYSQL_CLIENT_IGNORE_
SPACE 845
MYSQL_CLIENT_
INTERACTIVE 845
mysql_close() 846
mysql_connect() 844
mysql_erorr() 851
mysql_escape_string() 866
mysql_fetch_array() 851
mysql_fetch_assoc() 851, 853, 856
mysql_fetch_object() 851, 858
mysql_fetch_row() 851, 852, 856
mysql_num_fields() 860
mysql_num_rows() 859, 861
mysql_query() 850, 851
mysql_result() 851
mysql_select_db() 847
print_r() 856
require_once() 848
session_id() 878
session_start() 871
SQL-инъекция 868
unlink() 900
Язык структурированных
запросов (SQL) 6, 20
Язык программирования:
Perl 22
PHP 22
C/C++ 22