

---

# MySQL®

## Справочник по языку

# MySQL<sup>®</sup>

## Language Reference

MySQL AB



800 East 96th Street, Indianapolis, Indiana 46240 USA

# MySQL®

## Справочник по языку

Компания MySQL AB



Издательский дом "Вильямс"  
Москва ♦ Санкт-Петербург ♦ Киев  
2005

ББК 32.973.26-018.2.75

M11

УДК 681.3.07

Издательский дом "Вильямс"

Зав. редакцией С.Н. Тригуб

Перевод с английского Я.П. Волковой, Н.А. Мухина

Под редакцией Ю.Н. Артеменко

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:

info@williamspublishing.com, <http://www.williamspublishing.com>

115419, Москва, а/я 783; 03150, Киев, а/я 152.

### Компания MySQL AB.

M11 MySQL. Справочник по языку. : Пер. с англ. — М. : Издательский дом "Вильямс", 2005. — 432 с. — Парал. тит. англ.

ISBN 5-8459-0804-3 (рус.)

Эта книга, написанная специалистами компании MySQL AB, является всеобъемлющим справочником по языку SQL, который используется для организации запросов к базам данных, а также по особенностям реализации стандарта SQL в сервере MySQL. По сути — это официальная документация фирмы-производителя. В книге рассмотрен весь спектр вопросов, касающихся языковой структуры, допустимых типов столбцов, операторов, операций и функций, а также существующих расширений MySQL; кроме того, представлена информация, предназначенная для опытных программистов и администраторов.

Как известно, MySQL занимает лидирующие позиции среди множества систем управления базами данных с открытым исходным кодом. Благодаря высокой производительности и простоте настройки, богатому выбору API-интерфейсов, а также функциональным средствам работы с сетями, сервер MySQL стал одним из наиболее удачных вариантов для разработки Web-приложений, взаимодействующих с базами данных.

Книга рассчитана на разработчиков Web-приложений и администраторов любой квалификации, а также на студентов и преподавателей соответствующих дисциплин.

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

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

Authorized translation from the English language edition published by MySQL Press. Copyright © 2004

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Russian language edition published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2005

ISBN 5-8459-0804-3 (рус.)

ISBN 0-672-32633-7 (англ.)

© Издательский дом "Вильямс", 2005

© MySQL AB, 2004



# Оглавление

Об этой книге	14
От издательства	14
Глава 1. Общая информация	15
Глава 2. Структура языка	76
Глава 3. Поддержка наборов символов	92
Глава 4. Типы столбцов	120
Глава 5. Функции и операции	144
Глава 6. Синтаксис операторов SQL	213
Глава 7. Пространственные расширения в MySQL	331
Глава 8. Хранимые процедуры и функции	359
Глава 9. Обработка ошибок в MySQL	370
Приложение А. Поиск и устранение проблем с запросами	402
Приложение Б. Регулярные выражения MySQL	413
Предметный указатель	417

# Содержание

Об этой книге	14
От издательства	14
Глава 1. Общая информация	15
1.1. Что собой представляет это руководство	15
1.1.1. Соглашения, используемые в руководстве	16
1.2. Что такое система управления базами данных MySQL	18
1.2.1. История MySQL	19
1.2.2. Основные возможности MySQL	20
1.2.3. Стабильность MySQL	22
1.2.4. Размеры таблиц MySQL	23
1.2.5. Решение “проблемы 2000 года”	25
1.3. Компания MySQL AB	26
1.3.1. Бизнес-модель и услуги, оказываемые MySQL AB	27
1.3.1.1. Поддержка	28
1.3.1.2. Обучение и сертификация	28
1.3.1.3. Консультации	28
1.3.1.4. Коммерческие лицензии	29
1.3.1.5. Партнерство	29
1.3.2. Контактная информация	30
1.4. Поддержка и лицензирование MySQL	31
1.4.1. Поддержка, предоставляемая компанией MySQL AB	31
1.4.2. Авторские права и лицензии на MySQL	32
1.4.3. Лицензии на MySQL	32
1.4.3.1. Использование программного обеспечения MySQL по коммерческой лицензии	33
1.4.3.2. Бесплатное использование программного обеспечения MySQL по лицензии GPL	34
1.4.4. Логотипы и торговые марки MySQL AB	35
1.4.4.1. Оригинальный логотип MySQL	35
1.4.4.2. Логотипы MySQL, которые можно использовать без письменного разрешения	35
1.4.4.3. Когда необходимо иметь письменное разрешение на использование логотипов MySQL	36
1.4.4.4. Партнерские логотипы MySQL AB	36
1.4.4.5. Использование слова “MySQL” в печатном тексте и презентациях	36
1.4.4.6. Использование слова “MySQL” в названиях компаний и продуктов	36
1.5. План разработки MySQL	37
1.5.1. Кратко о MySQL 4.0	37
1.5.1.1. Возможности, доступные в MySQL 4.0	37
1.5.1.2. Встроенный сервер MySQL	39
1.5.2. Кратко о MySQL 4.1	39
1.5.2.1. Средства, доступные в MySQL 4.1	39
1.5.3. MySQL 5.0: Очередной разрабатываемый выпуск	41
1.6. MySQL и будущее (списки TODO)	41
1.6.1. Новые средства, запланированные для версии 4.1	41

1.6.2. Новые средства, запланированные для версии 5.0	42
1.6.3. Новые средства, запланированные для версии 5.1	43
1.6.4. Новые средства, запланированные на ближайшее будущее	43
1.6.5. Новые средства, запланированные на отдаленное будущее	46
1.6.6. Новые средства, которые не планируются к реализации	47
1.7. Источники информации по MySQL	48
1.7.1. Списки рассылки MySQL	48
1.7.1.1. Перечень списков рассылки MySQL	48
1.7.1.2. Как задавать вопросы и сообщать об ошибках	50
1.7.1.3. Как сообщать об ошибках и проблемах	51
1.7.1.4. Рекомендации по составлению ответов на вопросы из списков рассылки	56
1.7.2. Поддержка сообщества пользователей MySQL в IRC	56
1.8. Соответствие стандартам MySQL	56
1.8.1. Стандарты, которым соответствует MySQL	57
1.8.2. Выбор режимов SQL	57
1.8.3. Запуск MySQL в режиме ANSI	57
1.8.4. Расширения стандартного SQL в MySQL	58
1.8.5. Отличия MySQL от стандартного SQL	61
1.8.5.1. Подзапросы	61
1.8.5.2. Оператор SELECT INTO TABLE	61
1.8.5.3. Транзакции и атомарные операции	62
1.8.5.4. Хранимые процедуры и триггеры	65
1.8.5.5. Внешние ключи	65
1.8.5.6. Представления	66
1.8.5.7. '--' как начало комментария	67
1.8.6. Как MySQL работает с ограничениями	68
1.8.6.1. Ограничение PRIMARY KEY/UNIQUE	68
1.8.6.2. Ограничения NOT NULL и значения DEFAULT	68
1.8.6.3. Ограничения ENUM и SET	69
1.8.7. Известные ошибки и недостатки дизайна MySQL	70
1.8.7.1. Ошибки в версии 3.23, исправленные в более поздних версиях MySQL	70
1.8.7.2. Ошибки в версии 4.0, исправленные в более поздних версиях	70
1.8.7.3. Открытые ошибки и недостатки дизайна MySQL	70
<b>Глава 2. Структура языка</b>	<b>76</b>
2.1. Литеральные значения	76
2.1.1. Строки	76
2.1.2. Числа	78
2.1.3. Шестнадцатеричные значения	79
2.1.4. Булевские значения	79
2.1.5. Значение NULL	79
2.2. Имена баз данных, таблиц, индексов, столбцов и псевдонимов	80
2.2.1. Идентификационные квалификаторы	81
2.2.2. Чувствительность идентификаторов к регистру	82
2.3. Пользовательские переменные	84
2.4. Системные переменные	85
2.4.1. Структурированные системные переменные	86
2.5. Синтаксис комментариев	88
2.6. Трактовка зарезервированных слов MySQL	89
<b>Глава 3. Поддержка наборов символов</b>	<b>92</b>
3.1. Общие сведения о наборах символов и порядках сопоставления	92

3.2. Символьные наборы и порядки сопоставления MySQL	93
3.3. Определение символического набора и порядка сопоставления по умолчанию	95
3.3.1. Наборы символов и порядки сопоставления на уровне сервера	95
3.3.2. Наборы символов и порядки сопоставления на уровне базы данных	96
3.3.2. Наборы символов и порядки сопоставления на уровне таблицы	96
3.3.4. Наборы символов и порядки сопоставления на уровне столбца	97
3.3.5. Примеры назначения символического набора и порядка сопоставления	98
3.3.6. Наборы символов и порядки сопоставления на уровне соединения	99
3.3.7. Набор символов и порядок сопоставления строковых литералов	100
3.3.8. Применение COLLATE в операторах SQL	102
3.3.9. Приоритет конструкции COLLATE	102
3.3.10. Операция BINARY	102
3.3.11. Специальные случаи, в которых определение порядка сопоставления сложно	103
3.3.12. Порядок сопоставления должен подходить набору символов	104
3.3.13. Пример эффекта от порядка сопоставления	105
3.4. Операции, на которые влияет поддержка наборов символов	106
3.4.1. Результирующие строки	106
3.4.2. CONVERT()	106
3.4.3. CAST()	107
3.4.4. Операторы SHOW	107
3.5. Поддержка Unicode	108
3.6. UTF8 для метаданных	109
3.7. Совместимость с другими системами управления базами данных	110
3.8. Новый формат файлов определения символических наборов	111
3.9. Национальный набор символов	111
3.10. Обновление символических наборов от версии MySQL 4.0	111
3.10.1. Символьные наборы и соответствующие пары “символьный набор/порядок сопоставления” версии 4.1	112
3.10.2. Преобразование символических столбцов версии 4.0. в формат версии 4.1	113
3.11. Наборы символов и порядки сопоставления, которые поддерживает MySQL 4.1	114
3.11.1. Символьные наборы Unicode	115
3.11.2. Западноевропейские наборы символов	115
3.11.3. Центральное-европейские наборы символов	117
3.11.4. Южно-европейские и средневосточные наборы символов	117
3.11.5. Балтийские наборы символов	118
3.11.6. Кириллические наборы символов	118
3.11.7. Азиатские наборы символов	119
<b>Глава 4. Типы столбцов</b>	<b>120</b>
4.1. Обзор типов столбцов	120
4.1.1. Обзор числовых типов	120
4.1.2. Обзор типов даты и времени	123
4.1.3. Обзор строковых типов	124
4.2. Числовые типы	126
4.3. Типы даты и времени	128
4.3.1. Типы DATETIME, DATE и TIMESTAMP	130
4.3.1.1. Свойства TIMESTAMP в версиях MySQL, предшествующих 4.1	132
4.3.1.2. Свойства TIMESTAMP в MySQL версии 4.1 и выше	133
4.3.2. Тип TIME	134
4.3.3. Тип YEAR	135
4.3.4. Проблема двухтысячного года (Y2K) и типы данных	135

4.4. Строковые типы	136
4.4.1. Типы CHAR и VARCHAR	136
4.4.2. Типы BLOB и TEXT	137
4.4.3. Тип ENUM	138
4.4.4. Тип SET	140
4.5. Требования по хранению типов столбцов	141
4.6. Выбор правильного типа столбца	142
4.7. Использование типов столбцов их других систем управления базами данных	143
<b>Глава 5. Функции и операции</b>	<b>144</b>
5.1. Операции	145
5.1.1. Скобки	145
5.1.2. Операции сравнения	145
5.1.3. Логические операции	149
5.1.4. Операции, чувствительные к регистру	150
5.2. Функции управления потоком выполнения	151
5.3. Строковые функции	153
5.3.1. Функции сравнения строк	162
5.4. Числовые функции	164
5.4.1. Арифметические операции	164
5.4.2. Математические функции	165
5.5. Функции даты и времени	170
5.6. Функции полнотекстового поиска	185
5.6.1. Булевский полнотекстовый поиск	188
5.6.2. Полнотекстовый поиск с расширением запроса	189
5.6.3. Ограничения полнотекстового поиска	190
5.6.4. Тонкая настройка полнотекстового поиска MySQL	190
5.6.5. Что планируется сделать для полнотекстового поиска	192
5.7. Функции приведения	193
5.8. Другие функции	195
5.8.1. Поразрядные функции	195
5.8.2. Функции шифрования	196
5.8.3. Информационные функции	199
5.8.4. Различные функции	204
5.9. Функции и модификаторы, применяемые в конструкции GROUP BY	206
5.9.1. Агрегатные функции GROUP BY	206
5.9.2. Модификаторы GROUP BY	209
5.9.3. GROUP BY со скрытыми полями	212
<b>Глава 6. Синтаксис операторов SQL</b>	<b>213</b>
6.1. Операторы манипуляции данными	213
6.1.1. Синтаксис DELETE	213
6.1.2. Синтаксис DO	216
6.1.3. Синтаксис HANDLER	216
6.1.4. Синтаксис INSERT	217
6.1.4.1. Синтаксис INSERT...SELECT	221
6.1.4.2. Синтаксис INSERT DELAYED	221
6.1.5. Синтаксис LOAD DATA INFILE	224
6.1.6. Синтаксис REPLACE	231
6.1.7. Синтаксис SELECT	232
6.1.7.1. Синтаксис JOIN	238
6.1.7.2. Синтаксис UNION	240

6.1.8. Синтаксис подзапросов	241
6.1.8.1. Подзапрос, как скалярный операнд	242
6.1.8.2. Сравнения с использованием подзапросов	243
6.1.8.3. Подзапросы с ANY, IN и SOME	243
6.1.8.4. Подзапросы с ALL	244
6.1.8.5. Коррелированные подзапросы	244
6.1.8.6. EXISTS и NOT EXISTS	245
6.1.8.7. Подзапросы, возвращающие строку	246
6.1.8.8. Подзапросы в конструкции FROM	246
6.1.8.9. Ошибки подзапросов	247
6.1.8.10. Оптимизация подзапросов	248
6.1.8.11. Замена подзапросов соединениями для ранних версий MySQL	250
6.1.9. Синтаксис TRUNCATE	251
6.1.10. Синтаксис UPDATE	251
6.2. Операторы определения данных	253
6.2.1. Синтаксис ALTER DATABASE	253
6.2.2. Синтаксис ALTER TABLE	253
6.2.3. Синтаксис CREATE DATABASE	260
6.2.4. Синтаксис CREATE INDEX	260
6.2.5. Синтаксис CREATE TABLE	261
6.2.5.1. Создание внешних ключей	272
6.2.5.2. Неявные изменения спецификаций столбцов	274
6.2.6. Синтаксис DROP DATABASE	275
6.2.7. Синтаксис DROP INDEX	275
6.2.8. Синтаксис DROP TABLE	276
6.2.9. Синтаксис RENAME TABLE	276
6.3. Служебные операторы MySQL	277
6.3.1. Синтаксис DESCRIBE (получить информацию о столбцах)	277
6.3.2. Синтаксис USE	277
6.4. Операторы управления транзакциями и блокировкой MySQL	278
6.4.1. Синтаксис START TRANSACTION, COMMIT и ROLLBACK	278
6.4.2. Операторы, которые нельзя откатить	279
6.4.3. Операторы, вызывающие неявный COMMIT	279
6.4.4. Синтаксис SAVEPOINT и ROLLBACK TO SAVEPOINT	279
6.4.5. Синтаксис LOCK TABLES и UNLOCK TABLES	280
6.4.6. Синтаксис SET TRANSACTION	282
6.5. Операторы администрирования базы данных	283
6.5.1. Операторы управления учетными записями	283
6.5.1.1. Синтаксис DROP USER	283
6.5.1.2. Синтаксис GRANT и REVOKE	283
6.5.1.3. Синтаксис SET PASSWORD	290
6.5.2. Операторы обслуживания таблиц	291
6.5.2.1. Синтаксис ANALYZE TABLE	291
6.5.2.2. Синтаксис BACKUP TABLE	291
6.5.2.3. Синтаксис CHECK TABLE	292
6.5.2.4. Синтаксис CHECKSUM TABLE	294
6.5.2.5. Синтаксис OPTIMIZE TABLE	294
6.5.2.6. Синтаксис REPAIR TABLE	295
6.5.2.7. Синтаксис RESTORE TABLE	296
6.5.3. Синтаксис SET и SHOW	296
6.5.3.1. Синтаксис SET	297

6.5.3.2. Синтаксис SHOW CHARACTER SET	301
6.5.3.3. Синтаксис SHOW COLLATION	301
6.5.3.4. Синтаксис SHOW COLUMNS	302
6.5.3.5. Синтаксис SHOW CREATE DATABASE	302
6.5.3.6. Синтаксис SHOW CREATE TABLE	302
6.5.3.7. Синтаксис SHOW DATABASES	303
6.5.3.8. Синтаксис SHOW ENGINES	303
6.5.3.9. Синтаксис SHOW ERRORS	304
6.5.3.10. Синтаксис SHOW GRANTS	304
6.5.3.11. Синтаксис SHOW INDEX	305
6.5.3.12. Синтаксис SHOW INNODB STATUS	306
6.5.3.13. Синтаксис SHOW LOGS	306
6.5.3.14. Синтаксис SHOW PRIVILEGES	306
6.5.3.15. Синтаксис SHOW PROCESSLIST	307
6.5.3.16. Синтаксис SHOW STATUS	309
6.5.3.17. Синтаксис SHOW TABLE STATUS	310
6.5.3.18. Синтаксис SHOW TABLES	311
6.5.3.19. Синтаксис SHOW VARIABLES	312
6.5.3.20. Синтаксис SHOW WARNINGS	313
6.5.4 Другие операторы администрирования	315
6.5.4.1. Синтаксис CACHE INDEX	315
6.5.4.2. Синтаксис FLUSH	316
6.5.4.3. Синтаксис KILL	317
6.5.4.4. Синтаксис LOAD INDEX INTO CACHE	318
6.5.4.5. Синтаксис RESET	319
6.6. Операторы репликации	319
6.6.1. Операторы SQL для управления главными серверами	319
6.6.1.1. Синтаксис PURGE MASTER LOGS	319
6.6.1.2. Синтаксис RESET MASTER	320
6.6.1.3. Синтаксис SET SQL_LOG_BIN	320
6.6.1.4. Синтаксис SHOW BINLOG EVENTS	320
6.6.1.5. Синтаксис SHOW MASTER LOGS	321
6.6.1.6. Синтаксис SHOW MASTER STATUS	321
6.6.1.7. Синтаксис SHOW SLAVE HOSTS	321
6.6.2. SQL-операторы для управления подчиненными серверами	321
6.6.2.1. Синтаксис CHANGE MASTER TO	321
6.6.2.2. Синтаксис LOAD DATA FROM MASTER	324
6.6.2.3. Синтаксис LOAD TABLE имя_таблицы FROM MASTER	324
6.6.2.4. Синтаксис MASTER_POS_WAIT()	325
6.6.2.5. Синтаксис RESET SLAVE	325
6.6.2.6. Синтаксис SET GLOBAL SQL_SLAVE_SKIP_COUNTER	325
6.6.2.7. Синтаксис SHOW SLAVE STATUS	325
6.6.2.8. Синтаксис START SLAVE	329
6.6.2.9. Синтаксис STOP SLAVE	330
<b>Глава 7. Пространственные расширения в MySQL</b>	<b>331</b>
7.1. Введение	331
7.2. Геометрическая модель OpenGIS	332
7.2.1. Иерархия геометрических классов	332
7.2.2. Класс Geometry	333
7.2.3. Класс Point	335

7.2.4. Класс Curve	335
7.2.5. Класс LineString	335
7.2.6. Класс Surface	336
7.2.7. Класс Polygon	336
7.2.8. Класс GeometryCollection	336
7.2.9. Класс MultiPoint	337
7.2.10. Класс MultiCurve	337
7.2.11. Класс MultiLineString	337
7.2.12. Класс MultiSurface	338
7.2.13. Класс MultiPolygon	338
7.3. Поддерживаемые форматы пространственных данных	339
7.3.1. Формат WKT	339
7.3.2. Формат WKB	339
7.4. Создание базы данных MySQL для работы с пространственными данными	340
7.4.1. Типы пространственных данных MySQL	340
7.4.2. Создание пространственных значений	341
7.4.2.1. Создание геометрических значений с помощью WKT-функций	341
7.4.2.2. Создание геометрических значений с помощью WKB-функций	342
7.4.2.3. Создание геометрических значений с помощью специальных MySQL-функций	343
7.4.3. Создание пространственных столбцов	344
7.4.4. Заполнение пространственных столбцов	344
7.4.5. Выборка пространственных данных	345
7.4.5.1. Выборка пространственных данных во внутреннем формате	345
7.4.5.2. Выборка пространственных данных в WKT-формате	345
7.4.5.3. Выборка пространственных данных в WKB-формате	346
7.5. Анализ пространственной информации	346
7.5.1. Функции преобразования формата геометрических объектов	346
7.5.2. Геометрические функции	347
7.5.2.1. Общие функции геометрических объектов	347
7.5.2.2. Функции Point	348
7.5.2.3. Функции LineString	349
7.5.2.4. Функции MultiLineString	350
7.5.2.5. Функции Polygon	351
7.5.2.6. Функции MultiPolygon	351
7.5.2.7. Функции GeometryCollection	352
7.5.3. Функции для создания новых геометрий из существующих	352
7.5.3.1. Геометрические функции для создания новых геометрий	352
7.5.3.2. Пространственные операторы	353
7.5.4. Функции для проверки пространственных отношений между геометрическими объектами	353
7.5.5. Отношение минимальных ограничивающих прямоугольников	353
7.5.6. Функции для проверки пространственных отношений между геометриями	354
7.6. Оптимизация пространственного анализа	355
7.6.1. Создание пространственных индексов	355
7.6.2. Использование пространственного индекса	356
7.7. Соответствие и совместимость MySQL	358
7.7.1. Функции геоинформационных систем, которые пока не реализованы	358
<b>Глава 8. Хранимые процедуры и функции</b>	<b>359</b>
8.1. Синтаксис хранимой процедуры	360



8.1.1. Обслуживание хранимых процедур	360
8.1.1.1. CREATE PROCEDURE и CREATE FUNCTION	360
8.1.1.2. ALTER PROCEDURE и ALTER FUNCTION	362
8.1.1.3. DROP PROCEDURE и DROP FUNCTION	363
8.1.1.4. SHOW CREATE PROCEDURE и SHOW CREATE FUNCTION	363
8.1.2. SHOW PROCEDURE STATUS и SHOW FUNCTION STATUS	363
8.1.3. Оператор CALL	363
8.1.4. Составной оператор BEGIN ... END	363
8.1.5. Оператор DECLARE	364
8.1.6. Переменные в хранимых процедурах	364
8.1.6.1. Локальные переменные DECLARE	364
8.1.6.2. Оператор установки переменных SET	364
8.1.6.3. Оператор SELECT ...	364
8.1.7. Условия и обработчики	365
8.1.7.1. Условия DECLARE	365
8.1.7.2. Обработчики DECLARE	365
8.1.8. Курсоры	366
8.1.8.1. Объявление курсоров	367
8.1.8.2. Оператор открытия курсора OPEN	367
8.1.8.3. Оператор выборки курсора FETCH	367
8.1.8.4. Оператор закрытия курсора CLOSE	367
8.1.9. Конструкции управления потоком данных	367
8.1.9.1. Оператор IF	367
8.1.9.2. Оператор CASE	368
8.1.9.3. Оператор LOOP	368
8.1.9.4. Оператор LEAVE	368
8.1.9.5. Оператор ITERATE	368
8.1.9.6. Оператор REPEAT	369
8.1.9.7. Оператор WHILE	369
<b>Глава 9. Обработка ошибок в MySQL</b>	<b>370</b>
9.1. Возвраты по ошибке	370
9.2. Сообщения об ошибках	380
<b>Приложение А. Поиск и устранение проблем с запросами</b>	<b>402</b>
А.1. Проблемы, связанные с запросами	402
А.1.1. Чувствительность к регистру во время поиска	402
А.1.2. Проблемы при использовании столбцов DATE	403
А.1.3. Проблемы со значениями NULL	404
А.1.4. Проблемы с псевдонимами столбцов	405
А.1.5. Сбой оператора ROLLBACK при работе с нетранзакционными таблицами	405
А.1.6. Удаление строк из связанных таблиц	406
А.1.7. Решение проблем с несовпадающими строками	407
А.1.8. Проблемы при сравнении чисел с плавающей запятой	407
А.2. Проблемы, связанные с оптимизатором	410
А.3. Проблемы, связанные с определением таблиц	410
А.3.1. Проблемы с ALTER TABLE	410
А.3.2. Изменение порядка столбцов в таблице	411
А.3.3. Проблемы с TEMPORARY TABLE	412
<b>Приложение Б. Регулярные выражения MySQL</b>	<b>413</b>
<b>Предметный указатель</b>	<b>417</b>

## Об этой книге

Исходными материалами для этой книги послужили разделы онлайн-документации по MySQL (доступной в различных форматах на сайте <http://www.mysql.com>), посвященные вопросам использования языка SQL, который предназначен для выполнения запросов к базам данных в MySQL. Книга представляет собой полезный и полный справочник по языковой структуре, функциям и операциям, типам столбцов и синтаксису операторов SQL.

Первоначально это руководство было написано Майклом Монти Видениусом (Michael "Monty" Widenius) и Дэвидом Аксмарком (David Axmark), а теперь его поддержкой занимается группа разработки документации по MySQL.

Большой вклад в подготовку данной книги внесли Поль Дюбуа (Paul DuBois), Стефан Хинц (Stefan Hinz) и Аржен Ленц (Arjen Lentz) из группы разработки документации.

## От издательства

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

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

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

Наши координаты:

E-mail: [info@williamspublishing.com](mailto:info@williamspublishing.com)

WWW: <http://www.williamspublishing.com>

Информация для писем из:

России: 115419, Москва, а/я 783

Украины: 03150, Киев, а/я 152

# Общая информация

**П**рограммное обеспечение MySQL® представляет собой очень быстрый, многопоточный, многопользовательский и надежный сервер баз данных SQL (Structured Query Language – язык структурированных запросов). Сервер MySQL предназначен как для обслуживания критически важных, сильно загруженных производственных систем, так и для встраивания в программное обеспечение массового применения. MySQL – торговая марка, принадлежащая MySQL AB.

Программное обеспечение MySQL распространяется в соответствии с двойной лицензией (Dual License). Пользователь может использовать его либо как бесплатный продукт с открытым исходным кодом (Open Source/Free Software) на условиях общедоступной лицензии GNU General Public License, либо приобрести стандартную коммерческую лицензию у MySQL AB. Дополнительную информацию можно найти в разделе 1.4.

Актуальная информация о программном обеспечении MySQL доступна на Web-сайте MySQL (<http://www.mysql.com/>).

## 1.1. Что собой представляет это руководство

Настоящее руководство составлено на основании тех разделов справочного руководства по MySQL (*MySQL Reference Guide*), которые посвящены языку SQL, используемому для реализации запросов к базам данных. Здесь описывается структура языка, функции и операции, типы столбцов и синтаксис операторов SQL. Сопутствующая книга, *MySQL. Руководство администратора* (М.: Издательский дом “Вильямс”, 2005, ISBN 5-8459-0805-1), служит справочником по вопросам администрирования. В ней описывается установка программного обеспечения, конфигурирование сервера, ежедневные операции по обслуживанию, поддержке таблиц и репликации данных.

Это руководство касается MySQL 5.0.1, но также применимо и к более старым версиям MySQL (таким как 3.23 или 4.0), поскольку функциональные изменения между версиями специальным образом отмечаются.

Поскольку настоящее руководство служит справочником, оно не включает в себя общие сведения об SQL и концепциях реляционных баз данных. Оно также не обучает работе с операционной системой или интерпретатором командной строки.

Программное обеспечение баз данных MySQL постоянно развивается, и его справочное руководство обновляется настолько часто, насколько это возможно. Последняя версия руководства доступна в Internet по адресу <http://dev.mysql.com/doc/> в различных форматах, включая HTML, PDF и Windows CHM.

Если у вас есть предложения, касающиеся любых дополнений и исправлений настоящего руководства, пожалуйста, присылайте их группе, занятой созданием документации, по адресу `docs@mysql.com`.

Первоначально это руководство было написано Дэвидом Аксмарком (David Axmark) и Майклом Монти Видениусом (Michael “Monty” Widenius). Теперь его поддержкой занимается группа разработки документации по MySQL (MySQL Documentation Team), в состав которой входят Аржен Ленц (Arjen Lentz), Поль Дюбуа (Paul DuBois) и Стефан Хинц (Stefan Hinz).

Авторские права на это руководство (2004 год) принадлежат шведской компании MySQL AB (см. раздел 1.4.2).

### 1.1.1. Соглашения, используемые в руководстве

В настоящем руководстве приняты следующие типографские соглашения:

- **моноширинный**

Моноширинным шрифтом представляются имена и опции команд, SQL-операторы, имена баз данных, таблиц и столбцов, код на языках C и Perl, имена файлов и переменных окружения, а также URL-адреса. Например: “Чтобы увидеть, как работает `mysqladmin`, запустите его с опцией `--help`”.

- **моноширинный с полужирным начертанием**

С помощью моноширинного шрифта с полужирным начертанием обозначается пользовательский ввод в примерах.

- **моноширинный с курсивным начертанием**

Моноширинный шрифт с курсивным начертанием используется для обозначения ввода изменяющейся информации, когда пользователь должен подставлять то или иное значение.

- **'с'**

Моноширинный шрифт и одинарные кавычки применяются для задания последовательностей символов, например: “Для определения шаблона используйте символ `'%'`”.

- **курсив**

С помощью курсива выделяются важные слова и фраза, *например, так*.

- **полужирный**

Шрифт с полужирным начертанием используется для заголовков таблиц, а также для выделения **особо важных фраз**.

Когда необходимо показать команды, которые выполняются в среде какой-либо программы, перед командами помещается специальное приглашение. Например, `shell>` обозначает, что команда запускается в среде оболочки, а `mysql>` показывает, что оператор выполняется с помощью клиентской программы `mysql`.

```
shell> введите команду оболочки
mysql> введите оператор mysql
```

Здесь через “`shell`” обозначен используемый вами командный интерпретатор. В UNIX это обычно программа, подобная `sh` или `csh`. Эквивалентными программами в Windows являются `command.exe` или `cmd.exe`, обычно запускаемые в окне консоли.

При вводе команды или оператора, показанного в примере, приглашение вводить не нужно.

В приведенном выше примере пользовательский ввод представлен полужирным начертанием. Изменяющаяся информация, вместо которой нужно подставлять выбранное вами значение, выделяется курсивным начертанием. Имена баз данных, таблиц и их столбцов часто подставляются в аргументы операторов. В настоящем руководстве такие подстановки обязательно выделяются: *имя\_БД*, *имя\_таблицы*, *имя\_столбца*. Например, вы можете встретить в книге что-нибудь такое:

```
mysql> SELECT имя_столбца FROM имя_БД.имя_таблицы;
```

Это означает, что при вводе подобного оператора потребуется указать реальные имена базы данных, таблицы и столбца, например, следующим образом:

```
mysql> SELECT author_name FROM biblio_db.author_list;
```

Ключевые слова SQL нечувствительны к регистру, то есть их можно вводить как заглавными, так и прописными буквами. В настоящем руководстве ключевые слова представлены заглавными буквами.

В описаниях синтаксиса для обозначения необязательных слов или конструкций используются квадратные скобки ('[' и ']'). В приведенном ниже примере IF EXISTS является необязательной частью выражения:

```
DROP TABLE [IF EXISTS] имя_таблицы;
```

Когда элемент синтаксиса предусматривает несколько альтернатив, они разделяются с помощью вертикальной черты ('|'). Если из набора возможных альтернатив *может* быть выбран один элемент, последовательность вариантов помещается в квадратные скобки:

```
TRIM [{BOTH | LEADING | TRAILING} [ост_строка] FROM строка];
```

В случае если из набора возможных альтернатив *должен* быть выбран один элемент, альтернативные варианты перечисляются в фигурных скобках:

```
{DESCRIBE | DESC} имя_таблицы [имя_столбца | групповой_символ];
```

С помощью троеточия (...) обозначается пропущенная часть выражения; это позволяет сократить представление сложного синтаксиса. Например, INSERT...SELECT представляет собой сокращенную форму записи оператора INSERT и следующего за ним оператора SELECT.

Троеточие также означает, что предшествующий элемент синтаксиса может повторяться. В следующем примере можно задать множество значений *опция\_сброса*, каждое из которых, кроме первого, предваряется запятой:

```
RESET опция_сброса [, опция_сброса] ...
```

Команды установки значения переменных оболочки представлены в соответствие с синтаксисом оболочки Bourne. Например, последовательность, устанавливающая значение переменной оболочки и запускающая команду, выглядит следующим образом:

```
shell> ИМЯ_ПЕРЕМЕННОЙ=значение команда
```

Если используется csh или tcsh, команды вводятся по-другому. Приведенная выше последовательность будет выглядеть так:

```
shell> setenv ИМЯ_ПЕРЕМЕННОЙ значение  
shell> команда
```

## 1.2. Что такое система управления базами данных MySQL

Разработкой, распространением и поддержкой MySQL, наиболее популярной системы управления базами данных (СУБД) с открытым исходным кодом, занимается компания MySQL AB.

MySQL AB – коммерческая компания, основанная разработчиками MySQL, строит свой бизнес на предоставлении услуг, так или иначе связанных с СУБД MySQL. Более подробную информацию о компании MySQL AB можно найти в разделе 1.3.

Кроме того, на сайте MySQL (<http://www.mysql.com/>) представлена наиболее актуальная информация о СУБД MySQL и компании MySQL AB.

- MySQL – это система управления базами данных.

База данных представляет собой структурированный набор данных. Она может содержать различную информацию – от простого списка покупок до огромного объема данных, используемого в корпоративной сети.

- MySQL – это система управления реляционными базами данных.

Реляционная база данных хранит информацию в отдельных таблицах, а не в одном большом хранилище, благодаря чему достигается высокая производительность и гибкость. Часть “SQL” слова “MySQL” обозначает “Structured Query Language” (“Язык структурированных запросов”). SQL – наиболее общий стандартизованный язык доступа к базам данных; он соответствует стандарту ANSI/ISO SQL. Стандарт SQL впервые был принят в 1986 году и на настоящее время существует несколько его версий. В настоящем руководстве “SQL-92” ссылается на стандарт, принятый в 1992 году, “SQL:1999” – на стандарт, принятый в 1999 году, и “SQL:2003” – на текущую версию стандарта. В дальнейшем под “стандартом SQL” имеется в виду текущая версия данного стандарта.

- MySQL – это система с открытым исходным кодом.

Открытость исходного кода означает, что любой желающий имеет возможность использовать и модифицировать это программное обеспечение по своему усмотрению. Получить и развернуть программное обеспечение MySQL можно из Internet, причем совершенно бесплатно. Каждый пользователь, при желании, может изучить исходные тексты и изменить их в соответствии со своими потребностями. Программное обеспечение MySQL распространяется по лицензии GPL (GNU General Public License), которая регламентирует, что разрешено, а что нет в отношении программного обеспечения. Если по тем или иным причинам лицензия GPL не устраивает либо код MySQL требуется встраивать в коммерческие приложения, следует приобрести коммерческую лицензированную версию у компании MySQL AB (см. раздел 1.4.3).

- Сервер баз данных MySQL – очень быстрый, надежный и простой в эксплуатации сервер.

Если это как раз то, что вы ищете, стоит с ним поработать. Сервер MySQL включает в себя практичный набор средств, разработанных в тесной кооперации с сообществом пользователей. Результаты сравнительных тестов производительности MySQL и других СУБД доступны по адресу <http://dev.mysql.com/tech-resources/crash-me.php>. Изначально сервер MySQL был разработан для более быстрого управления большими базами данных, чем существующие решения в

этой области, и на протяжении ряда лет успешно эксплуатировался в средах, к которым предъявлялись весьма высокие требования. Несмотря на то что MySQL пребывает в непрекращающемся процессе разработки, на сегодняшний день он предоставляет богатый набор удобных в эксплуатации средств и функций. Присущие серверу MySQL возможности сетевого взаимодействия, производительность и безопасность делают его удачным вариантом для работы с базами данных в Internet.

- Сервер MySQL работает в клиент-серверных и встроенных системах.

СУБД MySQL является клиент-серверной системой, включающей многопоточный SQL-сервер, поддерживающий различные платформы, несколько клиентских программ и библиотек, инструменты администрирования и широкий диапазон программных интерфейсов приложений (API-интерфейсов).

Сервер MySQL существует также и в форме встраиваемое многопоточной библиотеки, которую можно связывать с разрабатываемыми приложениями, чтобы получить более компактные, быстрые и легкоуправляемые продукты.

- Доступен огромный объем программного обеспечения MySQL, написанного независимыми разработчиками.

Весьма вероятно, что предпочитаемое вами приложение и язык уже поддерживают сервер баз данных MySQL.

“MySQL” произносится как “май-эс-кю-эл” (а не “май-сиквел”), однако никто не против, если вы будете произносить название “MySQL” как вам заблагорассудится.

### 1.2.1. История MySQL

Все началось с намерения подключиться с помощью `mSQL` к нашим таблицам данных, используя наши собственные низкоуровневые (ISAM) процедуры. Не особо долгое тестирование показало, что `mSQL` не обладает достаточной производительностью и гибкостью, дабы полностью удовлетворить существующие у нас требования. В итоге был создан новый SQL-интерфейс к нашей базе данных, который обладал почти таким же API-интерфейсом, что и `mSQL`. Этот API-интерфейс был разработан так, чтобы существенно упростить перенос кода независимых разработчиков, ориентированного на взаимодействие с `mSQL`, на новую платформу, связанную с MySQL.

Происхождение наименования “MySQL” неоднозначно. Наименования нашего базового каталога и огромного количества библиотек и инструментов имели префикс “`my`” в течение более 10 лет. С другой стороны, дочку одного из наших соучредителей, Монти Видениуса (Monty Widenius), тоже звали Май (My). Какое именно из этих двух обстоятельств стало причиной появления наименования MySQL – до сих пор является тайной за семью печатями, даже для нас.

Кличка дельфина MySQL, изображенного на нашем логотипе – Сакила (Sakila) – была выбрана основателями компании MySQL AB из большого списка кличек, предложенных пользователями в процессе дискуссии “Дайте кличку дельфину”. Победила кличка, предложенная Эмброузом Твибейзом (Ambrose Twebaze), разработчиком программного обеспечения с открытым исходным кодом из Свазиленда (Африка). Как утверждал Эмброуз, кличка Сакила своими корнями уходит в Си-Свати (SiSwati) – язык, на котором говорят в Свазиленде. Сакила – это также название города в Аруше (Танзания), рядом с родной страной Эмброуза, Угандой.

## 1.2.2. Основные возможности MySQL

Ниже представлен список наиболее важных характеристик программного обеспечения баз данных MySQL. Дополнительную информацию о текущих и планируемых возможностях можно получить в разделе 1.5.

- Внутренние характеристики и переносимость.
  - Написан на языках C и C++.
  - Протестирован на широком спектре различных компиляторов.
  - Работает на множестве различных платформ.
  - Для обеспечения переносимости использует инструменты GNU – Automake, Autotools и Libtool.
  - Доступны API-интерфейсы для C, C++, Eiffel, Java, Perl, PHP, Python, Ruby и Tcl.
  - Полностью многопоточный с использованием потоков ядра. Может работать в многопроцессорных системах.
  - Обеспечивает транзакционный и нетранзакционный механизмы хранения.
  - Использует очень быстрые дисковые таблицы (MyISAM) со сжатием индексов на основе бинарных деревьев (B-деревьев).
  - Сравнительно простое добавление другого механизма хранения. Это удобно, если требуется добавить SQL-интерфейс к базе данных собственной разработки.
  - Очень быстрая система распределения памяти, основанная на потоках.
  - Очень быстрые соединения, использующие оптимизированные однопроводные мультисоединения.
  - Хранимые в памяти хеш-таблицы, которые используются в качестве временных таблиц.
  - Функции SQL реализованы с использованием высоко оптимизированной библиотеки классов и должны выполняться предельно быстро. Как правило, какого-либо распределения памяти после инициализации запроса не выполняется.
  - Код MySQL протестирован с помощью инструментов поиска утечки памяти, как коммерческих, так и с открытым исходным кодом.
  - Сервер доступен как отдельная программа для использования в клиент-серверной сетевой среде. Кроме того, он также поставляется в виде библиотеки, которая может быть встроена в отдельные автономные приложения. Такие приложения могут применяться в изолированной среде или среде, не имеющей доступа к сети.
- Типы столбцов
  - Множество типов данных для столбцов таблиц: знаковые/беззнаковые целые длиной в 1, 2, 3, 4 и 8 байт; типы FLOAT, DOUBLE, CHAR, VARCHAR, TEXT, BLOB, DATE, TIME, DATETIME, TIMESTAMP, YEAR, SET, ENUM и пространственные типы OpenGIS.
  - Записи фиксированной и переменной длины.
- Операторы и функции.
  - Полная поддержка операций и функций в конструкциях SELECT и WHERE запросов, например:



```
mysql> SELECT CONCAT(first_name, ' ', last_name)
-> FROM citizen
-> WHERE income/dependents > 10000 AND age > 30;
```

- Полная поддержка конструкций GROUP BY и ORDER BY. Поддержка групповых функций (COUNT(), COUNT(DISTINCT ...), AVG(), STD(), SUM(), MAX(), MIN() и GROUP\_CONCAT()).
  - Поддержка LEFT OUTER JOIN и RIGHT OUTER JOIN как с синтаксисом SQL, так и с синтаксисом ODBC.
  - Поддержка псевдонимов для таблиц и столбцов, как того требует стандарт SQL.
  - Операторы DELETE, INSERT, REPLACE и UPDATE возвращают количество строк, которые были изменены. Вместо этого можно задать возврат количества строк, соответствующих запросу, для чего потребуется установить соответствующий флаг при подключении к серверу.
  - Специфическая для MySQL команда SHOW может быть использована для извлечения информации о базах данных, таблицах и индексах. Команда EXPLAIN позволяет просмотреть, как оптимизатор выполняет запрос.
  - Имена функций не конфликтуют с именами таблиц и столбцов. Например, ABS – абсолютно корректное имя столбца. Единственное ограничение, которое накладывается на вызов функций, – это то, что между именем функции и следующей за ним открывающей скобкой '(' не должно быть пробелов.
  - Можно смешивать таблицы из разных баз данных в одном запросе (как в MySQL 3.22).
- **Безопасность.**
- Система, основанная на паролях и привилегиях, является исключительно гибкой и безопасной и позволяет организовать верификацию средствами хоста. Пароли защищены, поскольку весь трафик паролей во время соединения с сервером шифруется.
- **Масштабируемость и ограничения.**
- Поддерживает работу баз данных огромных объемов. Например, компания MySQL AB применяет сервер MySQL для обслуживания базы данных, содержащей 50 миллионов записей. Известна также организация, использующая сервер MySQL для обслуживания базы данных из 60 000 таблиц, которая хранит около 5 миллиардов записей.
  - Разрешается иметь до 64 индексов на таблицу (в версиях, предшествующих MySQL 4.1.2, допускалось до 32 индексов). Каждый индекс может содержать от 1 до 16 столбцов или частей столбцов. Максимальная ширина индекса составляет 1000 байт (500 байт в версиях, предшествующих MySQL 4.1.2). Для индекса может применяться префикс столбцов с типами CHAR, VARCHAR, BLOB и TEXT.
- **Сетевая связность**
- Клиенты могут подключаться к серверу MySQL, используя сокеты TCP/IP на любой платформе. В Windows-системах семейства NT (NT, 2000 или XP) клиенты могут подключаться с использованием именованных каналов. В системах на базе UNIX клиенты могут подключаться через файлы сокетов UNIX-доменов.

- Интерфейс Connector/ODBC позволяет MySQL поддерживать клиентские программы, которые используют ODBC-соединения. Например, для подключения к серверу MySQL можно использовать MS Access. Клиентское программное обеспечение может выполняться под управлением Windows или UNIX. Исходные тексты интерфейса Connector/ODBC доступны. Поддерживаются все функции ODBC 2.5, равно как и множество других.
  - Интерфейс Connector/JDBC позволяет MySQL взаимодействовать с клиентскими программами на Java, в которых используются JDBC-подключения. Клиентское программное обеспечение может выполняться под управлением Windows или UNIX. Исходные тексты интерфейса Connector/JDBC доступны.
- Локализация
- Сервер может выдавать клиентам сообщения об ошибках на разных языках.
  - Полностью поддерживаются несколько кодовых таблиц, включая latin1 (ISO-8859-1), german, big5, ujis и другие. Например, в именах таблиц и столбцов разрешается применять скандинавские символы наподобие 'å', 'ä' и 'ö'. Начиная с версии MySQL 4.1, также обеспечивается поддержка Unicode.
  - Все данные сохраняются в выбранной кодировке. Все сравнения столбцов с нормальными строками чувствительны к регистру.
  - Сортировка выполняется в соответствии с выбранной кодировкой (по умолчанию используется шведский набор). Во время запуска сервера MySQL это можно изменить. В качестве примера весьма совершенной сортировки рекомендуется обратить внимание на код сортировки для чешского языка. Сервер MySQL поддерживает множество различных кодировок, причем они могут быть указаны как во время компиляции, так и во время выполнения.
- Клиенты и инструменты.
- Сервер MySQL имеет встроенную поддержку SQL-операторов для проверки, оптимизации и восстановления таблиц. Эти операторы можно выполнять в режиме командной строки, используя клиентское приложение mysqlcheck. MySQL включает также myisamchk – очень быструю утилиту командной строки для реализации тех же операций над таблицами MyISAM.
  - Все программы MySQL можно запускать на выполнение с опцией --help или -? для получения быстрой подсказки.

### 1.2.3. Стабильность MySQL

Этот раздел отвечает на вопросы “Насколько стабилен сервер MySQL?” и “Можно ли положиться на MySQL в этом проекте?”. Мы попытаемся прояснить это, а также ответить на некоторые важные вопросы, которые касаются множества потенциальных пользователей. Информация, представленная в этом разделе, основана на данных, собранных из списков рассылки и писем пользователей, которые проявляют завидную активность при обсуждении проблем и способов их решения.

Основной программный код был разработан в начале восьмидесятых годов прошлого века. Он обеспечил стабильную базу и поддерживал формат таблиц ISAM, используемый оригинальным механизмом хранения, который остается обратно совместимым и по сей день. В ТсХ, компании-предшественнице MySQL AB, код MySQL без проблем работал в проектах, начиная со середины 1996 года. Когда программное обеспечение MySQL

впервые было представлено широкой публике, пользователи быстро нашли фрагменты, которые не были протестированы. Каждая новая реализация с тех пор имела все меньше и меньше проблем переносимости, несмотря на то, что в каждой из них также добавлялось множество новых возможностей.

Каждый выпуск сервера MySQL была работоспособным. Проблемы возникали только тогда, когда пользователи имели дело с кодом из так называемых “серых зон”. Естественно, новые пользователи не знают, что это за зоны, поэтому в настоящем разделе мы попытаемся документировать те из них, которые известны в настоящий момент. Это описание в большей степени касается версий MySQL 3.23 и MySQL 4.0. Все известные и документированные ошибки в последней версии были исправлены, за исключением тех, что перечислены в разделе ошибок, имеющих отношение к собственно проекту (см. раздел 1.8.7).

Проект MySQL построен по многоуровневому принципу с независимыми модулями. Некоторые из новейших модулей перечислены ниже с указанием того, насколько хорошо они были протестированы.

- Репликация (гамма-версия).

Большая группа серверов, использующих репликацию, показали хорошие результаты во время производственной эксплуатации. В MySQL 5.x продолжается работа над расширенными средствами репликации.

- Таблицы InnoDB (стабильная версия).

Транзакционный механизм хранения был объявлен стабильным в дереве выпусков MySQL 3.23, начиная с 3.23.49. Таблицы InnoDB используются в больших, сильно загруженных производственных системах.

- Таблицы BDB (гамма-версия).

Код Berkley DB очень стабилен, однако совершенствование интерфейса транзакционного механизма хранения BDB в сервере MySQL продолжается, поэтому еще пройдет некоторое время, прежде чем модуль обслуживания таблиц BDB можно будет объявить настолько же тщательно протестированным, как и соответствующие модули для других типов таблиц.

- Полнотекстовый поиск (гамма-версия).

Полнотекстовый поиск работает, но пока широко не используется. В версию MySQL 4.0 были внесены важные усовершенствования.

- Connector/ODBC 3.51 (стабильная версия).

Connector/ODBC 3.51 использует комплект ODBC SDK 3.51 и широко применяется в производственных системах. Некоторые его выпуски зависимы от приложений и не зависят от ODBC-драйвера или лежащего в основе сервера баз данных.

- Автоматическое восстановление таблиц MyISAM (гамма-версия).

Состояние “гамма” касается только нового кода в механизме хранения MyISAM, который при открытии таблиц проверяет, были ли они правильно закрыты и, если нет, выполняет автоматическую проверку и восстанавливает их.

## 1.2.4. Размеры таблиц MySQL

В MySQL 3.22 существовало ограничение в 4 Гбайт на размер таблиц. С применением механизма хранения MyISAM в MySQL 3.23 предельный размер таблиц вырос до 8

миллионов Тбайт ( $2^{63}$  байт). На практике это означает, что максимальные размеры таблиц теперь определяются ограничениями, накладываемыми операционной системой на размеры файлов, а не внутренними ограничениями MySQL.

Механизм хранения таблиц InnoDB поддерживает таблицы InnoDB внутри табличных пространств, которые могут состоять из нескольких файлов. Это позволяет таблице превысить максимальный размер отдельного файла. Табличное пространство может включать неразмеченные дисковые разделы (находящиеся вне файловой системы ОС), что позволяет иметь чрезвычайно большие таблицы. Максимальный размер табличного пространства составляет 64 терабайта.

В табл. 1.1 приведены некоторые примеры ограничений на размеры файлов, налагаемые операционными системами.

**Таблица 1.1.** Ограничения на размеры файлов со стороны операционных систем

Операционная система	Ограничение размера файла
Linux-Intel, 32-разрядная	2 Гбайт, при использовании файловой системы LFS значительно больше
Linux-Alpha	8 Тбайт (утверждается)
Solaris 2.5.1	2 Гбайт (4 Гбайт с обновлениями)
Solaris 2.6	4 Гбайт (может быть изменено с помощью флагов)
Solaris 2.7 Intel	4 Гбайт
Solaris 2.7 UltraSPARC	512 Гбайт
NetWare с файловой системой NSS	8 Тбайт

В Linux 2.2 можно иметь таблицы MyISAM размером свыше 2 Гбайт, если воспользоваться обновлением LFS (Large Files Support – поддержка больших файлов) для файловой системы ext2. В Linux 2.4 существуют также обновления поддержки больших файлов для файловой системы ReiserFS. Большинство современных дистрибутивов Linux базируются на ядре 2.4 и уже оснащены всеми необходимыми обновлениями LFS. Однако максимально доступный размер файлов по-прежнему зависит от ряда факторов, один из которых – это тип файловой системы, используемой для хранения таблиц MySQL.

Детальный обзор LFS для Linux можно найти на странице Андреаса Джаегера (Andreas Jaeger) “Large File Support in Linux” (“Поддержка больших файлов в Linux”) по адресу [http://www.suse.de/~aj/linux\\_lfs.html](http://www.suse.de/~aj/linux_lfs.html).

По умолчанию MySQL создает таблицы MyISAM с внутренней структурой, разрешающей максимальный размер таблицы в 4 Гбайт. Максимальный размер таблицы можно проверить с помощью команды `SHOW TABLE STATUS` или `myisamchk -dv имя_таблицы`.

Если необходимо работать с таблицами MyISAM размером свыше 4 Гбайт (и ваша операционная система поддерживает файлы упомянутого размера), на этот случай в операторе `CREATE TABLE` предусмотрены опции `AVG_ROW_LENGTH` и `MAX_ROWS`. Эти опции также можно изменить с помощью команды `ALTER TABLE` уже после создания таблицы, и таким образом увеличить ее максимально допустимый размер.

Ниже представлены другие способы преодоления ограничений на размер таблиц MyISAM, налагаемых операционной системой.

- Если большая таблица используется только для чтения, с помощью утилиты `myisampack` ее можно сжать. Как правило, эта утилита сжимает таблицу примерно

до 50% от начального размера. В результате можно иметь дело со значительно большими таблицами. Посредством утилиты `myisampack` можно также объединить несколько таблиц в одну.

- Другой способ преодоления ограничений на размер файла операционной системы для таблиц MySQL заключается в использовании опции `RAID`.
- MySQL включает в себя библиотеку `MERGE`, которая позволяет управлять набором таблиц MySQL с идентичной структурой, как одной объединенной таблицей.

### 1.2.5. Решение “проблемы 2000 года”

В сервере MySQL известная “проблема 2000 года” (Y2K) полностью решена.

- В MySQL используются функции времени Unix, которые обрабатывают даты вплоть до 2037 года как значения типа `TIMESTAMP`. Для значений типа `DATE` и `DATETIME` допустима работа с датами вплоть до 9999 года.
- Все функции времени MySQL хранятся в одном исходном файле – `sql/time.cc`; они реализованы настолько тщательно, что являются безопасными в контексте “проблемы 2000 года”.
- В MySQL 3.22 и последующих версиях столбцы типа `YEAR` могут хранить год 0 и значения лет от 1901 до 2155 в одном байте и отображать их в виде двузначного или четырехзначного числа. Все двузначные годы рассматриваются как принадлежащие диапазону от 1970 до 2069; это означает, что если в столбце `YEAR` сохраняется значение 01, MySQL трактует его как 2001 год.

Приведенный далее код служит простой демонстрацией того, что сервер MySQL не имеет проблем со значениями типов `DATE` и `DATETIME` вплоть до 9999 года, а со значениями типа `TIMESTAMP` – после 2030 года.

```
mysql> DROP TABLE IF EXISTS y2k;
Query OK, 0 rows affected (0.01 sec)
(Запрос успешно выполнен, затронуто 0 строк (0.01 с))

mysql> CREATE TABLE y2k (date DATE,
->                        date_time DATETIME,
->                        time_stamp TIMESTAMP);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO y2k VALUES
-> ('1998-12-31', '1998-12-31 23:59:59', 19981231235959),
-> ('1999-01-01', '1999-01-01 00:00:00', 19990101000000),
-> ('1999-09-09', '1999-09-09 23:59:59', 19990909235959),
-> ('2000-01-01', '2000-01-01 00:00:00', 20000101000000),
-> ('2000-02-28', '2000-02-28 00:00:00', 20000228000000),
-> ('2000-02-29', '2000-02-29 00:00:00', 20000229000000),
-> ('2000-03-01', '2000-03-01 00:00:00', 20000301000000),
-> ('2000-12-31', '2000-12-31 23:59:59', 20001231235959),
-> ('2001-01-01', '2001-01-01 00:00:00', 20010101000000),
-> ('2004-12-31', '2004-12-31 23:59:59', 20041231235959),
-> ('2005-01-01', '2005-01-01 00:00:00', 20050101000000),
-> ('2030-01-01', '2030-01-01 00:00:00', 20300101000000),
-> ('2040-01-01', '2040-01-01 00:00:00', 20400101000000),
-> ('9999-12-31', '9999-12-31 23:59:59', 99991231235959);
```

```
Query OK, 14 rows affected (0.01 sec)
Records: 14 Duplicates: 0 Warnings: 2
(Записей: 14 Дубликатов: 0 Предупреждений: 2)
```

```
mysql> SELECT * FROM y2k;
```

date	date_time	time_stamp
1998-12-31	1998-12-31 23:59:59	19981231235959
1999-01-01	1999-01-01 00:00:00	19990101000000
1999-09-09	1999-09-09 23:59:59	19990909235959
2000-01-01	2000-01-01 00:00:00	20000101000000
2000-02-28	2000-02-28 00:00:00	20000228000000
2000-02-29	2000-02-29 00:00:00	20000229000000
2000-03-01	2000-03-01 00:00:00	20000301000000
2000-12-31	2000-12-31 23:59:59	20001231235959
2001-01-01	2001-01-01 00:00:00	20010101000000
2004-12-31	2004-12-31 23:59:59	20041231235959
2005-01-01	2005-01-01 00:00:00	20050101000000
2030-01-01	2030-01-01 00:00:00	20300101000000
2040-01-01	2040-01-01 00:00:00	00000000000000
9999-12-31	9999-12-31 23:59:59	00000000000000

```
14 rows in set (0.00 sec)
```

```
(14 записей в наборе (0.00 с))
```

Последние два значения столбца TIME\_STAMP равны нулю, потому что последние два значения года (2040 и 9999) выходят за пределы, допустимые для типа данных TIMESTAMP. Этот тип данных, используемый для хранения текущего времени, поддерживает значения (со знаком) в диапазоне от 19700101000000 до 20300101000000 на 32-разрядных машинах. На 64-разрядных машинах тип TIMESTAMP допускает значения (без знака) до 2106 года.

Несмотря на то что сервер MySQL сам по себе безопасен с точки зрения “проблемы 2000 года”, неприятности могут быть вызваны приложениями, которые небезопасны в этом плане. Например, многие старые приложения хранят и манипулируют датами, используя двузначную запись года вместо четырехзначной. Эта проблема может еще более усугубиться приложениями, которые трактуют значения года 00 и 99 как индикатор “потерянных” или неопределенных значений. К сожалению, упомянутые проблемы зачастую очень трудно исправить, поскольку различные приложения разработаны разными программистами, каждый из которых может использовать собственный набор соглашений и функций обработки дат.

Таким образом, сам сервер MySQL защищен от “проблемы 2000 года”, а ответственность за корректный ввод ложится на приложения.

## 1.3. Компания MySQL AB

MySQL AB – это компания основателей и основных разработчиков MySQL. Она была основана в Швеции Дэвидом Аксмарком (David Axmark), Алланом Ларссоном (Allan Larsson) и Майклом Монти Видениусом (Michael “Monty” Widenius).

Все разработчики сервера MySQL работают на компанию. MySQL AB – виртуальная организация, объединяющая людей из нескольких десятков стран мира. Каждый день мы

интенсивно общаемся через Internet друг с другом, с нашими пользователями, сторонниками и партнерами.

Мы посвятили себя разработке программного обеспечения баз данных MySQL и продвижению его для новых пользователей. Компания MySQL AB владеет авторскими правами на исходные тексты MySQL, логотип и торговую марку MySQL, а также на данное руководство (см. раздел 1.2).

Ключевые принципы MySQL показывают наше отношение к продукту и инициативе программного обеспечения с открытым исходным кодом. Компания MySQL AB стремится, чтобы программное обеспечение сервера MySQL соответствовало следующим критериям:

- Лучший и наиболее широко используемый в мире.
- Доступный всем.
- Простой в эксплуатации.
- Постоянно совершенствующийся без влияния на производительность и защищенность.
- Приносящий удовольствие от использования и улучшения.
- Свободный от ошибок.

Далее перечислены принципы, которыми руководствуется компания MySQL AB и ее сотрудники.

- Мы поддерживаем философию открытого исходного кода, а также сообщество пользователей программного обеспечения с открытым исходным кодом (Open Source).
- Мы стремимся быть хорошими гражданами.
- Мы предпочитаем работать с партнерами, которые разделяют наши взгляды и убеждения.
- Мы отвечаем на электронную почту и обеспечиваем поддержку пользователей.
- Мы – виртуальная компания, взаимодействующая через Internet.
- Мы – противники патентования программного обеспечения.

Web-сайт MySQL (<http://www.mysql.com>) предоставляет актуальную информацию по MySQL и компании MySQL AB.

Кстати, часть “AB” в имени компании означает шведское “aktiebolag” – “акционерная компания”. То есть MySQL AB в английском варианте может выглядеть как MySQL, Inc. Фактически, MySQL, Inc. и MySQL GmbH являются представительствами MySQL AB, расположенными, соответственно, в США и Германии.

### 1.3.1. Бизнес-модель и услуги, оказываемые MySQL AB

Один из наиболее часто задаваемых нам вопросов звучит так: “Как вы можете жить с того, что предоставляете бесплатно?”. А вот как:

- Компания MySQL AB зарабатывает деньги на поддержке, услугах, коммерческих лицензиях и авторских отчислениях.
- Мы используем эти доходы для финансирования разработки и расширения бизнеса, связанного с MySQL.

Компания была прибыльной, начиная со дня ее основания. В октябре 2001 года мы получили финансирование от ведущих скандинавских инвесторов. Эти инвестиции использовались для упрочнения нашей бизнес-модели и заложили фундамент устойчивого роста.

### 1.3.1.1. Поддержка

Компанией MySQL AB владеют и управляют учредители и основные разработчики СУБД MySQL. Наши разработчики взяли на себя обязательство осуществлять поддержку клиентов и других пользователей с тем, чтобы постоянно быть в курсе их нужд и проблем. Вся поддержка выполняется только квалифицированным персоналом. На самые каверзные вопросы отвечает сам Майкл Монти Видениус – автор и идеолог сервера MySQL.

Коммерческие клиенты получают высококачественную поддержку непосредственно от MySQL AB. Компания также поддерживает множество списков рассылки, в которых любой может задать вопрос на интересующую его тему.

Более подробная информация о поддержке разного уровня содержится в разделе 1.4.

### 1.3.1.2. Обучение и сертификация

Компания MySQL AB распространяет продукт MySQL и проводит обучающие курсы по нему во всем мире. Мы предлагаем как открытые курсы, так и курсы на территории заказчика, в соответствии с нуждами вашей организации. Обучающие курсы по MySQL проводят также наши партнеры – авторизованные обучающие центры.

В наших обучающих материалах используются те же примеры баз данных, что и в документации и типовых приложениях, поставляемых вместе с продуктом. Эти материалы постоянно обновляются, чтобы соответствовать самой последней версии MySQL. Наши инструкторы постоянно взаимодействуют с группой разработчиков, что гарантирует качество обучения и постоянное совершенствование учебного материала.

Посещение наших обучающих курсов позволит быстрее достичь целей, которые ставятся при разработке MySQL-приложений. Кроме того, существуют и другие преимущества:

- Экономия времени.
- Повышение производительности приложений.
- Повышение степени защищенности приложений.
- Более полное удовлетворение потребностей заказчиков и сотрудников.
- Подготовка к сертификации по MySQL.

Если вы заинтересовались нашими обучающими курсами как потенциальный клиент или партнер, посетите соответствующий раздел на нашем сайте по адресу <http://www.mysql.com/training/> либо свяжитесь с нами по электронной почте: [training@mysql.com](mailto:training@mysql.com).

Подробная информация, касающаяся программы сертификации, доступна по адресу <http://www.mysql.com/certification/>.

### 1.3.1.3. Консультации

Компания MySQL AB и ее авторизованные партнеры предоставляют консалтинговые услуги пользователям сервера MySQL и тем, кто встраивает сервер MySQL в собственные приложения, по всему миру.



Наши консультанты окажут квалифицированную помощь в проектировании и настройке баз данных, разработке эффективных запросов, настройке платформ для достижения оптимальной производительности, решении вопросов миграции, репликации, построении устойчивых транзакционных приложений и многом другом.

Консультанты тесно сотрудничают с группой разработки, что обеспечивает высокое качество их профессиональных услуг. Период проведения консультаций может составлять от двух дней интенсивных сеансов до временных промежутков, исчисляемых несколькими неделями и месяцами. Консультации охватывают круг вопросов, связанных не только непосредственно с MySQL, но также и с языками программирования и написания сценариев, такими как Perl, PHP и так далее.

Все, кто заинтересован в услугах наших консультантов или же в партнерстве в этой области, могут посетить соответствующий раздел нашего сайта по адресу <http://www.mysql.com/consulting/> либо связаться с нами по электронной почте: [consulting@mysql.com](mailto:consulting@mysql.com).

#### 1.3.1.4. Коммерческие лицензии

СУБД MySQL распространяется на условиях общедоступной лицензии GNU (General Public License – GPL). Это означает, что в соответствии с GPL программное обеспечение MySQL может использоваться бесплатно. Если вы не согласны с требованиями GPL (такими, например, как то, что ваши приложения также должны распространяться на условиях лицензии GPL), можете приобрести коммерческую лицензию на этот же продукт у компании MySQL AB (<https://order.mysql.com/>). Поскольку права на исходный код MySQL принадлежат MySQL AB, у нас есть возможность применять практику двойного лицензирования, в соответствии с которым один и тот же продукт распространяется как на условиях GPL, так и по коммерческой лицензии. Это совершенно не означает, что компания MySQL AB отступает от принципов открытого исходного кода. Подробная информация о том, в каких случаях необходима коммерческая лицензия, содержится в разделе 1.4.3.

Компания MySQL AB также продает коммерческие лицензии на продукты Open Source GPL независимых разработчиков, которые приносят некоторую новую функциональность в сервер MySQL. Хорошим примером может служить транзакционный механизм хранения InnoDB, который предназначен для поддержки ACID, блокировок на уровне строк, восстановления после аварий, управления версиями, внешних ключей и так далее.

#### 1.3.1.5. Партнерство

Компания MySQL AB поддерживает глобальную партнерскую программу, включающую обучающие курсы, консультирование и поддержку, публикации, продажу и распространение MySQL и связанных с ним продуктов. Партнеры MySQL AB указаны на нашем Web-сайте по адресу <http://www.mysql.com/>. Там же имеется информация о правах на использование специальных версий торговых марок MySQL для идентификации партнерских продуктов и продвижения их на рынке.

Если вы заинтересованы в том, чтобы стать партнером MySQL AB, обращайтесь к нам по адресу [partner@mysql.com](mailto:partner@mysql.com).

Слово “MySQL” и логотип с дельфином MySQL являются торговыми марками компании MySQL AB (см. раздел 1.4.4). Признание и узнаваемость этих торговых марок говорят о значительном вкладе основателей MySQL в технологии СУБД с открытым исходным кодом.

Web-сайт MySQL (<http://www.mysql.com>) весьма популярен среди разработчиков и пользователей. В декабре 2003 года на нем было зарегистрировано около 16 миллионов посещений. Люди, проявляющие интерес к этому сайту, в большинстве своем представляют группу лиц, принимающих решения и выдающих рекомендации по поводу приобретения программного и аппаратного обеспечения. Двенадцать процентов визитеров уполномочены принимать такие решения, и только девять процентов не имеют отношения к этому процессу. Более 65% делали одну или более покупок за последние полгода и 70% планируют это в течение ближайших месяцев.

### 1.3.2. Контактная информация

Web-сайт MySQL (<http://www.mysql.com>) предоставляет самую последнюю информацию о MySQL и компании MySQL AB.

Если вы нуждаетесь в какой-то информации, не раскрытой в нашем разделе новостей (<http://www.mysql.com/news-and-events/>), присылайте запросы по электронной почте на адрес [press@mysql.com](mailto:press@mysql.com).

Если вы заключили договор о поддержке с компанией MySQL AB, вы получите своевременные исчерпывающие ответы на все технические вопросы относительно программного обеспечения MySQL. За подробной информацией обращайтесь в раздел 1.4.1, на наш сайт в раздел <http://www.mysql.com/support> или присылайте электронные письма по адресу [sales@mysql.com](mailto:sales@mysql.com).

Информацию, касающуюся обучающих курсов по MySQL, можно получить на сайте <http://www.mysql.com/training/> или по электронной почте, отправив запрос по адресу [training@mysql.com](mailto:training@mysql.com). Обратитесь также в раздел 1.3.1.2.

Информация о программе сертификации MySQL доступна по адресу <http://www.mysql.com/certification/>. Обратитесь также в раздел 1.3.1.2.

Если вы заинтересованы в консультациях, посетите соответствующий раздел на нашем сайте по адресу <http://www.mysql.com/consulting/> или присылайте запрос по электронной почте: [consulting@mysql.com](mailto:consulting@mysql.com). Обратитесь также в раздел 1.3.1.3.

Коммерческие лицензии можно приобрести через Internet по адресу <https://order.mysql.com/>. Там же вы найдете информацию о том, как отправить факс с заказом в компанию MySQL AB. Дополнительная информация о лицензировании доступна по адресу <http://www.mysql.com/products/pricing.html>. Если у вас возникли вопросы относительно лицензирования, заполните контактную форму на Web-сайте <http://www.mysql.com> либо пришлите электронное письмо по адресу [licensing@mysql.com](mailto:licensing@mysql.com) (по вопросам лицензирования) или по адресу [sales@mysql.com](mailto:sales@mysql.com) (по вопросам приобретения). Обратитесь также в раздел 1.4.3.

Если вы представляете компанию, заинтересованную в партнерских отношениях с MySQL AB, присылайте электронные письма по адресу [partner@mysql.com](mailto:partner@mysql.com). Обратитесь также в раздел 1.3.1.5.

Более подробную информацию о политике торговой марки MySQL можно получить по адресу <http://www.mysql.com/company/trademark.html> либо через электронную почту: [trademark@mysql.com](mailto:trademark@mysql.com). Обратитесь также в раздел 1.4.4.

Если вы заинтересованы поработать на компанию MySQL AB, обращайтесь в соответствующий раздел на сайте (<http://www.mysql.com/company/jobs/>) или пишите по адресу [jobs@mysql.com](mailto:jobs@mysql.com). Не отправляйте свое резюме как вложение, а размещайте его в виде простого текста в конце электронного сообщения.

Для подключения к дискуссиям с другими пользователями просмотрите перечень наших списков рассылки. Обратитесь также в раздел 1.7.1.

Сообщения об ошибках, равно как и вопросы и комментарии, должны отправляться в общий список рассылки MySQL (см. раздел 1.7.1). Если вы обнаружили существенные ошибки в системе безопасности сервера MySQL, незамедлительно уведомьте об этом по адресу [security@mysql.com](mailto:security@mysql.com) (см. раздел 1.7.1.3).

Если вы располагаете результатами тестирования, которые стоит опубликовать, присылайте их по адресу [benchmark@mysql.com](mailto:benchmark@mysql.com).

Если у вас есть предложения по дополнениям и исправлениям настоящего руководства, направляйте их группе документации ([docs@mysql.com](mailto:docs@mysql.com)).

Вопросы и комментарии относительно Web-сайта MySQL (<http://www.mysql.com>) присылайте по адресу [webmaster@mysql.com](mailto:webmaster@mysql.com).

Компания MySQL AB имеет собственную политику конфиденциальности, с которой можно ознакомиться по адресу <http://www.mysql.com/company/privacy.html>. Вопросы по этой теме можно задавать и по электронной почте: [privacy@mysql.com](mailto:privacy@mysql.com).

По всем остальным вопросам обращайтесь по адресу [info@mysql.com](mailto:info@mysql.com).

## 1.4. Поддержка и лицензирование MySQL

В этом разделе описаны соглашения по поддержке и лицензированию MySQL.

### 1.4.1. Поддержка, предоставляемая компанией MySQL AB

Техническая поддержка компанией MySQL AB означает предоставление индивидуальных ответов по поводу решения ваших уникальных проблем непосредственно от специалистов, которые занимаются разработкой СУБД MySQL.

Компания MySQL AB старается обеспечить широкий и всеобъемлющий подход к оказанию технической поддержки. Почти любая проблема, касающаяся MySQL, важна для нас, если она важна для вас. Обычно нашим заказчикам необходима помощь в том, чтобы заставить работать различные команды и утилиты, устранить узкие места в производительности, восстановить систему после аварии, разобраться с влиянием операционной системы или сетевой среды на MySQL, настроить процедуры резервирования и восстановления данных, применять API-интерфейсы и так далее. Наша поддержка касается только сервера MySQL и наших собственных утилит, но не продуктов независимых разработчиков, которые взаимодействуют с сервером MySQL, хотя по возможности мы и стараемся оказать помощь и в отношении их.

Детальная информация о различных типах поддержки доступна по адресу <http://www.mysql.com/support/>. Там же можно затребовать и поддержку в онлайн-овом режиме. Чтобы связаться с персоналом, который занимается продажами, обращайтесь по адресу [sales@mysql.com](mailto:sales@mysql.com).

Техническая поддержка во многом похожа на страхование жизни. Вы можете счастливо жить без нее многие годы. Однако когда ваш час наступит, она становится критически важной, но тогда уже слишком поздно ее приобретать. Если вы эксплуатируете сервер MySQL для обслуживания важных приложений и сталкиваетесь с внезапными сложностями, самостоятельный поиск ответов может оказаться слишком накладным в смысле времени. Вам понадобится немедленно связаться с самыми опытными специалистами по устранению проблем, которые работают на компанию MySQL AB.

## 1.4.2. Авторские права и лицензии на MySQL

Компания MySQL AB владеет авторскими правами на исходный текст MySQL, логотипы, торговые марки и настоящее руководство (см. раздел 1.3). Важно знать несколько лицензионных соглашений относительно распространения MySQL:

1. Исходный код сервера MySQL, библиотеки `mysqlclient`, клиентских утилит, а также GNU-библиотеки `readline`, подпадает под действие лицензии GNU General Public License (<http://fsf.org/licenses/>). Файл `COPYING` в дистрибутиве MySQL содержит текст этой лицензии.
2. Использование библиотеки GNU `getopt` регламентируется малой общедоступной лицензией GNU Lesser General Public License (<http://www.fsf.org/licenses/>).
3. Некоторые части исходных текстов (в частности, библиотека `regex`) защищены авторскими правами стиля Berkley.
4. Версии MySQL, предшествующие MySQL 3.22, регламентируются более строгими лицензиями (<http://www.mysql.com/products/mypl.html>). Детальную информацию можно найти в документации по этим версиям.
5. Распространение справочного руководства по MySQL не подпадает под действие лицензии GPL. Использование этого руководства ограничено следующими условиями:
  - Допускается преобразование в старые форматы файлов, но текущее содержание документа не может быть изменено или отредактировано каким-либо образом.
  - Разрешена подготовка печатных копий для личного использования.
  - Для любых других целей, таких как продажа печатных копий либо использование руководства или его частей в других публикациях, необходимо письменное согласие компании MySQL AB.
  - За дополнительной информацией или при возникновении желания принять участие в переводе документации присылайте предложения по адресу [docs@mysql.com](mailto:docs@mysql.com).

Сведения о действии лицензий MySQL на практике представлены в разделах 1.4.3 и 1.4.4 настоящего руководства.

## 1.4.3. Лицензии на MySQL

Программное обеспечение MySQL распространяется на условиях лицензии GNU General Public License (GPL), которая является, вероятно, наилучшей лицензией для систем с открытым исходным кодом. Формальные условия лицензии GPL доступны по адресу <http://www.fsf.org/licenses/>. Имеет смысл также просмотреть информацию по адресам <http://www.fsf.org/licenses/gplfaq.html> и <http://www.gnu.org/philosophy/enforcing-gpl.html>.

Наша лицензия GPL предполагает ряд необязательных исключений, которые позволяют многим приложениям, распространяемым на условиях Free/Libre and Open Source ("FLOSS"), включать в себя клиентские библиотеки MySQL, несмотря на тот факт, что не все лицензии FLOSS совместимы с GPL. Более подробную информацию по этому поводу можно найти на странице:

<http://www.mysql.com/products/licensing/foss-exception.html>

Поскольку программное обеспечение MySQL распространяется на условиях GPL, оно зачастую может использоваться бесплатно, однако для некоторых целей требуется коммерческая лицензия MySQL AB. Приобрести упомянутую лицензию можно по адресу <https://order.mysql.com/>. Просмотрите также информацию по адресу <http://www.mysql.com/products/licensing.html>.

Версии MySQL, предшествующие MySQL 3.22, подпадают под действие более строгих лицензий (<http://www.mysql.com/products/mypl.html>). Дополнительная информация доступна в документации по MySQL соответствующих версий.

Обратите внимание на то, что использование программного обеспечения MySQL по коммерческой лицензии, GPL либо по старой лицензии MySQL не предоставляет автоматически права на использование торговой марки компании MySQL AB (см. раздел 1.4.4).

### 1.4.3.1. Использование программного обеспечения MySQL по коммерческой лицензии

Лицензию GPL можно назвать “заразной” в том смысле, что когда программа связывается с GPL-программой, все исходные тексты всех частей результирующего продукта также подпадают под действие лицензии GPL. Если это требование не выполняется, тем самым нарушаются условия лицензии, а право на использование GPL-программы утрачивается. В этом случае вы рискуете тем, что от вас могут потребовать материальной компенсации за нарушение условий лицензии.

Коммерческая лицензия необходима в перечисленных ниже случаях.

- Если вы компонуете свою программу с любым кодом из состава программного обеспечения MySQL, который подпадает под действие лицензии GPL, и не хотите, чтобы результирующий продукт лицензировался на условиях GPL. Возможно, это необходимо из-за того, что вы разрабатываете коммерческий продукт либо по другим причинам желаете сохранить добавляемый вами код закрытым. Приобретая коммерческую лицензию, вы используете то же самое программное обеспечение MySQL, но только не в соответствии с лицензией GPL.
- Если вы распространяете приложение, использование которого не регламентируется лицензией GPL, но которое работает *только* с программным обеспечением MySQL, и поставляете это приложение вместе с программным обеспечением MySQL. Этот вариант решения рассматривается как компоновка, даже если части результирующего продукта взаимодействуют только через сеть.
- Если вы распространяете копии программного обеспечения MySQL без поставки исходного кода, как того требует лицензия GPL.
- Если вы хотите содействовать дальнейшей разработке СУБД MySQL, даже когда коммерческая лицензия формально не нужна. Оплата услуг по поддержке непосредственно компании MySQL AB – еще один хороший способ содействия разработке программного обеспечения MySQL, причем с непосредственной выгодой для вас (см. раздел 1.4.1).

Наша лицензия GPL предполагает ряд необязательных исключений, которые позволяют многим приложениям, распространяемым на условиях Free/Libre and Open Source (“FLOSS”), включать в себя клиентские библиотеки MySQL, несмотря на тот факт, что не все лицензии FLOSS совместимы с GPL. Более подробную информацию по этому поводу можно найти на странице:

<http://www.mysql.com/products/licensing/foss-exception.html>

Если вы решили отдать предпочтение коммерческой лицензии, вам понадобится отдельная лицензия на каждую установку MySQL. Ее действие распространяется на системы с любым количеством процессоров и с любым числом клиентов, которые подключаются к серверу, причем любым способом.

Для приобретения коммерческих лицензий посетите наш Web-сайт по адресу <http://www.mysql.com/products/licensing.html>. Чтобы заключить договор поддержки, обращайтесь по адресу <http://www.mysql.com/support/>. Если имеются какие-то специфические требования, свяжитесь с нашим персоналом, который занимается продажами, по электронной почте: [sales@mysql.com](mailto:sales@mysql.com).

### 1.4.3.2. Бесплатное использование программного обеспечения MySQL по лицензии GPL

Бесплатное использование программного обеспечения MySQL по лицензии GPL допускается при условии выполнения требований GPL. Дополнительная информация о GPL, в том числе ответы на часто задаваемые вопросы, доступна в разделе FAQ фонда бесплатного программного обеспечения (Free Software Foundation) по адресу <http://www.fsf.org/licenses/gpl-faq.html>.

Наша лицензия GPL предусматривает ряд необязательных исключений, которые позволяют многим приложениям, распространяемым на условиях Free/Libre and Open Source ("FLOSS"), включать в себя клиентские библиотеки MySQL, несмотря на тот факт, что не все лицензии FLOSS совместимы с GPL. Более подробно об этом см. <http://www.mysql.com/products/licensing/foss-exception.html>.

Ниже перечислены общие примеры использования GPL.

- Когда вы распространяете собственное приложение вместе с исходными текстами и исходным кодом MySQL на условиях GPL.
- Когда вы распространяете исходный код MySQL в связке с другими программами, которые не компонуется и функционально не зависят от MySQL, даже если эти программы распространяются на коммерческой основе. В лицензии GPL это называется "чистой агрегацией" ("mere aggregation").
- Когда вы не распространяете *никаких* частей системы MySQL, вы можете ее использовать бесплатно.
- Если вы являетесь поставщиком Internet-услуг, предоставляя своим клиентам Web-хостинг и доступ к серверу MySQL. Мы рекомендуем сотрудничать с поставщиками, у которых заключен договор поддержки MySQL, поскольку это дает уверенность в том, что поставщик располагает всеми ресурсами для решения любых возникающих проблем с MySQL. Даже если поставщик Internet-услуг не имеет коммерческой лицензии на сервер MySQL, его клиенты, как минимум, должны иметь доступ по чтению к исходным кодам установки MySQL, дабы они могли убедиться, что выполнены все изменения и исправления.
- Когда вы используете программное обеспечение баз данных MySQL в комплекте с Web-сервером, вы не нуждаетесь в коммерческой лицензии (до тех пор, пока не это является распространяемым вами продуктом). Это верно даже в том случае, если функционирующий Web-сервер является коммерческим, и он использует сервер MySQL, поскольку распространение частей системы MySQL не осуществляется. Однако в этом случае рекомендуется заключить договор поддержки MySQL, так как программное обеспечение MySQL оказывает влияние на успешность вашего предприятия.

Если вы используете программное обеспечение баз данных MySQL и не нуждаетесь в коммерческой лицензии, мы все равно рекомендуем вам заключить договор поддержки с компанией MySQL AB. Это способ содействия дальнейшему развитию MySQL и одновременно получения непосредственной выгоды для вас (см. раздел 1.4.1).

Если программное обеспечение баз данных MySQL применяется в коммерческих целях и от этого появляется некоторая прибыль, компания MySQL AB предлагает поддерживать дальнейшее развитие MySQL за счет заключения договора на поддержку определенного уровня. Мы полагаем, что если СУБД MySQL помогает вашему бизнесу, то логично попросить вас о содействии компании MySQL AB. (В противном случае получается, что задавая вопросы, касающиеся поддержки, вы не только бесплатно пользуетесь тем, во что вложена немалая работа, но и просите предоставить бесплатную поддержку.)

## 1.4.4. Логотипы и торговые марки MySQL AB

Многие пользователи СУБД MySQL желают размещать логотип с дельфином MySQL AB на своих Web-сайтах, в книгах или на коробках с программными продуктами. Мы приветствуем и поддерживаем это стремление, только при условии упоминания, что слово “MySQL” и логотип дельфина являются торговыми марками MySQL AB и могут использовать только так, как того требует наша политика относительно торговых марок (см. <http://www.mysql.com/company/trademark.html>).

### 1.4.4.1. Оригинальный логотип MySQL

Логотип MySQL с изображением дельфина был разработан финским рекламным агентством Priority в 2001 году. Дельфин был признан подходящим символом для СУБД MySQL – столь же умное, быстрое, подвижное и свободно ориентирующееся в безбрежном океане данных существо.

Оригинальный логотип MySQL может использоваться только как символ компании MySQL AB и только теми, кто имеет соответствующее письменное разрешение.

### 1.4.4.2. Логотипы MySQL, которые можно использовать без письменного разрешения

Мы разработали набор специальных *условно используемых* логотипов, которые можно загрузить с нашего Web-сайта (<http://www.mysql.com/press/logos.html>) и использовать на Web-сайтах независимых разработчиков без заключения письменных соглашений с компанией MySQL AB. Применение этих логотипов не является полностью неограниченным, а регламентируется политикой относительно торговых марок, с которой можно ознакомиться на нашем сайте. Если вы планируете использовать их, внимательно изучите все положения упомянутой политики. Ниже перечислены основные требования.

- Используйте требуемый логотип именно в том виде, в котором он представлен на сайте <http://www.mysql.com>. Разрешено масштабировать его до нужных размеров, однако нельзя изменять его цвета, дизайн, а также вносить любые другие изменения в этот графический образ.
- Явно укажите, что именно вы, а не компания MySQL AB являетесь создателем и владельцем сайта, на котором представлено изображение торговой марки MySQL.

- Не допускается использовать торговую марку способом, приносящим ущерб компании MySQL AB или ее торговым маркам. Мы оставляем за собой право отбирать права на использование торговых марок MySQL AB.
- При размещении изображения торговой марки на Web-сайте, предусмотрите ссылку, ведущую непосредственно на сайт <http://www.mysql.com>.
- Если вы используете СУБД MySQL на условиях лицензии GPL в приложении, это приложение должно быть с открытым исходным кодом и должно иметь возможность подключаться к серверу MySQL.

В случае если вас интересуют какие-то специальные условия, которые соответствуют вашим нуждам, свяжитесь с нами по электронной почте: [trademark@mysql.com](mailto:trademark@mysql.com).

#### **1.4.4.3. Когда необходимо иметь письменное разрешение на использование логотипов MySQL**

Письменное разрешение MySQL AB на использование логотипов MySQL необходимо получать в следующих случаях:

- При использовании логотипов MySQL AB в любом другом месте, отличном от Web-сайта.
- При использовании логотипа MySQL AB, не входящего в набор специальных *условно используемых* логотипов (о которых упоминалось выше) на Web-сайте или где-то еще.

На основе юридических и коммерческих соображений мы следим за использованием торговой марки MySQL в продуктах, книгах и других местах. Обычно мы требуем плату за изображение логотипа MySQL в коммерческих продуктах, поскольку считаем справедливым, чтобы некоторая часть дохода возвращалась нам для обеспечения дальнейшего развития СУБД MySQL.

#### **1.4.4.4. Партнерские логотипы MySQL AB**

Логотип партнера MySQL AB может использоваться только компанией, заключившей письменное соглашение о партнерстве с MySQL AB. Партнерские отношения предусматривают сертификацию кого-либо в качестве инструктора или консультанта по MySQL. Подробную информацию можно найти в разделе 1.3.1.5.

#### **1.4.4.5. Использование слова “MySQL” в печатном тексте и презентациях**

Компания MySQL AB приветствует ссылки на СУБД MySQL, однако при условии упоминания о том, что слово “MySQL” является торговой маркой MySQL AB. По этой причине к первому или наиболее заметному упоминанию слова “MySQL” в тексте должен быть добавлен значок торговой марки (®), а там, где это уместно, должно быть указано, что MySQL представляет собой торговую марку компании MySQL AB. За более подробной информацией о нашей политике относительно торговых марок обращайтесь по адресу <http://www.mysql.com/company/trademark.html>.

#### **1.4.4.6. Использование слова “MySQL” в названиях компаний и продуктов**

Использование слова “MySQL” в названиях компаний, продуктов или доменных именах Internet без письменного разрешения компании MySQL AB не допускается.



## 1.5. План разработки MySQL

В этом разделе в общих чертах представлен план разработки MySQL, включая основные средства, реализованные или планируемые для MySQL версий 4.0, 4.1, 5.0 и 5.1. Последующие разделы дают информацию о каждой серии выпусков.

Серия производственных выпусков на момент написания этой книги – это MySQL 4.0, которая была представлена как устойчивая версия 4.0.12, ориентированная на производственное применение (выпущена в марте 2003 года). Это означает, что дальнейшие разработки в рамках линейки 4.0 будут сводиться только к исправлению ошибок. В старой серии MySQL 3.23 будут исправляться только критические ошибки.

Активная разработка MySQL сейчас сосредоточена на сериях MySQL 4.1 и MySQL 5.0. Это означает, что новые возможности будут добавляться только к MySQL 4.1 и MySQL 5.0. На момент написания книги MySQL 4.1 доступен в виде бета-версии, а MySQL 5.0 – в виде альфа-версии.

Ниже подытожены планы реализации наиболее востребованных возможностей.

Возможность	Серия MySQL
Объединения	4.0
Подзапросы	4.1
R-деревья	4.1 (для таблиц MyISAM)
Хранимые процедуры	5.0
Представления	5.0
Курсоры	5.0
Внешние ключи	5.1 (уже реализованы в версии 3.23 для InnoDB)
Триггеры	5.1
Полные внешние соединения	5.1
Ограничения	5.1

### 1.5.1. Кратко о MySQL 4.0

Долгожданная версия MySQL 4.0 теперь доступна как производственный продукт. Ее можно загрузить из сайта <http://dev.mysql.com>, а также с наших зеркальных сайтов.

Версия MySQL 4.0 была протестирована большим числом пользователей и эксплуатируется на многих крупных Web-сайтах.

Основные возможности, вошедшие в состав сервера MySQL 4.0, были разработаны в соответствии с текущими требованиями нашего бизнеса и потребностями сообщества пользователей. Новые функции совершенствуют сервер баз данных MySQL как решение для ответственных, сильно загруженных систем баз данных. Другие новые средства ориентированы на пользователей встроенных баз данных.

#### 1.5.1.1. Возможности, доступные в MySQL 4.0

- Повышение скорости работы.
  - В версии MySQL 4.0 реализован кэш запросов, который обеспечивает существенный рост производительности для приложений, генерирующих повторяющиеся запросы.

- В версии MySQL 4.0 еще более увеличилась скорость выполнения многих операций, таких как пакетные операторы INSERT, поиск по упакованным индексам, полнотекстовый поиск (с использованием индексов FULLTEXT), а также COUNT (DISTINCT).
- Новый встроенный сервер MySQL.
  - С помощью новой библиотеки встроенного сервера можно легко создавать автономные и встроенные приложения. Встроенный сервер – это альтернатива MySQL в клиент-серверной среде.
- Механизм хранения InnoDB в качестве стандарта.
  - Механизм хранения InnoDB теперь позиционируется как стандартная функциональная возможность сервера MySQL. Это означает полную поддержку ACID-транзакций, внешних ключей с каскадными операторами INSERT и DELETE, а также блокировок на уровне строки.
- Новая функциональность.
  - Усовершенствованные поисковые свойства FULLTEXT в сервере MySQL 4.0 позволяют индексировать большие объемы текстовой информации, как для логики бинарного поиска, так и для поиска с применением естественного языка. Имеется возможность настройки минимальной длины слова и определения собственных списков недопустимых слов на любом естественном языке, что позволяет разрабатывать новый набор приложений, использующих MySQL-сервер.
- Соответствие стандартам, переносимость и миграция
  - Сервер MySQL теперь поддерживает оператор UNION – долгожданную возможность стандартного языка SQL.
  - MySQL теперь функционирует и на платформе Novell Netware, начиная с версии NetWare 6.0.
  - Средства, упрощающие миграцию из других баз данных в среду сервера MySQL, включая TRUNCATE TABLE (как у Oracle).
- Интернационализация
  - Немецкие, австрийские и швейцарские пользователи обратят внимание, что MySQL 4.0 поддерживает новый набор символов – latin1\_de, гарантирующий, что *порядок сортировки немецких символов* расположит слова с умляутиками в порядке, принятом в немецких телефонных справочниках.
- Удобство использования

В процессе реализации новых средств для новых пользователей мы не забываем и о запросах со стороны сообщества постоянных пользователей наших продуктов.

  - Большинство параметров mysqld (опций запуска) теперь могут устанавливаться без необходимости остановки сервера. Это удобное средство для администраторов баз данных.
  - Добавлены многотабличные операторы DELETE и UPDATE.
  - В среде Windows управление символическими ссылками на уровне базы данных теперь по умолчанию включено. В среде UNIX механизм хранения MyISAM теперь поддерживает символические ссылки на уровне таблиц (а не только на уровне базы данных, как ранее).

- Новые функции `SQL_CALC_FOUND_ROWS` и `FOUND_ROWS()` позволяют найти количество строк, которое должен вернуть оператор `SELECT` с конструкцией `LIMIT`, как если бы этой конструкции не было.

В разделе новостей онлайн-руководства можно найти более детальный список возможностей (см. <http://dev.mysql.com/doc/mysql/en/News.html>).

### 1.5.1.2. Встроенный сервер MySQL

Библиотека встроенного сервера `libmysqld` существенно расширяет сферу применения MySQL. Используя эту библиотеку, разработчики могут встраивать сервер MySQL в различные приложения и электронные устройства, при этом конечные пользователи могут даже не подозревать о лежащей в основе СУБД. Встроенный сервер MySQL идеален для использования в Internet-приложениях, общедоступных киосках, программно-аппаратных устройствах, высокопроизводительных Internet-серверах, автономных базах данных, распространяемых на компакт-дисках, и так далее.

Многие пользователи `libmysqld` получают несомненный выигрыш от двойного лицензирования MySQL. Для тех, кто не желает быть связанным ограничениями лицензии GPL, это программное обеспечение доступно по коммерческой лицензии. Библиотека встроенного MySQL использует тот же интерфейс, что и обычная клиентская библиотека, поэтому она проста и удобна в эксплуатации.

## 1.5.2. Кратко о MySQL 4.1

Сервер MySQL версии 4.0 заложил основу для появления новых возможностей, таких как подзапросы и поддержка Unicode, которые реализованы в версии 4.1, а также для работы над хранимыми процедурами, которая завершается в версии 5.0. Упомянутые возможности возглавляют список пожеланий большинства наших клиентов.

С появлением этих усовершенствований критикам MySQL понадобится проявить гораздо большее воображение, чтобы указать на недостатки в СУБД MySQL. Уже широко зарекомендовавший себя как стабильный, быстрый и простой в эксплуатации, сервер MySQL теперь готов к тому, чтобы удовлетворить требования самых взыскательных потребителей.

### 1.5.2.1. Средства, доступные в MySQL 4.1

Все возможности MySQL 4.1, описанные в данном разделе, уже реализованы. Несколько других средств MySQL 4.1 только планируются к реализации (см. раздел 1.6.1).

Набор возможностей, добавленных в версии 4.1, в основном утвержден. Большинство новых средств, пребывающих в состоянии разработки, уже доступны или будут доступны в MySQL 5.0 (см. раздел 1.6.2).

MySQL 4.1 в настоящее время находится в стадии бета-тестирования, и его бинарные файлы доступны для загрузки по адресу:

<http://dev.mysql.com/downloads/mysql/4.1.html>

Все бинарные реализации прошли интенсивное тестирование без каких-либо ошибок на тех платформах, для которых они собраны.

Для тех, кто желает использовать наиболее актуальные исходные тексты, находящиеся в процессе разработки, открыт доступ к нашему репозиторию BitKeeper для MySQL 4.1.

MySQL 4.1 проходит стадию *альфа-тестирования* (в течение которой новые средства могут быть добавлены или изменены), стадию *бета-тестирования* (когда новая разработка замораживается и выполняется только исправление ошибок) и стадию *гамма-тестирования* (означающую, что производственный выпуск должен появиться в течение нескольких недель). В конце этого процесса MySQL 4.1 становится новым производственным выпуском.

- Поддержка подзапросов и порожденных таблиц.
  - “Подзапрос” – это оператор `SELECT`, вложенный внутри другого оператора. “Порожденная таблица” (неименованное представление) – это подзапрос в конструкции `FROM` другого оператора.
- Увеличение скорости.
  - Ускоренный бинарный клиент-серверный протокол с поддержкой предварительно подготовленных операторов и связывание параметров.
  - Индексация `BTREE` теперь поддерживается для таблиц `HEAP`, значительно снижая время реакции при нечетком поиске.
- Новая функциональность.
  - Оператор `CREATE TABLE имя_таблицы2 LIKE имя_таблицы1` позволяет с помощью единственного оператора создавать новую таблицу со структурой, точно такой же, как у существующей таблицы.
  - Механизм хранения `MyISAM` теперь поддерживает пространственные типы `OpenGIS` для хранения геометрических данных.
  - Репликация может выполняться через `SSL`-соединения.
- Соответствие стандартам, переносимость и миграция
  - Новый клиент-серверный протокол добавляет возможность передачи множественных предупреждений и сообщений клиенту вместо единственного результата, как было раньше. Это упрощает процесс поиска причин проблем, которые могут возникнуть в таких операциях, как пакетная загрузка данных.
  - `SHOW WARNINGS` показывает предупреждения, касающиеся последней выполненной команды.
- Интернационализация
  - Для поддержки приложений, требующих использования национальных языков, программное обеспечение MySQL обеспечивает работу с кодировкой `Unicode` через символьные наборы `utf8` и `ucs2`.
  - Символьные наборы теперь можно задавать для столбца, таблицы и базы данных. Это обеспечивает высокую степень гибкости при проектировании приложений, в частности, многоязычных `Web`-сайтов.
- Удобство использования
  - В ответ на многочисленные просьбы мы добавили команду `HELP` серверной стороны, которая может предоставлять вспомогательную информацию об операторах `SQL`. Выгода от нахождения этой информации на стороне сервера заключается в том, что клиенту всегда доступна справка по той версии сервера, к которому он подключился. Поскольку эта информация доступна через оператор `SQL`, любой клиент можно написать так, чтобы к ней был доступ. Напри-

мер, команда `help` в клиенте командной строки `mysql` была соответствующим образом модифицирована.

- В новом клиент-серверном протоколе множественные операторы могут быть отправлены одним вызовом.
- Новый клиент-серверный протокол также поддерживает возврат множества результирующих наборов. Это может случиться, например, как результат множественного запроса.
- Реализован новый синтаксис `INSERT...ON DUPLICATE KEY UPDATE...`, который позволяет выполнять оператор `UPDATE` для существующей записи, если оператор `INSERT` может привести к дублированию поля, служащего первичным ключом или ключом уникального индекса.
- Введена новая агрегатная функция `GROUP_CONCAT()`, которая добавляет исключительно удобную возможность соединения значений столбцов из группированных строк в единственную результирующую строку.

В разделе новостей онлайн-руководства можно найти более детальный список возможностей (см. <http://dev.mysql.com/doc/mysql/en/News.html>).

### 1.5.3. MySQL 5.0: Очередной разрабатываемый выпуск

В настоящее время процесс разработки MySQL сфокусирован на выпуске 5.0, который будет оснащен хранимыми процедурами и другими новыми возможностями (см. раздел 1.6.2).

Для тех, кто желает взглянуть на передний край разработки MySQL, открыт публичный доступ к репозиторию BitKeeper для MySQL 5.0. Начиная с декабря 2003 года, доступны также бинарные сборки версии 5.0.

## 1.6. MySQL и будущее (списки TODO)

В этом разделе описаны возможности, которые запланированы к реализации в сервере MySQL. Позиции перечисляются по порядку номеров выпусков. Внутри списка позиции следуют в том порядке, в котором, предположительно, они будут реализованы.

### На заметку!

Если вы пользователь уровня предприятия, испытывающий срочную потребность в каком-то конкретном средстве, свяжитесь с нашими специалистами по электронной почте ([sales@mysql.com](mailto:sales@mysql.com)), чтобы обсудить возможности спонсирования. Целевое финансирование компаниями-спонсорами позволяет нам выделять дополнительные ресурсы для специфических целей. Одним из примеров реализации такого сотрудничества, имевшего место в прошлом, является репликация.

### 1.6.1. Новые средства, запланированные для версии 4.1

Перечисленные ниже средства в MySQL 4.1 пока не реализованы, но планируются к реализации до того момента, как MySQL 4.1 достигнет фазы *бета-тестирования*. Список того, что уже реализовано в MySQL 4.1, представлен в разделе 1.5.2.1.

- Стабильная поддержка OpenSSL. (Поддержка SSL в MySQL 4.0 была в зачаточном состоянии и не на 100% протестирована.)
- Дополнительно протестирован механизм предварительно подготовленных операторов.

- Дополнительно протестирована поддержка множественных символьных наборов в одной таблице.

## 1.6.2. Новые средства, запланированные для версии 5.0

Перечисленные ниже средства планируются к включению в состав MySQL 5.0. Некоторые из них, например, хранимые процедуры, уже готовы и включены в выпуск MySQL 5.0 alpha, который уже доступен сейчас. Другие, такие как курсоры, готовы лишь частично. Ожидается, что эти и некоторые другие средства будут поддерживаться в будущих выпусках.

Следует отметить, что поскольку мы имеем дело с множеством разработчиков, которые заняты в разных проектах, то и количество дополнительных средств должно быть значительным. Есть небольшая вероятность того, что часть из них войдут в выпуск MySQL 4.1. Список того, что уже реализовано в MySQL 4.1, представлен в разделе 1.5.2.1.

Для тех, кто желает взглянуть на передний край разработки MySQL, открыт публичный доступ к репозиторию BitKeeper для MySQL 5.0. Начиная с декабря 2003 года, доступны также бинарные сборки версии 5.0.

- Хранимые процедуры.
  - Хранимые процедуры в настоящее время реализованы на базе стандарта SQL:2003.
- Новая функциональность.
  - Элементарная поддержка курсоров.
  - Возможность явного указания для таблиц MyISAM, что индекс должен быть построен как индекс RTREE (в MySQL 4.1 индексы RTREE используются внутренне для геометрических данных GIS, но не могут быть созданы по запросу).
  - Динамическая длина строк для таблиц MEMORY.
- Соответствие стандартам, переносимость и миграция
  - Добавлена полноценная поддержка типа VARCHAR (ширина столбцов свыше 255 символов без усечения завершающих пробелов). В настоящее время существует поддержка этого в механизме хранения MyISAM, но пока это недоступно на уровне пользователя.
- Увеличение скорости.
  - Оператору SHOW COLUMNS FROM имя\_таблицы (используется клиентом mysql для того, чтобы позволить расширение имен столбцов) не требуется открывать таблицу, а только файл определений. Это требует меньших затрат памяти и получается значительно быстрее.
  - Оператору DELETE на таблицах MyISAM теперь разрешено использовать кэш записей. Чтобы обеспечить это, понадобилось обновлять кэш записей потоков при обновлении файлов .MYD.
  - Улучшенная поддержка таблиц MEMORY:
    - Динамическая длина строк.
    - Ускоренное управление строками (меньше копирования).
- Удобство использования
  - Решение проблемы, возникающую при попытке выполнить RENAME TABLE для таблиц, включенных в активные таблицы MERGE (возможно повреждение таблиц).

В разделе новостей онлайн-руководства можно найти более детальный список возможностей (см. <http://dev.mysql.com/doc/mysql/en/News.html>).

### 1.6.3. Новые средства, запланированные для версии 5.1

- Новая функциональность.
  - Поддержка внешних ключей (FOREIGN KEY) для всех типов таблиц, а не только для InnoDB.
  - Ограничения уровня столбца.
  - Онлайн-резервное копирование с минимальным снижением производительности. Это позволит легко добавлять новые базы без необходимости останавливать репликацию.
- Увеличение скорости.
  - Новый формат файлов определения таблиц (.frm) и табличный кэш для определения таблиц. Это позволит выполнять более быстрые запросы к структурам таблиц и увеличить эффективность поддержки внешних ключей.
  - Оптимизация типа BIT для хранения одного бита (в настоящее время тип BIT хранится в байте и рассматривается как синоним TINYINT).
- Удобство использования.
  - Добавление опций клиент-серверного протокола для получения информации о процессе выполнения команд, занимающих длительное время.
  - Реализация оператора RENAME DATABASE. Чтобы сделать это безопасным для всех механизмов хранения, он должен работать следующим образом:
    1. Создать новую базу данных.
    2. Переименовать каждую таблицу в другую базу, как это делается командой RENAME.
    3. Удалить старую базу.
  - Изменение нового внутреннего интерфейса файлов. Это обобщит управление всеми файлами и станет возможным добавление расширений, подобных RAID.

### 1.6.4. Новые средства, запланированные на ближайшее будущее

- Новая функциональность.
  - Представления, реализованные в пошаговой манере, вплоть до полной функциональности.
  - Подобные Oracle конструкции CONNECT BY PRIOR оператора SELECT для извлечения древовидных иерархических структур.
  - Добавление всех пропущенных стандартных типов SQL и ODBC 3.0.
  - Добавление SUM(DISTINCT).
  - INSERT SQL\_CONCURRENT и mysqld --concurrent-insert для параллельной вставки в конец таблицы, если таблица заблокирована по чтению.

- Разрешение обновления переменных оператором UPDATE, например, UPDATE foo SET @a:=a+b, a=@a, b=@a+c.
- Когда пользовательские переменные изменены, разрешение использования их в конструкции GROUP BY, как показано в следующем примере: SELECT id, @a:=COUNT(\*), SUM(sum\_col)/@a FROM имя\_таблицы GROUP BY id.
- Добавление опции IMAGE к LOAD DATA INFILE, чтобы не обновлять столбцы TIMESTAMP и AUTO\_INCREMENT.
- Добавление синтаксиса LOAD DATA INFILE...UPDATE, работающего следующим образом:
  - Для таблиц с первичными ключами, если вводимая запись имеет значение первичного ключа, совпадающее с существующей записью, то последняя обновляется значениями столбцов вводимой записи. Столбцы, пропущенные во вводимой записи, остаются без изменений.
  - Для таблиц с первичными ключами, если вводимая запись не содержит первичного ключа, или же какая-то часть ключа пропущена, запись обрабатывается как LOAD DATA INFILE...REPLACE INTO.
- Изменение оператора LOAD DATA INFILE, чтобы стал возможным такой синтаксис:

```
LOAD DATA INFILE 'имя_файла.txt' INTO TABLE имя_таблицы
TEXT_FIELDS (текстовый_столбец1, текстовый_столбец2, текстовый_столбец3)
SET столбец_таблицы1=CONCAT(текстовый_столбец1, текстовый_столбец2),
    текстовый_столбец3=23
IGNORE текстовый_столбец3
```

Это может использоваться для того, чтобы пропустить лишние столбцы в текстовом файле или обновить столбцы на основе выражения, составленного из прочитанных данных.

- Новые функции для работы со столбцами типа SET:
  - ADD\_TO\_SET(значение, набор)
  - REMOVE\_FROM\_SET(значение, набор)
- В настоящее время, если выполнение mysql прерывается посреди запроса, нужно открыть другое соединение и уничтожить этот выполняющийся запрос. Необходимо сделать так, чтобы такую ситуацию обнаруживал и разрешал сам сервер.
- Добавление интерфейса механизма хранения для табличной информации таким образом, чтобы его можно было использовать как системную таблицу. Это может несколько снизить скорость, если запрашивается информация обо всех таблицах, однако существенно увеличится гибкость. Необходимо реализовать SHOW INFO FROM имя\_таблицы для базовой информации о таблице.
- Реализация SELECT a FROM имя\_таблицы1 LEFT JOIN имя\_таблицы2 USING (a); здесь предполагается, что a поступает из таблицы имя\_таблицы1.
- Добавление опций DELETE и REPLACE к оператору UPDATE (чтобы при возникновении ошибки, связанной с дублированием ключа во время обновления, строки удалялись).
- Изменение формата DATETIME, чтобы можно было хранить доли секунды.



- Обеспечение возможности использования новой библиотеки GNU regex вместо применяемой сейчас (новая библиотека значительно быстрее).
- Соответствие стандартам, переносимость и миграция.
  - Не добавлять автоматически значения по умолчанию к столбцам (DEFAULT). Генерировать ошибку для любого оператора INSERT, в котором пропущены значения столбцов, не имеющих значений DEFAULT.
  - Добавить групповые функции ANY(), EVERY() и SOME(). В стандартном языке SQL это работает только на столбцах с булевыми значениями, но мы можем расширить это для работы на всех столбцах или выражениях, трактуя нулевые значения как FALSE и ненулевые как TRUE.
  - Исправить тип возврата функции MAX(*столбец*), чтобы она возвращала тот же тип, что и ее аргумент:

```
mysql> CREATE TABLE t1 (a DATE);
mysql> INSERT INTO t1 VALUES (NOW());
mysql> CREATE TABLE t2 SELECT MAX(a) FROM t1;
mysql> SHOW COLUMNS FROM t2;
```
- Увеличение скорости.
  - Не разрешать создание большего, чем определено, количества потоков при запуске восстановления MyISAM в одно и то же время.
  - Изменить оператор INSERT INTO...SELECT так, чтобы можно было при желании использовать параллельные вставки.
  - Добавить опцию периодического сбрасывания ключевых страниц для таблиц с задержанными ключами, если они долго не используются.
  - Разрешить объединения на частях ключей (для целей оптимизации).
  - Добавить анализатор файлов протоколов, чтобы можно было извлекать информацию о том, какие таблицы используются наиболее часто, насколько часто запрашиваются многотабличные объединения и так далее. Это поможет пользователям идентифицировать то, что подлежит оптимизации для более эффективного выполнения запросов.
- Удобство использования.
  - Возвращать оригинальный тип столбца при выполнении SELECT MIN(column) ... GROUP BY.
  - Обеспечить возможность указывать long\_query\_time с точностью до миллисекунд.
  - Скомпоновать код myisampack с сервером, чтобы он мог выполнять операции PACK и COMPRESS.
  - Добавить временный буфер кэша ключей при выполнении INSERT, DELETE и UPDATE с тем, чтобы стало возможным восстановление в случае переполнения индексного файла.
  - Если выполняется оператор ALTER TABLE для таблицы, указанной через символическую ссылку и расположенной на другом диске, создавать временные таблицы на том же диске.

- Реализовать типы `DATE` и `DATETIME` так, чтобы они корректно обрабатывали информацию о временных зонах, тем самым упростив работу с датами в разных зонах.
- Исправить `configure`, чтобы все библиотеки (подобно `MyISAM`) могли быть скомпилированы без потоков.
- Разрешить применение пользовательских переменных в качестве аргументов `LIMIT`, например: `LIMIT @a, @b`.
- Добавить автоматический вывод `mysql` в Web-браузер.
- Добавить `LOCK DATABASES` (с различными опциями).
- Дополнительная информация для `SHOW STATUS`. Чтение и обновление записей. Запросы к отдельным таблицам и запросы к объединениям. Среднее количество таблиц в запросе. Количество запросов с конструкциями `ORDER BY` и `GROUP BY`.
- Операция копирования `mysqladmin copy база_данных новая_база_данных`; Это потребует добавления операции `COPY` в `mysqld`.
- Вывод списка процессов должен показывать количество запросов/потоков.
- `SHOW HOSTS` для вывода информации о кэше имен хостов.
- Изменить имена таблиц с пустой строки на `NULL` для вычисляемых столбцов.
- Не использовать `Item_copy_string` для числовых значений, чтобы избежать преобразования число-строка-число в случае наподобие:  

```
SELECT COUNT(*)*(id+0) FROM имя_таблицы GROUP BY id
```
- Изменить оператор `ALTER TABLE` таким образом, чтобы он не прерывал клиентов, выполняющих `INSERT DELAYED`.
- Внести исправление, чтобы при ссылке на столбцы в конструкции `UPDATE` они содержали старые значения, которые были до начала обновления.
- Новые операционные системы.
  - Перенести `MySQL` на платформу `LynxOS`.

### 1.6.5. Новые средства, запланированные на отдаленное будущее

- Реализация функции `get_changed_tables(тайм-аут, таблица1, таблица2, ...)`.
- Изменить чтение таблиц так, чтобы использовалась `mmap()`, где это возможно. В настоящее время `mmap()` используют только сжатые таблицы.
- Сделать код автоматических временных меток более изящным. Добавить автоматические временные метки в протокол обновлений с `SET TIMESTAMP=значение`;
- Использовать семафоры чтения-записи в некоторых местах для повышения скорости.
- Автоматически закрывать некоторые таблицы, если таблица, временная таблица или временный файл получают ошибку 23 (слишком много открытых файлов).
- Улучшенное распространение констант. Когда в выражении встречается `имя_столбца=n`, причем `n` – константа, заменять все вхождения `имя_столбца` в выражении на `n`. В настоящее время это выполняется только в некоторых случаях.

- Заменить все константные выражения вычисляемыми, если возможно.
- Оптимизировать сравнения *ключ = выражение*. В настоящее время оптимизируются только *ключ = столбец* и *ключ = константа*.
- Объединить некоторые функции копирования для получения более изящного кода.
- Заменить `sql_yacc.yy` на встроенный анализатор, чтобы уменьшить его размер и получить более удобную диагностику ошибок.
- Изменить анализатор таким образом, чтобы использовать только одно правило для разного числа аргументов функции.
- Использовать полные вычисленные имена в порядковой части (для СУБД Access 97).
- Реализовать MINUS, INTERSECT и FULL OUTER JOIN (сейчас поддерживаются UNION и LEFT|RIGHT OUTER JOIN).
- Разрешить использование `SQL_OPTION MAX_SELECT_TIME=значение` для указания временного ограничения в запросе.
- Сделать возможной запись протоколов обновлений в базу данных.
- Усовершенствовать LIMIT, чтобы обеспечить извлечение данных из конца результирующего набора.
- Организовать выдачу предупреждений клиентскими функциями подключения/чтения/записи.
- Обратите внимание на изменения в `mysqld_safe`: согласно стандарту FSSTND (которому старается следовать Debian), PID-файлы должны размещаться в `/var/run/имя_программы>.pid`, а файлы протоколов – в `/var/log`. Было бы неплохо, если бы можно было помещать “DATADIR” в первое объявление “pidfile” и “log”, чтобы местоположение этих файлов можно было менять с помощью единственного оператора.
- Разрешить клиенту запрашивать протоколирование.
- Разрешить оператору `LOAD DATA INFILE` читать файлы, сжатые с помощью gzip.
- Исправить сортировку и группирование столбцов BLOB (частично решено сейчас).
- Использовать семафоры при подсчете потоков. Сначала потребуется реализовать библиотеку семафоров для потоков MIT-pthreads.
- Добавить полную поддержку JOIN со скобками.
- В качестве альтернативы модели “один поток на соединение” управлять пулом потоков для управления запросами.
- Разрешить `GET_LOCK()` получать более одной блокировки. При этом обрабатывать возможные взаимные блокировки, к которым это усовершенствование может привести.

### 1.6.6. Новые средства, которые не планируются к реализации

Наша цель состоит в достижении как можно более полной совместимости со стандартом ANSI/ISO SQL. Нет таких средств, которые мы НЕ планируем реализовывать.

## 1.7. Источники информации по MySQL

### 1.7.1. Списки рассылки MySQL

В этом разделе представлены списки рассылки MySQL и руководство по их использованию. После того, как вы подпишетесь на список рассылки, вам начнет поступать вся корреспонденция из этого списка по электронной почте. Кроме того, вы сможете отправлять свои вопросы, а также ответы на вопросы других подписчиков.

#### 1.7.1.1. Перечень списков рассылки MySQL

Чтобы подписаться или отказаться от подписки на любой список рассылки, упоминаемый в настоящем разделе, зайдите на <http://lists.mysql.com>. Не посылайте запросов на подписку или отказ от подписки в любой из списков, поскольку такие письма автоматически рассылаются тысячам подписчиков.

Ваш локальный сайт может иметь множество подписчиков на списки рассылки MySQL. Если это так, имеет смысл завести локальные списки рассылки, чтобы сообщения, отправленные на [lists.mysql.com](http://lists.mysql.com), рассылались адресатам, занесенным в них. В этом случае обратитесь к системному администратору, чтобы он внес вас в локальный список рассылки MySQL.

Если вы желаете, чтобы сообщения из рассылки попадали в отдельный почтовый ящик в вашей почтовой программе, настройте фильтр на базе заголовков сообщений. Для идентификации этих сообщений можно использовать заголовки `List-ID:` или `Delivered-TO:`.

Ниже представлены списки рассылки MySQL.

- `announce`

Это рассылка объявлений о новых версиях MySQL и сопутствующих программ. Это рассылка с малой активностью; на нее должны быть подписаны все пользователи MySQL.

- `mysql`

Это основная рассылка для общих дискуссий по вопросам MySQL. Стоит заметить, что некоторые темы более подробно обсуждаются в специальных рассылках. Если вы отправите письмо не в ту рассылку, то, вероятно, не получите ответа.

- `mysql-digest`

Это рассылка `mysql` в форме дайджеста. Если вы подпишетесь на эту рассылку, то будете ежедневно получать группу сообщений одним письмом.

- `bugs`

Эта рассылка может быть интересна, если вы хотите быть в курсе сообщений о последней версии MySQL или желаете включиться в процесс поиска и исправления ошибок.

- `bugs-digest`

Это рассылка `bugs` в форме дайджеста.

- `internals`

Эта рассылка предназначена в основном для тех, кто имеет дело с кодом MySQL. Это также форум для дискуссий о разработке MySQL и рассылки исправлений.

- `internals-digest`

Это рассылка `internals` в форме дайджеста.

- `mysqldoc`

Эта рассылка для тех, кто работает над созданием документации MySQL: людей из MySQL AB, переводчиков и других членов сообщества.

- `mysqldoc-digest`

Это рассылка `mysqldoc` в форме дайджеста.

- `benchmarks`

Это рассылка для тех, кого интересуют вопросы производительности. Дискуссии сосредоточены вокруг производительности СУБД (не только MySQL), а также касаются более широких категорий, включая производительность ядра, файловых систем, дисковых систем и так далее.

- `benchmarks-digest`

Это рассылка `benchmarks` в форме дайджеста.

- `packagers`

Это рассылка для дискуссий об объединении в пакеты и распространении MySQL. В данном форуме участвуют те, кто занимается поддержкой распространения для обмена идеями о том, как формировать пакеты MySQL и как добиваться того, чтобы MySQL выглядел и работал насколько возможно единообразно на всех платформах и операционных системах.

- `packagers-digest`

Это рассылка `packagers` в форме дайджеста

- `java`

Это рассылка для дискуссий, связанных с сервером MySQL и языком Java. В основном здесь обсуждаются JDBC-драйвера, включая MySQL Connector/J.

- `java-digest`

Это рассылка `java` в форме дайджеста.

- `win32`

Этот список предназначен для обсуждения всего, что касается использования MySQL под управлением операционных систем семейства Microsoft, таких как Windows 9x, Me, NT, 2000 и XP.

- `win32-digest`

Это рассылка `win32` в форме дайджеста.

- `myodbc`

Эта рассылка относится ко всему, что связано с подключением к MySQL через ODBC.

- `myodbc-digest`

Это рассылка `myodbc` в форме дайджеста.

- `gui-tools`

Здесь обсуждаются инструменты MySQL с графическим интерфейсом пользователя, включая MySQL Administrator и графический клиент MySQL Control Center.

- `gui-tools-digest`

Это рассылка `gui-tools` в форме дайджеста.

- `plusplus`

Этот список рассылки касается вопросов программирования на C++ для MySQL.

- `plusplus-digest`

Это рассылка `plusplus` в форме дайджеста.

- `mysql-mysql-modules`

Эта рассылка для всех вопросов, связанных с поддержкой MySQL языка Perl, в частности, модулем `DBD:mysql`.

- `mysql-mysql-modules-digest`

Это рассылка `mysql-mysql-modules` в форме дайджеста.

Если вы не можете получить ответ на заданный вопрос в списках рассылки MySQL, выходом может быть заключение договора поддержки с компанией MySQL AB. Это позволит вам напрямую контактировать с разработчиками MySQL (см. раздел 1.4.1).

Ниже представлен перечень списков рассылки MySQL на других языках (кроме английского). Эти рассылки компанией MySQL AB не управляются.

- `mysql-france-subscribe@yahoogroups.com`

Список рассылки на французском языке.

- `list@tinc.net`

Список рассылки на корейском языке. Для подписки отправьте по адресу списка сообщение `subscribe mysql ваш@почтовый.адрес`.

- `mysql-de-request@lists.4t2.com`

Список рассылки на немецком языке. Для подписки отправьте по адресу списка сообщение `subscribe mysql-de ваш@почтовый.адрес`. Дополнительную информацию об этом списке можно найти по адресу <http://www.4t2.com/mysql>.

- `mysql-br-request@listas.linkway.com.br`

Список рассылки на португальском языке. Для подписки отправьте по адресу списка сообщение `subscribe mysql-br ваш@почтовый.адрес`.

- `mysql-alta@elistas.net`

Список рассылки на испанском языке. Для подписки отправьте по адресу списка сообщение `subscribe mysql ваш@почтовый.адрес`.

### 1.7.1.2. Как задавать вопросы и сообщать об ошибках

Прежде чем посылать вопрос или сообщение об ошибке, выполните следующие действия:

- Начните с поиска в онлайнном руководстве по MySQL на <http://dev.mysql.com/doc/>. Мы стараемся поддерживать это руководство в актуальном состоянии, часто обновляя его по мере решения обнаруженных проблем. Полезной может оказаться и хронология изменений (<http://dev.mysql.com/doc/mysql/en/News.html>), поскольку весьма вероятно, что в новых версиях та или иная проблема уже решена.
- Просмотрите базу данных ошибок, которая доступна по адресу <http://bugs.mysql.com/>. Возможно, ошибка, о которой вы собираетесь сообщить, уже обнаружена и исправлена.
- Поищите в архиве списков рассылки на <http://lists.mysql.com/>.
- Воспользуйтесь поисковой службой <http://www.mysql.com/search/> для поиска по всему Web-сайту MySQL AB, включая онлайнное руководство.

Если вы не можете найти ответ в справочном руководстве и архивах, обратитесь к местному эксперту по MySQL. Если и он не даст ответа на вопрос, то перед тем как связаться с нами, изучите приведенную ниже инструкцию по отправке писем в списки рассылки MySQL.

### 1.7.1.3. Как сообщать об ошибках и проблемах

Обычное место, куда нужно направлять сообщения об ошибках, это <http://bugs.mysql.com/> – адрес нашей базы данных ошибок. Эта база общедоступна, любой может ее просматривать и выполнять в ней поиск. Если вы зарегистрируетесь в системе, то также сможете вводить в нее новые сообщения.

Написание качественного отчета об ошибке требует немалого терпения, однако, составив его правильно, вы сэкономите как свое время, так и наше. Хороший отчет об ошибке должен содержать полное описание пути, приводящего к ее проявлению (случай тестирования); весьма вероятно, что мы исправим обнаруженную вами ошибку уже в очередном выпуске. Сведения, представленные в этом разделе, посвящены тому, как правильно составлять такие отчеты, дабы не пришлось впустую тратить время на то, что либо мало поможет нам, либо вообще не поможет.

Для генерации отчета об ошибке (или сообщения о любой проблеме) мы рекомендуем использовать сценарий `mysqlbug`. Упомянутый сценарий находится в каталоге `scripts` (исходного дистрибутива) и в каталоге `bin` (бинарного дистрибутива MySQL). Если запустить `mysqlbug` не удастся (например, если вы работаете в среде Windows), все равно важно включить всю необходимую информацию, указанную в настоящем разделе (и самое главное – описание операционной системы и версии MySQL).

Сценарий `mysqlbug` поможет сгенерировать отчет об ошибке, собрав большую часть информации автоматически, но кое-что важное придется ввести вручную. Внимательно прочтите настоящий раздел и убедитесь, что вся перечисленная здесь информация должным образом отражена в отчете.

Прежде всего, необходимо убедиться в наличии проблемы в последней производственной версии MySQL или версии, находящейся на стадии разработки. Любой может воспроизвести найденную ошибку, просто запустив `mysql test < файл_сценария` или же запустив Perl-сценарий, включенный в отчет об ошибке.

Все ошибки, отправленные в базу ошибок на <http://bugs.mysql.com/>, будут исправлены или документированы в следующем выпуске MySQL. Если исправление ошибки требует только небольших изменений в коде, возможно, будет разослано только исправление.

Если вы обнаружили существенную ошибку в системе безопасности MySQL, сообщение об этом необходимо прислать по адресу [security@mysql.com](mailto:security@mysql.com).

Если вы подготовили воспроизводимый отчет об ошибке, присылайте его в базу ошибок по адресу <http://bugs.mysql.com/>. Помните, что даже в этом случае желательно запустить сценарий `mysqlbug` для сбора информации о вашей системе. Любая ошибка, которую мы сможем воспроизвести, имеет хорошие шансы на то, чтобы быть исправленной в очередном выпуске MySQL.

Сообщения о проблемах иного рода можно отправлять в списки рассылки MySQL.

Помните, что мы можем ответить на сообщения, содержащие слишком много информации, но не можем ответить на те, что содержат ее слишком мало. Люди часто пропускают некоторые факты, поскольку думают, что имеют представление о причинах проблем, и предполагают, что детали значения не имеют. Хороший принцип, которым следует руководствоваться, может быть сформулирован следующим образом: если вы

сомневаетесь, стоит ли сообщать о чем-то, то сообщайте. Получится гораздо быстрее и проще, если вы напишете кучу дополнительных строк в сообщении, чем если нам придется запрашивать у вас дополнительную информацию, пропущенную в первичном сообщении, и ждать ответа.

Наиболее часто встречающиеся ошибки в отчетах таковы: (а) не указан номер версии используемого дистрибутива MySQL и (б) не полностью описана платформа, на которой установлен сервер MySQL (включая тип платформы и номер версии). Это чрезвычайно важная информация, и в 99 случаях из 100 сообщение об ошибке без нее бесполезно. Очень часто мы получаем вопросы вроде такого: "Почему у меня то-то и то-то не работает?" Потом выясняется, что возможность просто в данной версии MySQL не реализована, или же эта ошибка известна и исправлена в более новой версии MySQL. Иногда ошибка оказывается зависимой от платформы и в этом случае мы не в состоянии исправить что-либо, не зная операционной системы и номера ее версии.

Если вы скомпилировали MySQL из исходных текстов, не забудьте также указать информацию о компиляторе, если это связано с проблемой. Часто люди сталкиваются с ошибками компиляторов, а думают, что ошибки связаны с кодом MySQL. Большинство компиляторов находятся в постоянном процессе разработки и становятся лучше от версии к версии. Чтобы определить, не вызвана ли проблема компилятором, нам надо знать, какой компилятор вы используете. Помните, что любые проблемы с компиляцией должны рассматриваться как ошибки и сообщаться соответствующим образом.

Лучше всего, если в отчет об ошибке включено исчерпывающее описание проблемы. Имеется в виду описание всего того, что вы делали и столкнулись с ошибкой, а также подробное описание самой проблемы. То есть наилучшими отчетами об ошибках являются такие, которые содержат полный пример, показывающий, как воспроизвести описанную ошибку или проблему.

Если какая-то программа выдает сообщение об ошибке, очень важно включить это сообщение в отчет. Если мы попытаемся найти что-нибудь в архивах, лучше, чтобы сообщение об ошибке, сгенерированное программой, было точно в том виде, как вы его увидели (важен даже регистр символов). Никогда не пытайтесь воспроизвести по памяти это сообщение, а вместо этого просто скопируйте его и вставьте в отправляемый отчет.

Если у вас возникла проблема с Connector/ODBC (MyODBC), пожалуйста, попытайтесь сгенерировать трассировочный файл MyODBC и прислать его вместе с отчетом.

Помните, что многие люди, которые будут читать ваш отчет об ошибке, используют отображение с шириной 80 символов. Поэтому при генерации сообщения об ошибке с помощью `mysqlbug` применяйте опцию `--vertical` (или символ завершения операторов `\G`) для вывода, который может по ширине превысить общепринятые размеры (например, с оператором `EXPLAIN SELECT`, как показано выше в примере).

В отправляемый отчет должна быть включена следующая информация:

- Номер версии используемого дистрибутива MySQL, например, MySQL 4.0.12. Эту информацию можно получить, запустив `mysqladmin version`. Программа `mysqladmin` расположена в подкаталоге `bin` каталога с инсталляцией MySQL.
- Производитель и модель компьютера, на котором возникла проблема.
- Наименование и версия операционной системы. Если вы работаете под управлением Windows, получить эту информацию можно, выполнив двойной щелчок на пиктограмме My Computer (Мой компьютер) и выбрав в меню Help (Справка) пункт About (О программе). Для большинства Unix-подобных операционных систем эту информацию можно получить через команду `uname -a`.



- Иногда важен объем памяти (физической и виртуальной). При наличии каких-либо сомнений, включите в отчет и эту информацию.
- Если вы используете дистрибутив MySQL с исходными текстами, понадобится наименование и номер версии компилятора. Если дистрибутив только бинарный, потребуется его наименование.
- Если проблема возникает при компиляции, включите в отчет сообщение об ошибке компилятора и несколько строк из контекста, окружающего то место в исходном коде, где возникла ошибка.
- Если `mysqld` аварийно завершился, сообщите в отчете текст запроса, приводящего к таким последствиям. Обычно это можно сделать, запустив `mysqld` с включенной опцией протоколирования запросов и заглянув в файл протокола после отказа `mysqld`.
- Если к ошибке имеет отношение таблица базы данных, включите вывод `mysqldump --nodata имя_базы_данных имя_таблицы`. Это очень просто сделать и это отличный способ получить информацию о любой таблице в базе данных. В результате у нас появится возможность смоделировать сложившуюся у вас ситуацию.
- При описании проблем, имеющих отношение к скорости или к оператору `SELECT`, всегда необходимо включать вывод команды `EXPLAIN SELECT ...`, а также, по меньшей мере, число строк, которое возвращает оператор `SELECT`. Также следует включить вывод команды `SHOW CREATE TABLE имя_таблицы` для каждой таблицы, участвующей в запросе. Чем больше информации о ситуации вы сообщите, тем более вероятно, что кто-то сможет вам помочь. Ниже представлен пример очень хорошего отчета об ошибке. Он может быть отправлен через сценарий `mysqlbug`. Этот пример использует утилиту командной строки `mysql`. Отметьте применение ограничителя операторов `\G` для тех из них, чей вывод превышает ширину отображения в 80 символов:

```
mysql> SHOW VARIABLES;
mysql> SHOW COLUMNS FROM ... \G
      <вывод SHOW COLUMNS>
mysql> EXPLAIN SELECT ... \G
      <вывод EXPLAIN>
mysql> FLUSH STATUS;
mysql> SELECT ...;
      <Сокращенная версия вывода SELECT,
        включая время выполнения запроса>
mysql> SHOW STATUS;
      <вывод SHOW STATUS>
```

- Если ошибка или проблема возникает во время работы `mysqld`, постарайтесь предоставить сценарий, который воспроизводит аномалию. Этот сценарий должен включать все необходимые исходные файлы. Чем ближе к реальности сценарий сможет воспроизвести ситуацию, тем лучше. Если вы можете составить воспроизводимый случай тестирования, присылайте его по адресу <http://bugs.mysql.com/> для высокоприоритетной обработки. Если вы не можете предоставить сценарий, по крайней мере, включите в отчет вывод команды `mysqladmin variables extended-status processlist`, чтобы дать представление о том, как ваша система настроена.

- Если вы не можете предоставить случай тестирования в нескольких строках или же случай тестирования слишком большой, чтобы присылать его в группу рассылки (более 10 строк), вам потребуется сбросить дамп таблиц с помощью `mysqldump` и создать файл `README` с описанием проблемы. Создайте архив своих файлов с помощью утилит `tar`, `gzip` или `zip` и отправьте его через FTP-протокол по адресу `ftp://ftp.mysql.com/pub/mysql/upload/`. Затем введите описание проблемы в базу данных ошибок по адресу `http://bugs.mysql.com/`.
- Если вам кажется, что MySQL выдает странный результат запроса, включите не только собственно результат, а также ваше предположение, каким он должен быть, и опишите основания своих предположений относительно результата.
- При описании примера возникновения проблемы лучше использовать имена переменных, таблиц и так далее такими, какими они были в вашей ситуации, а не придумывать новые имена. Проблема может иметь отношение к имени переменной или таблицы. Это случается редко, но лучше подстраховаться и не вносить искажений в описание проблемы. К тому же вам будет проще, да и нам удобней, если вы приведете пример реальной ситуации. Если у вас есть данные, которые вы не хотите показывать всем, их можно отправить через FTP-протокол по адресу `ftp://ftp.mysql.com/pub/mysql/upload`. Если же информация настолько секретна, что вы не хотите ее показывать даже нам, тогда приводите пример с измененными именами, но это только в крайнем случае.
- Включите в отчет все опции всех программ, имеющих отношение к ошибочной ситуации, если это возможно. Так, например, укажите опции, которые вы использовали для запуска сервера `mysqld`, как и опции любых клиентских программ MySQL. Опции программ `mysqld`, `mysql`, сценария `configure` часто являются ключами к ответу и поэтому очень важны. Никогда не стоит пренебрегать этим. Если вы применяете модули, подобные Perl или PHP, пожалуйста, укажите номера их версий.
- Если ваш вопрос имеет отношение к системе привилегий, включите в отчет вывод утилит `mysqlaccess`, `mysqladmin reload` и все сообщения об ошибках, которые выдаются при попытке подключения. Когда вы проверяете существующие привилегии, то должны сначала запустить `mysqlaccess`. После этого запустите `mysqladmin reload version` и попытайтесь подключиться с помощью той программы, которая приводила к проблемам.
- Если вы располагаете модулем исправления ошибки, упомяните в отчете и о нем. Однако не ожидайте, что ваш модуль исправления – это все, что нам нужно или что мы его используем, особенно если переданный отчет об ошибке не включает в себя случай тестирования, который воспроизводит ошибку, фиксируемую модулем исправления. Мы можем обнаружить проблемы, связанные с вашим модулем исправления, или вообще не понять его. Если так случится, мы не станем его использовать. Если мы не можем проверить, для чего предназначен данный модуль исправления, мы также не станем его использовать. В этом нам помогут случаи тестирования. Покажите, что модуль исправления справляется со всеми ситуациями, которые могут возникнуть. Если мы обнаружим какие-то ограничения (даже очень редкие), при которых модуль исправления не работает, он может не пригодиться.

- Предположения о природе обнаруженной ошибки, причинах ее возникновения или зависимостях, как правило, неверны. Даже группа разработчиков MySQL не может делать каких-то предположений, пока не воспользуется средствами отладки для нахождения истинной причины ошибки.
- Отмечайте в отчете об ошибке, что вы просматривали руководство и архивы списков рассылки, чтобы остальные знали, что вы пытались решить проблему самостоятельно.
- Если вы получили сообщение об ошибке “parse error” (“ошибка синтаксического анализа”), тщательно проверьте код на предмет корректности синтаксиса. Если вы не находите в нем ничего некорректного, вполне возможно, что ваша версия сервера MySQL не поддерживает используемый вами синтаксис. Если вы используете текущую версию сервера и руководство на <http://dev.mysql.com/doc/> не описывает применяемый вами синтаксис, значит, сервер MySQL подобный запрос не поддерживает. В этом случае единственный выход для вас – реализовать требуемый синтаксис самостоятельно или отправить письмо по адресу [licensing@mysql.com](mailto:licensing@mysql.com) с просьбой реализовать его. Если в руководстве описан синтаксис, который вы применяете, но у вас более старая версия сервера MySQL, стоит просмотреть хронологию изменений MySQL, чтобы найти, когда этот синтаксис был реализован. В этом случае у вас есть возможность обновить сервер MySQL до более новой версии.
- Если возникшая проблема связана с повреждением данных, либо ошибки возникают при попытке обращения к отдельной таблице, потребуется сначала проверить, а затем восстановить таблицы с помощью команд `CHECK TABLE` и `REPAIR TABLE`, либо с помощью утилиты `myisamchk`. Если вы работаете в среде Windows, убедитесь, что команда `SHOW VARIABLES LIKE 'lower_case_table_names'` возвращает значение 1 или 2.
- Если повреждения таблиц случаются часто, попытайтесь выяснить, когда и почему это происходит. В этом случае протокол ошибок в словаре данных MySQL может содержать некоторую информацию о том, что происходит. (Это файл с суффиксом `.err` в имени). Включите в отчет об ошибке любую важную информацию из этого файла. Обычно `mysqld` никогда не портит таблиц, если только ничто не уничтожает его в процессе обновления данных. Если вы сможете выяснить причину краха `mysqld`, нам будет значительно легче помочь вам решить проблему.
- Если возможно, загрузите и установите самую последнюю версию сервера MySQL и проверьте, осталась ли проблема нерешенной. Все версии программного обеспечения MySQL тестируются очень тщательно и должны функционировать без проблем. Мы стараемся обеспечить обратную совместимость, насколько это возможно, поэтому переход на новую версию должен пройти без особых проблем.

Если вы – клиент, заключивший с нами договор поддержки, перешлите отчет об ошибке по адресу [mysql-support@mysql.com](mailto:mysql-support@mysql.com) для высокоприоритетной обработки, а также отправьте этот отчет в соответствующий список рассылки, чтобы проверить, не сталкивался ли кто-нибудь с подобной проблемой и, возможно, каким-то образом решил ее.

Если ответы направляются вам персонально, а не в группу рассылки, хорошим тоном считается резюмировать ответы и отправлять их в список рассылки, чтобы другие подписчики тоже получили пользу от ответов, которые помогли вам решить возникшую проблему.

#### 1.7.1.4. Рекомендации по составлению ответов на вопросы из списков рассылки

Если вы считаете, что ваш ответ может заинтересовать многих, возможно, имеет смысл отправить его в список рассылки вместо того, чтобы отвечать персонально тому, кто задал вопрос. Постарайтесь максимально обобщить ответ, дабы он принес пользу и другим, а не только тому, кто спрашивает. Отправляя ответ в список рассылки, убедитесь, что он не дублирует предыдущие ответы на тот же вопрос. Постарайтесь подытожить существенную часть вопроса в ответе, но не считайте необходимым полностью цитировать оригинальный вопрос.

Не отправляйте почтовых сообщений с помощью браузера с включенным режимом HTML. Многие пользователи читают письма, не прибегая к услугам браузера.

#### 1.7.2. Поддержка сообщества пользователей MySQL в IRC

В дополнение к спискам рассылки MySQL опытных пользователей можно найти и в IRC (Internet Relay Chat – беседы в Internet). Ниже перечислены лучшие сети/каналы, известные на данный момент.

- **freenode** (<http://www.freenode.net/>)
  - **#mysql**. В основном обсуждаются вопросы, связанные с MySQL, но можно задавать вопросы и по другим СУБД и SQL.
  - **#mysqlphp**. Вопросы, связанные с использованием популярной комбинации продуктов – MySQL + PHP.
  - **#mysqlperl**. Вопросы, связанные с использованием другой популярной комбинации продуктов – MySQL + Perl.
- **EFnet** (<http://www.efnet.org/>)
  - **#mysql**. Вопросы по MySQL.

Если вам необходимо программное обеспечение IRC-клиента для подключения к сетям IRC, рекомендуем воспользоваться X-Chat (<http://www.xchat.org/>). X-Chat (регламентируется лицензией GPL) доступен как для Unix-, так и для Windows-платформ.

### 1.8. Соответствие стандартам MySQL

В этом разделе представлена информация, касающаяся того, как MySQL соотносится со стандартами ANSI/ISO SQL. В сервере MySQL реализовано множество расширений стандарта SQL, и здесь вы найдете сведения о том, что они собой представляют и как их использовать. Кроме того, представлена информация о функциональности, которая отсутствует в сервере MySQL, а также о том, как преодолевать некоторые расхождения со стандартом.

Стандарт SQL появился в 1986 году и на сегодняшний день существует несколько его версий. В настоящем руководстве “SQL-92” ссылается на стандарт, изданный в 1992 году, “SQL:1999” – на стандарт, изданный в 1999 году, и “SQL:2003” – на текущую версию стандарта. Под “стандартом SQL” понимается последняя версия стандарта.

Наша цель состоит в том, чтобы не сужать рамки применения сервера MySQL без веских на то причин. Даже если у нас не хватает ресурсов для разработки, ориентированной на любое возможное применение, мы всегда стараемся оказать помощь людям, которые пытаются применять сервер MySQL в новых областях.

Одна из главных задач при разработке этого продукта состоит в том, чтобы продолжать работу в направлении максимального соответствия стандарту SQL, однако, не жертвуя при этом производительностью и надежностью. Мы не боимся добавлять собственные расширения к SQL или поддерживать не-SQL средства, если это значительно увеличивает удобство применения сервера MySQL для большого сегмента нашей пользовательской базы. Примером такой стратегии может служить интерфейс HANDLER в сервере MySQL 4.0.

Мы будем продолжать поддержку транзакционной и не-транзакционной баз данных, чтобы удовлетворить запросы как тех пользователей, которым нужна работа с ответственными данными по схеме “24 часа в сутки, 7 дней в неделю”, так и тех, кому необходима напряженная работа с Web и регистрацией.

Изначально сервер MySQL разрабатывался для баз данных средних размеров (10–100 миллионов записей, или около 100 Мбайт на таблицу) в малых компьютерных системах. Сегодня сервер MySQL обслуживает терабайтные базы данных, но его код по-прежнему может быть скомпилирован в ограниченную версию, применимую в портативных и встроенных системах. Компактный дизайн сервера MySQL делает возможным продолжение разработки в обоих направлениях, без каких-либо конфликтов в дереве исходного кода.

В настоящее время мы не планируем поддержку систем реального времени, но, несмотря на это, средства репликации MySQL уже предлагают достаточно развитую функциональность.

Поддержка кластеризованных баз данных планируется на основе интеграции приобретенной нами технологии NDB-кластеров с новым механизмом хранения, который стал доступным в 2004 году.

Мы также готовимся предоставить поддержку XML на сервере баз данных.

### 1.8.1. Стандарты, которым соответствует MySQL

Мы нацелены на реализацию полной поддержки стандарта ANSI/ISO, но без компромиссов в отношении производительности и качества кода.

ODBC уровней 0–3.51.

### 1.8.2. Выбор режимов SQL

Сервер MySQL может работать в различных режимах SQL и может по-разному применять эти режимы для различных клиентов. Это дает приложениям возможность приспособлять функционирование сервера к существующим требованиям.

Режимы определяют, какой синтаксис SQL должен поддерживать сервер MySQL и какой тип проверок он должен выполнять для данных. Это позволяет использовать MySQL во множестве различных сред, а также применять его вместе с другими серверами баз данных.

Режим SQL по умолчанию устанавливается во время запуска `mysqld` с помощью опции `--sql-mode="modes"`. Начиная с MySQL 4.1, можно изменять режим после запуска сервера путем установки переменной `sql_mode` с помощью оператора `SET [SESSION|GLOBAL] sql_mode='modes'`.

### 1.8.3. Запуск MySQL в режиме ANSI

Чтобы перевести `mysqld` в ANSI-режим, запустите его с опцией `--ansi`.

Выполнение сервера в режиме ANSI эквивалентно его запуску со следующими опциями (значения `--sql-mode` должны указываться в одной строке):

```
--transaction-isolation=SERIALIZABLE  
--sql-mode=REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,  
IGNORE_SPACE,ONLY_FULL_GROUP_BY
```

В MySQL 4.1 можно получить тот же результат с помощью следующих операторов:

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
SET GLOBAL sql_mode = 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,  
IGNORE_SPACE,ONLY_FULL_GROUP_BY';
```

См. раздел 1.8.2.

В MySQL 4.1.1 опция `sql_mode` может быть также установлена так, как показано ниже:

```
SET GLOBAL sql_mode='ansi';
```

В этом случае значением переменной `sql_mode` будут все опции, имеющие отношение к режиму ANSI. Вы можете проверить это так:

```
mysql> SET GLOBAL sql_mode='ansi';  
mysql> SELECT @@global.sql_mode;  
-> 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,  
IGNORE_SPACE,ONLY_FULL_GROUP_BY,ANSI';
```

## 1.8.4. Расширения стандартного SQL в MySQL

В состав сервера MySQL входит ряд расширений, которых, возможно, вы не найдете в других базах данных SQL. Помните, что если вы используете их, ваш код перестанет быть переносимым на другие серверы SQL. В некоторых случаях вы можете писать код, включающий расширения MySQL, но остающийся переносимым, используя для этого форму комментариев `/*! ... */`. В этом случае сервер MySQL разбирает и выполняет код внутри комментария, как и любые другие SQL-операторы, а все другие серверы SQL расширение проигнорируют. Например:

```
SELECT /*! STRAIGHT_JOIN */ имя_столбца FROM таблица1,таблица2 WHERE ...
```

Если после символа `!` добавить номер версии, синтаксис внутри комментария будет выполняться только сервером MySQL указанной и более поздних версий:

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

Это означает, что если работа выполняется в версии MySQL 3.23.02 или более поздней, то ключевое слово `TEMPORARY` будет использовано.

В приведенном ниже списке описаны расширения MySQL по категориям.

### ■ Организация данных на диске.

Сервер MySQL отображает каждую базу данных на подкаталог внутри каталога данных MySQL, а таблицы внутри базы – на имена файлов в этом каталоге. Отсюда вытекает несколько следствий:

- Имена баз данных и таблиц MySQL зависят от регистра в средах операционных систем, в которых имена файлов чувствительны к регистру символов (большинство Unix-систем).
- Можно использовать стандартные системные команды для резервного копирования, переименования, перемещения, удаления и копирования таблиц, управ-

ляемых механизмами хранения MyISAM или ISAM. Например, чтобы переименовать таблицу MyISAM, потребуется переименовать файлы .MYD, .MYI и .frm, которые относятся к таблице.

Имена баз данных, таблиц, индексов, столбцов и псевдонимы могут начинаться с цифры (но не должны состоять только из цифр).

■ **Общий синтаксис языка.**

- Строки могут ограничиваться и одиночными и двойными кавычками.
- Символ '\ ' используется в строках как управляющий.
- Внутри SQL-операторов можно получать доступ к таблицам из разных баз данных посредством синтаксиса *имя\_базы\_данных.имя\_таблицы*. Некоторые серверы SQL предоставляют ту же функциональность, но называют ее пространством пользователя (User space). Сервер MySQL не поддерживает табличных пространств, как в следующем операторе: `CREATE TABLE ralph.my_table...IN my_tablespace.`

■ **Синтаксис SQL-операторов.**

- Операторы `ANALYZE TABLE`, `CHECK TABLE`, `OPTIMIZE TABLE` и `REPAIR TABLE`.
- Операторы `CREATE DATABASE` и `DROP DATABASE`.
- Оператор `DO`.
- `EXPLAIN SELECT` — для получения описания способа объединения таблиц в запросе.
- Операторы `FLUSH` и `RESET`.
- Оператор `SET`.
- Оператор `SHOW`.
- `LOAD DATA INFILE`. Во многих случаях этот синтаксис совместим с аналогичным синтаксисом Oracle.
- `RENAME TABLE`.
- `REPLACE` вместо `DELETE + INSERT`.
- Конструкции `CHANGE имя_столбца`, `DROP имя_столбца`, `DROP INDEX`, `IGNORE` и `RENAME` в операторе `ALTER TABLE`. Использование множественных конструкций `ADD`, `ALTER`, `DROP` и `CHANGE` в операторе `ALTER TABLE`.
- Использование имен индексов, индексов в префиксах полей, а также конструкций `INDEX` или `KEY` в операторе `CREATE TABLE`.
- Использование `IF EXISTS` вместе с `DROP TABLE`.
- Можно удалять несколько таблиц одним оператором `DROP TABLE`.
- Конструкции `ORDER BY` и `LIMIT` в операторах `UPDATE` и `DELETE`.
- Синтаксис `INSERT INTO...SET имя_столбца = ...`.
- Конструкция `DELAYED` в операторах `INSERT` и `REPLACE`.
- Конструкция `LOW_PRIORITY` в операторах `INSERT`, `REPLACE`, `DELETE` и `UPDATE`.
- Использование `INTO OUTFILE` и `STRAIGHT_JOIN` в операторе `SELECT`.
- Опция `SQL_SMALL_RESULT` оператора `SELECT`.

- Нет необходимости перечислять все выбранные столбцы в конструкции GROUP BY. Это обеспечивает лучшую производительность для некоторых очень специфических, но вполне нормальных запросов.
- Можно применять ASC или DESC вместе с GROUP BY.
- Имеется возможность присваивать значения переменным с помощью операции присваивания := в операторах:

```
mysql> SELECT @a:=SUM(total),@b=COUNT(*),@a/@b AS avg
        -> FROM test_table;
mysql> SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
```

#### ■ Типы столбцов.

- Типы столбцов MEDIUMINT, SET, ENUM и различные варианты типов BLOB и TEXT.
- Атрибуты столбцов AUTO\_INCREMENT, BINARY, NULL, UNSIGNED и ZEROFILL.

#### ■ Функции и операции.

- Чтобы облегчить жизнь пользователям, привыкшим к другим SQL-средам, сервер MySQL поддерживает псевдонимы для многих функций. Например, все строковые функции поддерживают как стандартный синтаксис SQL, так и синтаксис ODBC.
- Сервер MySQL воспринимает операции && и || как логическое “И” (AND) и логическое “ИЛИ” (OR), по аналогии с языком программирования C. В контексте сервера MySQL операции || и OR являются синонимами, равно как и && и AND. По этой причине MySQL не поддерживает стандартную SQL-операцию || для конкатенации строк. Вместо этого необходимо применять функцию CONCAT(). Поскольку CONCAT() принимает любое количество аргументов, преобразовать все операции || очень легко.
- Использование COUNT(DISTINCT список), где список содержит более одного элемента.
- Все сравнения строк по умолчанию чувствительны к регистру, а порядок сортировки определяется текущим выбранным набором символов (по умолчанию ISO-8859-1 Latin1). Если это не подходит, потребуется объявить столбец с атрибутом BINARY либо воспользоваться приведением к BINARY, что заставит выполнять сравнение и сортировку в соответствии с кодами символов, а не в лексикографическом порядке.
- Операция % является синонимом функции MOD(). То есть, выражение  $N \% M$  эквивалентно MOD(N, M). ‘%’ поддерживается для удобства программистов на языке C и достижения совместимости с СУБД PostgreSQL.
- Операции =, <>, <=, <, >=, >, <<, >>, <=>, AND, OR и LIKE могут применяться для сравнения столбцов слева от конструкции FROM в операторах SELECT, например:
 

```
mysql> SELECT col1=1 AND col2=2 FROM имя_таблицы;
```
- Функция LAST\_INSERT\_ID() возвращает самое последнее значение AUTO\_INCREMENT.
- LIKE можно применять к числовым столбцам.
- Расширенные операции обработки регулярных выражений REGEXP и NOT REGEXP.



- Функции `CONCAT()` и `CHAR()` принимают один и более аргументов.
- Функции `BIT_COUNT()`, `CASE`, `ELT()`, `FROM_DAYS()`, `FORMAT()`, `IF()`, `PASSWORD()`, `ENCRYPT()`, `MD5()`, `ENCODE()`, `DECODE()`, `PERIOD_ADD()`, `PERIOD_DIFF()`, `TO_DAYS()` и `WEEKDAY()`.
- Применение `TRIM()` для усеечения подстрок. Стандартный язык SQL поддерживает только удаление последовательностей одинаковых символов.
- Возможность в конструкции `GROUP BY` обращаться к функциям `STD()`, `BIT_OR()`, `BIT_AND()`, `BIT_XOR()` и `GROUP_CONCAT()`.

Для ознакомления с перечнем новых расширений, которые планируется добавить в сервер MySQL, а также с их приоритетностью, просмотрите список "TODO" по адресу <http://dev.mysql.com/doc/mysql/en/TODO.html>. В настоящем руководстве представлена последняя на данный момент версия списка "TODO". См. также раздел 1.6.

## 1.8.5. Отличия MySQL от стандартного SQL

Мы стараемся, чтобы MySQL в основном следовал требованиям стандартов ANSI SQL и ODBC SQL, но в приведенных ниже случаях некоторые операции MySQL выполняет иначе:

- В столбцах типа `VARCHAR` завершающие пробелы удаляются при сохранении значения (см. раздел 1.8.7).
- В некоторых случаях столбцы типа `CHAR` скрыто преобразуются в `VARCHAR`, когда определяется либо изменяется структура таблицы.
- Привилегии для таблицы при удалении таблицы автоматически не удаляются. Для этого необходимо явно вызвать оператор `REVOKE`.

### 1.8.5.1. Подзапросы

MySQL 4.1 поддерживает подзапросы и вторичные таблицы. Подзапрос – это оператор `SELECT`, вложенный в другой оператор. Вторичная таблица (неименованное представление) – это подзапрос в конструкции `FROM` другого оператора. Для более старых версий MySQL большинство подзапросов могут быть переписаны в виде объединений или с использованием других методов.

### 1.8.5.2. Оператор `SELECT INTO TABLE`

В сервере MySQL не реализована поддержка следующего расширения SQL от Sybase: `SELECT...INTO TABLE...`. Вместо этого MySQL поддерживает стандартный SQL-синтаксис `INSERT INTO...SELECT...`, который в основном делает то же самое.

```
INSERT INTO tbl_temp2 (fld_id)
  SELECT tbl_temp1.fld_order_id
  FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

В качестве альтернативы можно воспользоваться `SELECT INTO OUTFILE...` или `CREATE TABLE SELECT...`.

Начиная с версии 5.0, MySQL поддерживает `SELECT...INTO` с пользовательскими переменными.

### 1.8.5.3. Транзакции и атомарные операции

Сервер MySQL (старшие выпуски версий 3.23 и все версии, начиная с 4.0) поддерживает транзакции в механизмах хранения InnoDB и BDB. InnoDB обеспечивает *полную* совместимость с ACID.

Остальные нетранзакционные механизмы хранения MySQL (такие, как MyISAM) следуют различным парадигмам обеспечения целостности данных, которые называются “атомарными операциями”. В терминологии транзакций таблицы MyISAM всегда работают в режиме AUTOCOMMIT=1. Атомарные операции часто предлагают сопоставимую целостность с более высокой производительностью.

Поскольку сервер MySQL поддерживает обе парадигмы, вы сами решаете, будут ли ваши приложения лучше работать со скоростью атомарных операций или с использованием средств управления транзакциями. Этот выбор осуществляется на уровне таблиц.

Как упоминалось ранее, различия в работе между транзакционными и нетранзакционными таблицами отражаются в основном на производительности. Транзакционные таблицы требуют значительно больших затрат памяти, дискового пространства и нагрузки на центральный процессор. С другой стороны, транзакционные таблицы, подобные InnoDB, также предлагают много существенных дополнительных возможностей. Модульная архитектура сервера MySQL допускает одновременное использование разных механизмов хранения для удовлетворения различным требованиям и достижения оптимальной производительности во всех ситуациях.

Но как использовать средства сервера MySQL для поддержки строгих требований целостности данных даже на нетранзакционных таблицах MyISAM, и как эти средства сравнить с работой с транзакционными таблицами?

1. Если ваше приложение написано таким образом, что оно зависит от возможности вызывать ROLLBACK вместо COMMIT в критических ситуациях, транзакции более удобны. Транзакции также гарантируют, что незавершенные обновления или результаты сбоев не будут записаны в базу данных. Сервер имеет возможность выполнить автоматический откат и сохранить базу данных. Если же вы применяете нетранзакционные таблицы, сервер MySQL почти во всех случаях предоставляет вам возможность разрешить потенциальные проблемы, включив простые проверки перед обновлением или, запуская простые сценарии, которые проверяют базу данных на непротиворечивость и автоматически вносят исправления либо выдают предупреждения, если обнаружены какие-то противоречия. Стоит отметить, что даже просто включая протоколирование работы MySQL или добавляя дополнительный протокол, вы можете нормально исправить таблицы без потери целостности.
2. В большинстве случаев критические транзакционные обновления могут быть переписаны как атомарные операции. Вообще говоря, все проблемы целостности, которые решают транзакции, могут быть предотвращены блокировкой таблиц LOCK TABLE или атомарными обновлениями, гарантирующими, что вы никогда не будете автоматически прерваны сервером, что является общей проблемой транзакционных систем управления базами данных.
3. Даже транзакционные системы могут терять данные, если сервер отключается. Разница между системами состоит только в том, насколько мал промежуток времени, в течение которого возможна потеря данных. Нет систем, безопасных на 100%, а есть только “достаточно безопасные”. Даже СУБД Oracle, имеющая репу-

тацию наиболее безопасной из транзакционных систем, периодически сообщает об утере данных в ситуациях подобного рода.

Для безопасной работы с сервером MySQL, независимо от того используются или нет транзакционные таблицы, нужно иметь резервные копии и держать включенным бинарное протоколирование. В этом случае вы сможете восстановить данные после любой ситуации, в которую можно попасть, имея дело с другими системами. Вообще говоря, располагать актуальными резервными копиями полезно при работе с любой СУБД.

Транзакционная парадигма обладает своими преимуществами и недостатками. Многие пользователи и разработчики приложений зависят от того, насколько просто можно написать код, моделирующий систему, для которой прерывание работы возможно или необходимо. Однако, даже если вы новичок в парадигме атомарных операций, либо лучше знакомы с транзакционной моделью, согласитесь, что выигрыш в производительности в 3–5 раз, который дает применение нетранзакционных таблиц по сравнению с наиболее быстрыми и оптимизированными транзакционными, весьма существен.

В ситуациях, когда целостность данных чрезвычайно важна, сервер MySQL демонстрирует надежность уровня транзакционных систем даже при работе с нетранзакционными таблицами. Если выполняется блокировка таблицы командой `LOCK TABLE`, все обновления приостанавливаются до тех пор, пока не выполнятся все проверки целостности. Если применяется блокировка `READ LOCAL` (в отличие от блокировки записи) для таблицы, допускающей параллельную вставку в конец, чтение разрешено, равно как и вставка другими клиентами. Вновь добавленные записи не будут видимы клиентом, установившим блокировку чтения, до тех пор, пока он не снимет блокировку. Применяя `INSERT DELAYED`, вы можете вставлять записи в локальную очередь до тех пор, пока не будет снята блокировка, не заставляя клиента ожидать завершения операции вставки.

Слово “атомарный” в том смысле, в каком мы его понимаем, не несет в себе ничего сверхъестественного. Оно означает только то, что вы можете быть уверены, что пока специфическое обновление идет, никакой другой пользователь не может взаимодействовать с ним, и поэтому не будет никакого автоматического отката (что может случиться с транзакционными таблицами, если вы не проявите достаточную осторожность). Сервер MySQL также гарантирует, что не будет никаких недействительных результатов чтения (dirty read).

Ниже перечислены некоторые приемы работы с нетранзакционными таблицами.

- Циклы, которые нуждаются в транзакциях, обычно могут быть закодированы с помощью `LOCK TABLES`; необходимости иметь дело с курсорами для обновления записей на лету нет.
- Для того чтобы избежать использования `ROLLBACK`, можно прибегнуть к следующей стратегии:
  1. Используйте `LOCK TABLES` для блокировки всех таблиц, к которым нужен доступ.
  2. Проверяйте все условия, которые должны быть истинными до начала обновлений.
  3. Выполняйте обновления только если все в порядке.
  4. Используйте `UNLOCK TABLES` для разблокирования таблиц.

Обычно это значительно более быстрый метод, чем применение транзакций с возможными откатами, хотя и не всегда. Единственная ситуация, когда это реше-

ние не удачно, это если кто-то прервет поток приложения во время обновления. В таком случае все блокировки будут сняты, но часть обновлений останется невыполненной.

- Можно также использовать функции для обновления записей за одну операцию. Можно получить весьма эффективные приложения, применяя следующую технику:
  - Модифицировать столбцы в соответствии с их текущими значениями.
  - Обновлять только те столбцы, которые изменились.

Например, когда выполняется обновление информации о заказчиках, мы обновляем только те данные, что изменились, либо данные, зависящие от измененных данных, сравнив их новые значения с исходными. Проверка измененных данных делается в конструкции WHERE оператора UPDATE. Если в результате запись не изменилась, потребуется выдать клиенту сообщение наподобие: "Некоторые из изменяемых вами данных изменены другим пользователем". Затем следует отобразить старую и новую версии записи о клиенте, чтобы пользователь решил, какую из них принять.

Это обеспечивает механизм, подобный блокировке столбца, но на самом деле даже лучше, потому что мы обновляем только некоторые из столбцов, используя новые значения, зависящие от их текущих значений. Это означает, что типовые операторы UPDATE должны выглядеть, как показано ниже:

```
UPDATE tablename SET pay_back=pay_back+125;
UPDATE customer
SET
    customer_date='current_date',
    address='new address',
    phone='new phone',
    money_owed_to_us=money_owed_to_us-125
WHERE
    customer_id=id AND address='old address' AND phone='old phone';
```

Такой подход весьма эффективен и работает, даже если другой клиент изменяет значения столбцов pay\_back и money\_owed\_to\_us.

- Во многих случаях пользователи хотят применять LOCK TABLES и/или ROLLBACK для целей управления уникальными идентификаторами. Это также можно обработать гораздо более эффективно без блокировок и откатов, используя столбцы AUTO\_INCREMENT и либо SQL-функцию LAST\_INSERT\_ID(), либо функцию C API с именем mysql\_insert\_id().

Обычно можно написать код, обслуживающий ситуации, когда требуется блокировка уровня строки. В некоторых ситуациях это действительно необходимо, и таблицы InnoDB поддерживают такие блокировки. Для таблиц MyISAM можно воспользоваться столбцами-флагами и поступить примерно так:

```
UPDATE имя_таблицы SET row_flag = 1 WHERE id = ID;
```

MySQL вернет 1 в качестве количества обновленных записей, если строка найдена и row\_flag не был равен 1 в исходной строке.

Можете думать об этом, как если бы сервер MySQL изменил предыдущий запрос на такой:

```
UPDATE имя_таблицы SET row_flag = 1 WHERE id = ID AND row_flag <> 1;
```

### 1.8.5.4. Хранимые процедуры и триггеры

Хранимые процедуры реализованы в MySQL 5.0.

Триггеры запланированы к реализации в версии 5.1. Триггер – это разновидность хранимой процедуры, которая вызывается при наступлении какого-то события. Например, можно написать процедуру, которая обрабатывает каждый раз, когда запись удаляется из транзакционной таблицы, и эта процедура автоматически удаляет соответствующего заказчика из таблицы заказчиков, когда все его финансовые транзакции удалены.

### 1.8.5.5. Внешние ключи

В сервере MySQL 3.23.44 и последующих версий механизм хранения InnoDB поддерживает проверку ограничений на внешние ключи, включая CASCADING, ON DELETE и ON UPDATE.

Для других механизмов хранения MySQL анализирует синтаксис FOREIGN KEY в операторе CREATE TABLE, но не использует и не хранит его. В будущих реализациях планируется сохранять эту информацию в файле спецификаций таблиц, чтобы она могла быть извлечена с помощью mysqldump и через ODBC. На более поздней стадии ограничения внешних ключей будут реализованы и для таблиц MyISAM.

Применение внешних ключей даст разработчикам баз данных некоторые преимущества:

- Если предположить, что отношения между таблицами спроектированы правильно, ограничения внешних ключей значительно затрудняют программистам возможность внести в базу данных какую-либо противоречивую информацию.
- Централизованная проверка ограничений со стороны сервера делает излишней эту проверку со стороны приложения. Это исключает вероятность того, что некоторые приложения могут ее выполнять, а некоторые – нет.
- Применением каскадных обновлений и удалений может существенно упростить код приложений.
- Правильно спроектированные внешние ключи упрощают документирование отношений между таблицами.

Однако следует помнить, что эти выгоды достигаются за счет большей нагрузки на сервер баз данных, которому приходится выполнять все проверки. Это приводит к некоторому снижению производительности, что для ряда приложений может оказаться настолько нежелательным, что лучше обойтись без этого вообще. (Некоторые важные коммерческие программы по этой причине имеют встроенную проверку логики внешних ключей на уровне приложения.)

MySQL дает возможность разработчикам выбирать требуемый подход. Если вам не нужны внешние ключи, и вы хотите избежать лишней нагрузки, связанной с проверками ссылочной целостности, вы можете выбрать другой тип таблиц, такой как MyISAM. Например, механизм хранения MyISAM обеспечивает очень высокую производительность для приложений, которые выполняют только операции INSERT и SELECT, поскольку вставки могут выполняться одновременно с выборками.

Если вы предпочитаете обойтись без преимуществ проверки ссылочной целостности, вам следует иметь в виду следующее:

- При отсутствии проверки отношений внешних ключей со стороны сервера этим должно заниматься само приложение. Например, придется заботиться о вставке записей в таблицы в правильном порядке, избегая появления висячих дочерних

записей. Приложение также должно быть готовым к восстановлению после ошибки, связанных с прерыванием операций массовой вставки записей.

- Если приложению требуется только ссылочная целостность типа ON DELETE, следует отметить, что в MySQL 4.0 имеется возможность выполнять многотабличные операции DELETE для удаления записей из многих таблиц одним оператором.
- Обходной путь, позволяющий компенсировать отсутствие ON DELETE, заключается в том, чтобы добавить соответствующие дополнительные операторы DELETE в приложение для удаления записей из таблиц, имеющих внешние ключи. На практике часто это работает так же быстро, как и автоматические внешние ключи, но при этом значительно более переносимо.

Не забывайте, что применение внешних ключей иногда порождает проблемы:

- Поддержка внешних ключей применима ко многим случаям, когда нужно обеспечить ссылочную целостность, но это не отменяет необходимости очень тщательного проектирования отношения ключей, чтобы избежать циклических зависимостей или некорректных комбинаций каскадного удаления.
- Не столь уж необычна ситуация, когда администратор баз данных создает такую топологию отношений между таблицами, которая затрудняет восстановление отдельных таблиц из резервной копии. (MySQL смягчает сложность ситуаций подобного рода, позволяя временно отключать проверки внешних ключей на время загрузки данных в таблицы, зависящие от других таблиц. В MySQL 4.1.1 утилита `mysqldump` генерирует файлы дампа, которые это делают автоматически при загрузке.)

Следует помнить, что внешние ключи в SQL применяются для проверки и поддержания ссылочной целостности, но не для объединения таблиц. Если вам нужен результат запроса к множеству таблиц от одного оператора SELECT, вы делаете это за счет описания объединения между ними:

```
SELECT * FROM t1, t2 WHERE t1.id = t2.id;
```

Синтаксис FOREIGN KEY без ON DELETE... часто применяется в ODBC-приложениях для автоматической генерации конструкций WHERE.

### 1.8.5.6. Представления

Представления в настоящее время реализованы и появятся в версии 5.0 сервера MySQL. Неименованные представления (*вторичные таблицы*), подзапросы в конструкции FROM оператора SELECT уже реализованы в версии 4.1.

Исторически сложилось так, что сервер MySQL больше всего использовался в приложениях и Web-системах, где разработчик имел полный доступ к используемой базе данных. Однако ситуация постепенно менялась, и теперь мы обнаружили, что все возрастающее число пользователей считают представления очень важным и полезным средством.

Представления применяются для того, чтобы предоставить пользователям доступ к группе таблиц таким образом, как будто это одна таблица и тем самым ограничить доступ к ним. Представления также можно использовать для ограничения доступа к строкам таблицы (выделяя, таким образом, подмножество записей). Для ограничения доступа к столбцам представления не нужны, поскольку сервер MySQL обладает развитой системой привилегий.

Многие СУБД не разрешают выполнять операции обновления представлений. Вместо этого необходимо обновлять отдельные таблицы. При разработке механизма поддержки представлений в MySQL нашей целью было, оставаясь, насколько возможно, в рамках стандарта SQL, достичь полной совместимости с шестым правилом Кодда для реляционных баз данных. Все представления, теоретически обновляемые, должны быть обновляемы на практике.

#### 1.8.5.7. '--' как начало комментария

В ряде других СУБД '--' используется для обозначения начала комментария. Сервер MySQL в качестве начального символа комментария использует '#'. Также поддерживаются комментарии в стиле языка C: /\* пример комментария \*/.

Сервер MySQL 3.23.3 и более поздние версии поддерживают комментарии, начинающиеся с '--', предполагая, что за комментарием следует пробел (или управляющий символ вроде перевода строки). Требование, касающееся пробелов, предотвращает проблемы с автоматически сгенерированными SQL-запросами, которые используют что-нибудь наподобие показанного ниже кода, где вместо !payment! автоматически подставляется значение оплаты:

```
UPDATE account SET credit=credit-!payment!
```

Подумайте, что случится, если значение оплаты будет отрицательным, например, -1:

```
UPDATE account SET credit=credit--1
```

credit--1 — вполне корректное выражение на языке SQL, но если -- интерпретируется как начало комментария, то часть выражения после этого фрагмента теряется. В результате получаем оператор, имеющий совершенно другой смысл, чем ожидалось:

```
UPDATE account SET credit=credit
```

Он не выполняет никакого обновления вообще! Это иллюстрирует то обстоятельство, что разрешение использовать в качестве начала комментария '--' чревато довольно-таки неприятными последствиями.

Используя реализацию метода задания комментариев, поддерживаемую в MySQL 3.23.3 и последующих версиях, получаем безопасное выражение credit--1.

Другое безопасная возможность состоит в том, что интерпретатор командной строки mysql удаляет строки, начинающиеся с '--'.

Приведенная далее информация справедлива, только если вы работаете с MySQL версий, предшествующих 3.23.3.

Если у вас есть SQL-программа в текстовом файле, которая включает комментарии, начинающиеся с '--', вам придется с помощью утилиты replace заменить их на комментарии с символом '#':

```
shell> replace " --" " #" < text-file-with-funny-comments.sql \  
| mysql имя_базы_данных
```

вместо обычного:

```
shell> mysql имя_базы_данных < text-file-with-funny-comments.sql
```

Можно также отредактировать командный файл на месте, заменив '--' на '#':

```
shell> replace " --" " #" -- text-file-with-funny-comments.sql
```

Вернуться к исходному варианту можно следующим образом:

```
shell> replace " #" " --" -- text-file-with-funny-comments.sql
```

## 1.8.6. Как MySQL работает с ограничениями

MySQL позволяет работать как с транзакционными таблицами, для которых возможен откат, так и с нетранзакционными, для которых откат не предусмотрен, поэтому работа с ограничениями (constraints) в MySQL несколько отличается от того, как это делается в других СУБД.

Нам приходится иметь дело с ситуацией, когда нужно обновить множество строк в нетранзакционной таблице, причем нет возможности выполнить откат при возникновении ошибки.

Базовая философия состоит в том, чтобы попытаться сгенерировать ошибку для всего, что можно обнаружить во время компиляции, но попытаться восстановиться после любой ошибки, возникшей во время выполнения. Мы достигаем этого в большинстве случаев, но пока еще не во всех (см. раздел 1.6.4).

Выбор, который стоит перед MySQL сводится к тому, что в случае возникновения ошибки необходимо остановить выполнение оператора на полпути или восстановить данные настолько хорошо, насколько возможно, и продолжать работу.

В последующих разделах описано, что происходит в этих случаях с различными типами ограничений.

### 1.8.6.1. Ограничение PRIMARY KEY/UNIQUE

Нормально, если генерируется ошибка при попытке вставить или обновить запись, которая приводит к нарушению ограничений первичного ключа, внешнего ключа или уникального ключа. Если используется транзакционный механизм хранения, подобный InnoDB, MySQL автоматически выполнит откат транзакции. Если же применяется нетранзакционный механизм хранения, MySQL остановится на некорректной записи и оставит все оставшиеся записи неизменными.

Чтобы облегчить жизнь, MySQL поддерживает ключевое слово IGNORE для большинства команд, которые могут привести к нарушению ключей (например, INSERT IGNORE или UPDATE IGNORE). В этих случаях MySQL просто игнорирует любые нарушения ограничений ключей и продолжает обрабатывать остальные строки. Информация о том, что предпринял MySQL, доступна через функцию `mysql_info()` API-интерфейса C. В MySQL 4.1 и последующих версиях можно также воспользоваться командой `SHOW WARNINGS`.

Стоит напомнить, что на данный момент внешние ключи поддерживают только таблицы InnoDB. Реализация внешних ключей для таблиц MyISAM запланирована в версии MySQL 5.1.

### 1.8.6.2. Ограничения NOT NULL и значения DEFAULT

Чтобы обеспечить простое управление нетранзакционными таблицами, все столбцы таблиц в MySQL имеют значения по умолчанию.

Если вы вставляете “неправильное” значение в столбец, такое как NULL в столбец, объявленный как NOT NULL, или же слишком большое значение в числовой столбец, MySQL установит его значение в “лучшее из возможных” вместо того, чтобы выдавать ошибку:

- Если вы попытаетесь сохранить в числовом столбце значение, выходящее за пределы допустимого диапазона, сервер MySQL вставит вместо него ноль, минимально возможное либо максимально возможное значение для этого столбца.



- В строковом столбце сервер MySQL сохранит либо пустую строку, либо самую длинную, которая может поместиться в данный столбец.
- Если вы попытаетесь вставить строку, начинающуюся не с цифры, в числовой столбец, MySQL запишет туда 0.
- Если попытаться вставить значение NULL в столбец, не допускающий значений NULL, MySQL вставит вместо этого 0 в числовой столбец и пустую строку – в символьный. (Однако такое поведение при вставке одиночных записей может быть изменено, если скомпилировать сервер MySQL с опцией компиляции `-DONT_USE_DEFAULT_FIELDS.`) Это заставит операторы INSERT генерировать ошибки, если только вы явно не зададите значения для всех столбцов, которые требуют значений NOT NULL.
- MySQL позволит сохранить некоторые неправильные значения даты в столбцы типа DATE и DATETIME (например '2000-02-31' или '2000-02-00'). Идея состоит в том, что проверка корректности дат не является обязанностью сервера SQL. Если MySQL может сохранять значение даты и извлекать точно такое значение, он сохраняет его таким, как получил. Если дата абсолютно неправильная (за пределами возможностей сервера сохранить ее), то вместо нее сохраняется специальное значение '0000-00-00'.

Причина существования изложенных правил заключается в том, что мы не можем проверить эти условия до тех пор, пока не начнется выполнение запроса. Мы не можем выполнить откат изменений, если сталкиваемся с проблемой после того, как несколько строк уже обновлены, поскольку данный тип таблиц может не поддерживать транзакции. Выбор в пользу прерывания выполнения оператора представляется неудачным – в этом случае получается, что обновление выполнено наполовину, что приводит к худшему сценарию. В этом случае предпочтительнее сделать “лучшее из возможного” и затем продолжить, как будто ничего не произошло.

Это означает, что не стоит полагаться на MySQL в смысле проверки корректности данных столбцов. Вместо этого приложение само должно заботиться о том, чтобы отправлять серверу MySQL только правильные данные.

В MySQL 5.0 мы планируем провести улучшения, выдавая предупреждения, когда происходит автоматическое преобразование столбцов, плюс опцию, позволяющую откатить оператор, который пытается выполнить запрещенное присваивание данных до тех пор, пока используются только транзакционные таблицы.

### 1.8.6.3. Ограничения ENUM и SET

В MySQL 4.x ENUM не является настоящим ограничением, а просто наиболее эффективным способом определения столбцов, которые могут содержать значения только из заданного набора возможных значений.

Если вы вставляете некорректное значение в столбец типа ENUM, оно устанавливается в зарезервированное перечислимое значение 0, которое отображается как пустая строка в строчном контексте.

При попытке вставки некорректного значения в столбец типа SET это значение игнорируется. Например, если столбец может содержать значения 'a', 'b' и 'c', попытка сохранить в столбце значения 'a,x,b,y' приведет к сохранению 'a,b'.

## 1.8.7. Известные ошибки и недостатки дизайна MySQL

### 1.8.7.1. Ошибки в версии 3.23, исправленные в более поздних версиях MySQL

Перечисленные ниже известные ошибки не были исправлены в версии 3.23, поскольку их исправление привело бы к изменению большей части кода, что могло, в свою очередь, вызвать появление других, даже намного худших ошибок. Эти ошибки классифицируются как “не критические” или “терпимые”.

- Можно возникнуть взаимная блокировка, если использовать `LOCK TABLE` для блокировки нескольких таблиц и затем в том же соединении удалить одну из них оператором `DROP TABLE` в то время, когда другой поток пытается ее заблокировать. (Для устранения взаимной блокировки можно с помощью команды `KILL` завершить любой из участвующих в ней потоков.) Эта проблема была решена в MySQL 4.0.12.
- `SELECT MAX(ключевой_столбец) FROM t1,t2,t3...`, где одна из таблиц пуста, не возвращает `NULL`, а возвращает максимальное значение столбца. Это исправлено в MySQL 4.0.11.
- `DELETE FROM таблица_heap` без конструкции `WHERE` не работает на заблокированной `HEAP`-таблице.

### 1.8.7.2. Ошибки в версии 4.0, исправленные в более поздних версиях

Перечисленные ниже известные ошибки не были исправлены в версии 4.0, поскольку их исправление привело бы к изменению большей части кода, что могло, в свою очередь, вызвать появление других, даже намного худших ошибок. Эти ошибки также классифицируются, как “не критические” или “терпимые”.

- В `UNION` первый же оператор `SELECT` определяет свойства результирующих столбцов – тип, максимальную длину (`max_length`), возможность записи значений `NULL`. Это было исправлено в MySQL 4.1.1 – значения свойств столбцов определяются на основании строк из всех частей `UNION`.
- В операторе `DELETE` по нескольким таблицам невозможно обращаться к таблицам по псевдонимам. Исправлено в MySQL 4.1.

### 1.8.7.3. Открытые ошибки и недостатки дизайна MySQL

Следующие проблемы уже известны и решение их имеет наивысший приоритет:

- Удаление ограничений `FOREIGN KEY` не работает на реплицированной копии, поскольку ограничение может иметь другое имя, чем в исходной базе.
- `REPLACE` (и `LOAD DATA` с опцией `REPLACE`) не вызывает `ON DELETE CASCADE` (каскадного удаления).
- Невозможно смешивать `UNION ALL` и `UNION DISTINCT` в одном и том же запросе. Если используется `ALL` для одного из `UNION`, это касается их всех.
- Если один пользователь запустил долго выполняемую транзакцию, а другой удалил таблицу, которая обновлялась в этой транзакции, то существует небольшая вероятность того, что в бинарный протокол попадет команда `DROP TABLE`, прежде чем таблица будет задействована в транзакции. Мы планируем исправить это в

версии 5.0, заставив оператор `DROP TABLE` ожидать до тех пор, пока таблица используется хотя бы в одной транзакции.

- При вставке больших целочисленных значений (между  $2^{63}$  и  $2^{64} - 1$ ) в десятичный или строковый столбец, оно вставляется как отрицательное значение, поскольку число оценивается в контексте целого со знаком. Мы планируем исправить это в MySQL 4.1.
- `FLUSH TABLES WITH READ LOCK` не блокирует `CREATE TABLE` или `COMMIT`, что может привести к проблемам с позицией в бинарном протоколе при выполнении полного резервного копирования таблиц и бинарного протокола.
- `ANALYZE TABLE` для таблиц BDB в некоторых случаях может приводить к тому, что таблицы становятся недоступными до тех пор, пока не будет перезапущен `mysqld`. Если это случается, в файл ошибок MySQL добавляется такая строка:  
`001207 22:07:56 bdb: log_flush: LSN past current end-of-log`
- MySQL принимает скобки в конструкции `FROM` оператора `SELECT`, но молча игнорирует их. Причина того, что сообщения об ошибках не выдаются, состоит в том, что очень многие клиенты при автоматической генерации запросов добавляют скобки в конструкцию `FROM`, даже если в этом нет необходимости.
- Объединение многих `RIGHT JOIN` или комбинаций `LEFT JOIN` и `RIGHT JOIN` в одном запросе могут не приводить к корректному результату, потому что MySQL генерирует строки `NULL` для таблицы, следующей за `LEFT JOIN` или перед `RIGHT JOIN`. Это будет исправлено в 5.0, одновременно мы добавим поддержку скобок в предложение `FROM`.
- Не выполняйте `ALTER TABLE` для таблицы BDB, участвующей в многотабличных транзакциях, до тех пор, пока все они не завершатся. (Транзакции могут быть проигнорированы.)
- Команды `ANALYZE TABLE`, `OPTIMIZE TABLE` и `REPAIR TABLE` могут вызывать проблемы в таблицах, в которых используется `INSERT DELAYED`.
- Выполнение `LOCK TABLE...` и `FLUSH TABLES...` не гарантирует, что в таблице не окажется наполовину завершенных транзакций.
- Таблицы BDB открываются несколько медленно. Если в базе данных есть много таблиц BDB, потребуется немало времени, чтобы запустить клиент `mysql` с опцией `-A` или выполнить `rehash`. В особенности это касается тех случаев, когда имеется большой табличный кэш.
- Репликация использует протоколирование уровня запросов. Реплицируемая база фиксирует выполняемые запросы в бинарном протоколе. Это очень быстрый, компактный и эффективный метод протоколирования, который в большинстве случаев работает превосходно. Однако, хотя нам и не известны такие случаи, существует теоретическая возможность того, что данные в исходной и целевой базе окажутся разными, если запрос составлен таким образом, что модификация данных будет недетерминированной. Это остается на совести оптимизатора запросов. (Вообще говоря, это очень плохая практика, даже за рамками темы репликации!). Например:
  - Операторы `CREATE...SELECT` или `INSERT...SELECT`, которые вставляют нулевые или `NULL`-значения в столбцы `AUTO_INCREMENT`.

- DELETE, если выполняется для таблицы, имеющей внешние ключи со свойством ON DELETE CASCADE.
- REPLACE...SELECT, INSERT IGNORE...SELECT, если во вставляемых данных присутствуют дублированные ключи.

Если и только если все эти запросы не содержат конструкцию ORDER BY, гарантируется детерминированный порядок.

Например, для INSERT...SELECT без ORDER BY, оператор SELECT может вернуть записи в другом порядке (что может оказаться результатом того, что записи имеют разный ранг и, следовательно, разные номера в столбцах AUTO\_INCREMENT), в зависимости от того, как сработают оптимизаторы в исходной и целевой базах при репликации. Запросы могут быть оптимизированы различным образом в двух базах, если:

- Файлы, используемые в запросах, не одинаковы. Например, OPTIMIZE TABLE был запущен для исходных таблиц и не запускался для целевых (для исправления этого, начиная с MySQL 4.1.1, OPTIMIZE TABLE, ANALYZE TABLE и REPAIR TABLE записываются в бинарный протокол).
- Исходная и целевая таблицы обрабатываются разными механизмами хранения (это возможно; например, вы можете использовать InnoDB в исходной реплицируемой базе и MyISAM – в целевой базе, если в ней нет дискового пространства достаточного объема).
- Размер буферов разный (key\_buffer\_size и так далее).
- Исходная и целевая базы работают под управлением разных версий MySQL, и коды оптимизаторов у них отличаются.

Эта проблема может также затронуть восстановление базы с использованием mysqlbinlog|mysql.

Наилучший способ избежать этих проблем во всех случаях – добавлять конструкцию ORDER BY к подобным недетерминированным запросам, дабы гарантировать, что строки сохраняются и модифицируются в одном и том же порядке. В будущих версиях MySQL мы будем автоматически добавлять ORDER BY туда, где это необходимо.

Следующие проблемы известны и будут решены в надлежащее время:

- Имена файлов протоколов основаны на имени хоста (если только явно не указаны при запуске). На сегодняшний день, если изменяется имя хоста, нужно использовать опции наподобие --log-bin=старое\_имя\_хоста-bin. Другой способ состоит в том, чтобы просто переименовать старые файлы в соответствии с изменением имени хоста.
- mysqlbinlog не удаляет временные файлы, которые остаются после выполнения команды LOAD DATA INFILE.
- RENAME не работает на временных таблицах и таблицах, используемых в таблицах MERGE.
- При использовании функции RPAD() в запросах, выполнение которых потребует создания временных таблиц, во всех результирующих строках заключительные пробелы будут удалены. Ниже показан пример такого запроса.

```
SELECT RPAD(t1.column1, 50, ' ') AS f2, RPAD(t2.column2, 50, ' ') AS f1
FROM table1 as t1 LEFT JOIN table2 AS t2 ON t1.record=t2.joinID
ORDER BY t2.record;
```

Вследствие этой ошибки вы не получите завершающих пробелов в результирующих значениях. Эта проблема также относится ко всем остальным строковым функциям, добавляющим пробелы справа.

Причиной является то, что HEAP-таблицы, которые используются вначале для временных таблиц, не могут работать со столбцами типа VARCHAR.

Подобное поведение присуще всем версиям MySQL и будет исправлено в одном из выпусков в рамках серии 4.1.

- Из-за способа хранения файла определений таблиц невозможно использовать символ 255 (CHAR(255)) в именах таблиц, столбцов или перечислений. Исправление запланировано в версии 5.1, когда мы будем иметь новый формат файлов определения таблиц.
- При использовании SET CHARACTER SET невозможно использовать переведенные символы в именах баз данных, таблиц и столбцов.
- Невозможно использовать шаблонные символы '\_', '%' вместе с ESCAPE в LIKE... ESCAPE.
- Если есть столбец типа DECIMAL, в котором одно и то же число сохраняется в разных форматах (например, +01.00, 1.00, 01.00), то GROUP BY может рассматривать такие значения как различные.
- Невозможно собрать сервер в другом каталоге при использовании потоков MIT-pthreads. Поскольку для этого нужно изменять потоки MIT-pthreads, по всей видимости, исправлять мы это не будем.
- Значения типа BLOB не могут "надежно" применяться в GROUP BY, ORDER BY или DISTINCT. В этих случаях для сравнения значений BLOB используются только первые max\_sort\_length байт. Значение по умолчанию max\_sort\_length равно 1024 байта. Это может быть изменено во время запуска сервера. Чтобы обойти это ограничение, в большинстве случаев можно применить подстроки, например:  

```
SELECT DISTINCT LEFT(столбец_blob, 2048) FROM имя_таблицы
```
- Вычислительные операции выполняются над типами BIGINT и DOUBLE (оба они обычно имеют длину 64 байта). Какую точность вы получите, зависит от функции. Общее правило звучит так: поразрядные функции выполняются с точностью BIGINT, IF и ELT() – с точностью BIGINT или DOUBLE, все остальные – с точностью DOUBLE. Старайтесь избегать применения беззнаковых длинных значений, если они могут оказаться длиной более 63 бит (9223372036854775807) во всех случаях, кроме битовых полей. Версия MySQL 4.0 лучше справляется с BIGINT, чем версия MySQL 3.23.
- У всех строковых столбцов, кроме BLOB и TEXT, при извлечении из базы автоматически удаляются завершающие пробелы. Для типа CHAR это нормально. Ошибка состоит в том, что столбцы VARCHAR обрабатываются точно так же.
- В одной таблице можно иметь не более 255 столбцов типа ENUM и SET.

- В `MIN()`, `MAX()` и других агрегатных функциях в настоящее время столбцы типов `ENUM` и `SET` сравниваются по их строковым значениям вместо того, чтобы делать это по относительному положению строки в наборе.
- `mysqld_safe` перенаправляет все сообщения от `mysqld` в журнал `mysqld`. Одна связанная с этим проблема состоит в том, что если запустить `mysqladmin refresh` для закрытия и повторного открытия журнала, стандартный поток вывода `stdout` и стандартный поток ошибок `stderr` остаются перенаправленными в старый журнал. Если `--log` применяется интенсивно, нужно отредактировать `mysqld_safe`, чтобы он выводил протокол в `имя_хоста.err` вместо `имя_хоста.log`. В этом случае можно легко использовать дисковое пространство, занятое старым журналом, удалив его и запустив `mysqladmin refresh`.
- Оператор `UPDATE` обновляет столбцы слева направо. Если вы ссылаетесь на обновляемые столбцы, то получаете их новые значения вместо старых. Например, следующий оператор увеличит значение `KEY` на 2, а не на 1:

```
mysql> UPDATE имя_таблицы SET KEY=KEY+1,KEY=KEY+1;
```

- Можно обращаться к множеству временных таблиц в одном запросе, но нельзя обращаться к одной и той же временной таблице более одного раза. Например, приведенный ниже оператор не работает:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
ERROR 1137: Can't reopen table: 'temp_table'
(ERROR 1137: Невозможно повторно открыть таблицу: 'temp_table')
```

- Оптимизатор может обрабатывать `DISTINCT` по-разному, когда используются “скрытые” столбцы в объединении и когда они не используются. Скрытые столбцы в объединении считаются частью результата (даже если они не показываются), тогда как в нормальных запросах скрытые столбцы не участвуют в сравнениях для `DISTINCT`. Возможно, мы исправим это в будущем, чтобы скрытые столбцы никогда не участвовали в вычислении `DISTINCT`.

Ниже представлены соответствующие примеры:

```
SELECT DISTINCT mp3id FROM band_downloads
WHERE userid = 9 ORDER BY id DESC;
```

и

```
SELECT DISTINCT band_downloads.mp3id
FROM band_downloads,band_mp3
WHERE band_downloads.userid = 9
AND band_mp3.id = band_downloads.mp3id
ORDER BY band_downloads.id DESC;
```

Во втором случае сервер MySQL 3.23.x может выдать две идентичные строки в результирующем наборе (поскольку значения скрытого столбца `id` у них различны). Отметим, что это случается с запросами, где в результате не выводятся столбцы, участвующие в конструкции `ORDER BY`.

- Поскольку MySQL позволяет работать с типами таблиц, для которых не поддерживаются транзакции, и поэтому невозможен откат, некоторые вещи работают здесь несколько иначе, чем в других серверах SQL. Это связано только с тем, чтобы гарантировать, что откат никогда не понадобится MySQL для выполнения

SQL-операторов. Иногда это может оказаться неудобным, потому что значения столбцов должны проверяться самим приложением, но это дает существенный выигрыш в скорости, так как позволяет MySQL выполнить оптимизацию, которую в противном случае было бы очень трудно реализовать.

Если вы устанавливаете некорректное значение столбца, MySQL вместо отката пытается сохранить вместо него “наилучшее возможное” значение. Информацию о том, как это происходит, можно найти в разделе 1.8.6.

- Если запускается PROCEDURE на запросе, возвратившем пустой результирующий набор, в некоторых случаях PROCEDURE не трансформирует столбцы.
- При создании таблиц типа MERGE не выполняется проверка входящих в них таблиц на предмет соответствия типов.
- Если вы используете ALTER TABLE вначале для добавления уникального индекса к таблице, включенной в MERGE, а затем пытаетесь добавить нормальный индекс к MERGE-таблице, порядок ключей будет разным, если существовал старый ключ, который был не уникальным для таблицы. Это происходит ввиду того, что ALTER TABLE помещает уникальные индексы перед нормальными индексами, чтобы как можно раньше обнаружить дублированные ключи.

Далее перечислены известные ошибки в ранних версиях MySQL.

- В описанных ниже случаях может случиться аварийный отказ ядра.
  - Обработчик отложенных вставок задерживает вставки в таблицу.
  - LOCK TABLE с WRITE.
  - FLUSH TABLE.
- До версии сервера MySQL 3.23.2 оператор UPDATE, обновляющий значения ключей, упомянутых в конструкции WHERE, мог аварийно завершиться, поскольку ключи используются для поиска записей, и одни и те же строки могли быть найдены по несколько раз:

```
UPDATE имя_таблицы SET KEY=KEY+1 WHERE KEY > 100;
```

Обходной путь выглядит следующим образом:

```
UPDATE имя_таблицы SET KEY=KEY+1 WHERE KEY+0 > 100;
```

Это работает потому, что сервер MySQL не использует индексы для выражений в конструкции WHERE.

- До версии сервера MySQL 3.23 значения всех числовых типов трактовались как числа с фиксированной запятой. Это означало, что нужно было явно указывать, сколько десятичных знаков должно иметь поле с плавающей запятой. Все результаты возвращались с корректным количеством знаков.

# Структура языка

**Н**астоящая глава посвящена правилам написания следующих элементов SQL-операторов при использовании MySQL:

- Литеральных значений, таких как строки и числа.
- Идентификаторов, таких как имена таблиц и столбцов.
- Пользовательских и системных переменных.
- Комментариев.
- Резервированных слов.

## 2.1. Литеральные значения

Этот раздел объясняет, как писать литеральные значения в MySQL. Сюда входят строки, числа, шестнадцатеричные значения, булевские значения и NULL. Раскрываются также некоторые нюансы и особенности, с которыми вы можете столкнуться, имея дело с упомянутыми базовыми типами MySQL.

### 2.1.1. Строки

Строка – это последовательность символов, окруженная либо одинарными (''), либо двойными (") кавычками, например:

```
'строка'
"еще строка"
```

Если SQL-режим сервера активизирует ANSI\_QUOTES, то строковые литералы могут быть окружены только одинарными кавычками. Строки с двойными кавычками будут интерпретироваться как идентификаторы.

Начиная с MySQL 4.1.1, строковые литералы могут иметь назначенный им необязательный набор символов и порядок сортировки:

```
[_имя_набора_символов]'строка' [COLLATE имя_сопоставления]
```

Примеры:

```
SELECT _latin1'строка';
SELECT _latin1'строка' COLLATE latin1_danish_ci;
```

Более подробную информацию о синтаксисе строк можно найти в разделе 3.3.7.



Внутри строки некоторые последовательности имеют специальное значение. Каждая из таких последовательностей начинается с обратной косой черты ('\'), называемой *символом отмены*. MySQL распознает следующие последовательности отмены:

- \0 Символ с ASCII-кодом 0 (NUL).
- \' Символ одинарной кавычки ('').
- \" Символ двойной кавычки (").
- \b Символ забоя.
- \n Символ перевода строки.
- \r Символ возврата каретки.
- \t Символ табуляции.
- \z Символ с ASCII-кодом 26 (<Control-Z>). Этот символ может быть закодирован как '\z', чтобы позволить обойти проблему, связанную с тем, что ASCII 26 предназначен для обозначения конца файла в Windows. (ASCII 26 может вызвать проблемы, если вы попытаетесь применить команду вроде `mysql имя_базы_данных < имя_файла.`)
- \\ Символ обратной косой черты ('\').
- \% Символ '%'. См. комментарии ниже.
- \\_ Символ '\_'. См. комментарии ниже.

Эти последовательности чувствительны к регистру. Например, '\b' интерпретируется как символ забоя, тогда как '\B' - как символ 'B'.

Последовательности '\%' и '\\_' используются при поиске литеральных включений экземпляров '%' и '\_' в контексте сравнения с шаблонами, где в противном случае они могут рассматриваться как шаблонные символы. См. раздел 5.3.1. Следует отметить, что если вы используете '\%' и '\\_' в других контекстах, то они возвращают строки '\%' и '\\_', а не '%' и '\_'.

Во всех последовательностях отмены обратная косая черта игнорируется. То есть защищенный им символ интерпретируется так, будто обратной косой черты нет.

Существует несколько способов включить знаки кавычек в строку:

- Одинарная кавычка (') внутри строки, ограниченной одинарными кавычками, может записываться, как ''.
- Двойная кавычка внутри строки, ограниченной двойными кавычками, может быть записана, как "".
- Одинарная кавычка внутри строки, ограниченной двойными кавычками, не нуждается ни в какой дополнительной обработке и не требует отмены и удвоения. То же самое касается двойной кавычки, которая встречается внутри строки, ограниченной одинарными кавычками.

Следующие операторы SELECT демонстрируют, как работает отмена и помещение в кавычки:

```
mysql> SELECT 'hello', '"hello"', '""hello""', 'hel''lo', '\\hello';
+-----+-----+-----+-----+-----+
| hello | "hello" | ""hello"" | hel'lo | '\\hello |
+-----+-----+-----+-----+-----+
```

```
mysql> SELECT "hello", "'hello'", '''hello''', "hel"lo", "\"hello";
+-----+-----+-----+-----+-----+
| hello | 'hello' | '''hello''' | hel"lo | "\"hello |
+-----+-----+-----+-----+

mysql> SELECT 'This\nIs\nFour\nLines';
+-----+
| This
Is
Four
Lines |
+-----+
```

Если вы хотите вставить двоичные данные в строковый столбец (такой как BLOB), следующие символы должны быть представлены последовательностями отмены:

- |      |  |
|------|--|
| NULL | Нулевой байт (ASCII 0). Представляется как '\0' (обратная косая черта с последующим символом ASCII '0'). |
| \    | Обратная косая черта (ASCII 92). Представляется как '\\'.  |
| '    | Одинарная кавычка (ASCII 39). Представляется, как '\ '.  |
| "    | Двойная кавычка (ASCII 34). Представляется, как '\"'.  |

При написании прикладных приложений к любой строке, которая может включать любой из этих специальных символов, должна быть правильно применена последовательность отмены, прежде чем строка будет использована в качестве значения данных в операторе SQL, который отправляется серверу MySQL. Это можно сделать двумя способами:

- Обработать строку функцией, которая отменяет специальные символы. Например, в API-интерфейсе C для этого можно использовать функцию `mysql_real_escape_string()`. Программный интерфейс Perl DBI предлагает метод `quote` для преобразования специальных символов в корректные последовательности отмены.
- В качестве альтернативы явной отмене специальных символов, многие программные интерфейсы MySQL предлагают возможность указания на месте за счет вставки специальных маркеров в строку запроса, с последующим связыванием данных с ними, когда запрос отправляется. В этом случае API-интерфейс сам займется об отмене специальных символов.

## 2.1.2. Числа

Целые числа представляются последовательностью цифр. Действительные (с плавающей точкой) используют точку '.' в качестве разделителя целой и дробной части. Любой из этих двух типов чисел может иметь предваряющий знак минус '-' для обозначения отрицательных значений.

Ниже приведены примеры допустимых целых чисел:

```
1221
0
-32
```

А вот примеры допустимых чисел с плавающей точкой:

```
294.42
-32032.6809e+10
148.00
```

Целое может использоваться в контексте числа с плавающей точкой, при этом оно интерпретируется как эквивалентное число с плавающей точкой.

### 2.1.3. Шестнадцатеричные значения

MySQL поддерживает шестнадцатеричные значения. В числовом контексте они выступают в качестве целых (с 64-битной точностью). В строковом контексте они являются бинарными строками, в которых каждая пара шестнадцатеричных цифр преобразуется в символ:

```
mysql> SELECT x'4D7953514C';
-> 'MySQL'
mysql> SELECT 0xa+0;
-> 10
mysql> SELECT 0x5061756c;
-> 'Paul'
```

В MySQL 4.1 (и в MySQL 4.0 при использовании опции `--new`), типом по умолчанию для шестнадцатеричного значения является строковый. Если вы хотите иметь гарантию, что значение будет восприниматься как число, можете использовать `CAST(...AS UNSIGNED)`:

```
mysql> SELECT 0x41, CAST(0x41 AS UNSIGNED);
-> 'A', 65
```

Синтаксис `0x` базируется на ODBC. В ODBC шестнадцатеричные строки часто используются для задания значений столбцов типа BLOB. Синтаксис `x'`шестнадцатеричная\_строка' является новым в версии 4.0 и основан на стандарте SQL.

Начиная с MySQL 4.0.1, вы можете преобразовать строку или число в строку в шестнадцатеричном формате с помощью функции `HEX()`:

```
mysql> SELECT HEX('cat');
-> '636174'
mysql> SELECT 0x636174;
-> 'cat'
```

### 2.1.4. Булевские значения

Начиная с версии MySQL 4.1, константа `TRUE` оценивается как 1, а константа `FALSE` — как 0. Имена констант могут записываться в любом регистре.

```
mysql> SELECT TRUE, true, FALSE, false;
-> 1, 1, 0, 0
```

### 2.1.5. Значение NULL

Значение `NULL` означает “отсутствие данных”. `NULL` можно записывать в любом регистре.

Имейте в виду, что `NULL` отличается от таких значений, как 0 для числовых или пустая строка — для строковых (см. раздел A.1.3).

Для операций импорта или экспорта, выполняемых с помощью `LOAD DATA INFILE` или `SELECT...INTO OUTFILE`, значение `NULL` представляется как последовательность `\N`. См. раздел 6.1.5.

## 2.2. Имена баз данных, таблиц, индексов, столбцов и псевдонимов

Имена баз данных, таблиц, индексов, столбцов и псевдонимов — это идентификаторы. В настоящем разделе описан допустимый синтаксис для идентификаторов в MySQL.

В табл. 2.1 приведены максимальные длины и допустимые символы для каждого типа идентификаторов.

Таблица 2.1. Максимальные длины и допустимые символы для идентификаторов

Идентификатор	Максимальная длина (в байтах)	Допустимые символы
База данных	64	Любые символы, разрешенные в именах каталогов, кроме <code>'/'</code> , <code>'\'</code> и <code>'.'</code> .
Таблица	64	Любые символы, разрешенные в именах файлов, кроме <code>'/'</code> , <code>'\'</code> и <code>'.'</code> .
Столбец	64	Любые символы.
Индекс	64	Любые символы.
Псевдоним	255	Любые символы.

В дополнение к ограничениям, указанным в табл. 2.1, ни один идентификатор не может содержать символ ASCII 0 или байт со значением 255. Имена баз, таблиц и столбцов не должны завершаться пробелами. До MySQL 4.1.1 символы кавычек не должны были применяться с идентификаторами.

Начиная с MySQL 4.1.1, идентификаторы сохраняются в кодировке Unicode (UTF8). Это применимо к идентификаторам в определениях таблиц, хранящихся в файлах `.frm`, а также к идентификаторам, сохраняемым в таблицах привилегий базы данных `mysql`. Несмотря на то что идентификаторы Unicode могут включать в себя многобайтные символы, помните, что максимальная их длина указана в байтах. Если идентификатор включает многобайтные символы, то допустимое количество символов меньше, чем значение, приведенное в таблице.

Идентификатор может быть или не быть окруженным кавычками. Если идентификатор совпадает с зарезервированным словом либо содержит специальные символы, вы должны заключать его в кавычки всякий раз, когда к нему обращаетесь. Список зарезервированных слов приведен в разделе 2.6. Специальные символы находятся вне набора букв и цифр текущего символического набора плюс `'_'` и `'$'`.

Символом кавычки для окружения идентификаторов служит обратная кавычка (```):

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

Если SQL-режим сервера активизирует опцию `ANSI_QUOTES`, то можно также заключать идентификаторы в двойные кавычки:

```
mysql> CREATE TABLE "test" (col INT);
ERROR 1064: You have an error in your SQL syntax. (...)
```

```
mysql> SET sql_mode='ANSI_QUOTES';
mysql> CREATE TABLE "test" (col INT);
Query OK, 0 rows affected (0.00 sec)
```

См. раздел 1.8.2.

Начиная с версии MySQL 4.1, символы кавычек могут находиться внутри идентификатора, если сам идентификатор взят в кавычки. Если символ, который нужно включить в состав идентификатора, совпадает с тем, что служит для него кавычкой, его следует повторить. Следующий оператор создает таблицу под названием а`б, которая содержит столбец по имени с"d:

```
mysql> CREATE TABLE `a``b` (`c"d` INT);
```

Помещение идентификаторов в кавычки было введено в MySQL 3.23.6 с тем, чтобы разрешить использование идентификаторов, совпадающих с зарезервированными словами или содержащих специальные символы. До версии 3.23.6 нельзя было применять идентификаторы, которые требуют кавычек, поэтому правила образования идентификаторов для старых версий MySQL более строгие:

- Имя может включать алфавитно-цифровые символы из текущего набора символов, а также символы '\_' и '\$'. В качестве набора символов по умолчанию принят ISO-8859-1 (Latin1). Его можно изменить с помощью опции `mysqld --default-character-set`.
- Имя может начинаться с любого допустимого символа. В частности, имя может начинаться с цифры, что весьма отличается от множества других систем управления базами данных. Однако имя, не помещенное в кавычки, не может содержать *только* цифры.
- В именах нельзя применять символ точки '.', поскольку точка служит для расширения формата, когда можно обращаться к столбцам (см. раздел 2.2.1).

Не рекомендуется использовать имена вроде `1e`, потому что выражение, подобное `1e+1`, неоднозначно. Оно может интерпретироваться как выражение `1e + 1` либо как число `1e+1`, в зависимости от контекста.

## 2.2.1. Идентификационные квалификаторы

MySQL разрешает имена, которые состоят из одного или нескольких идентификаторов. Компоненты составных имен должны разделяться символом точки '.'. Начальные (левые) части составных идентификаторов служат квалификаторами, влияющими на контекст, в котором конечный идентификатор должен интерпретироваться.

В MySQL вы можете обращаться к столбцу, используя любую из следующих форм:

Ссылка на столбец	Смысл
<code>имя_столбца</code>	Столбец <code>имя_столбца</code> из любой таблицы, используемой в запросе и содержащей столбец с таким именем.
<code>имя_таблицы.имя_столбца</code>	Столбец <code>имя_столбца</code> из таблицы <code>имя_таблицы</code> базы данных по умолчанию.
<code>имя_базы_данных.имя_таблицы.имя_столбца</code>	Столбец <code>имя_столбца</code> из таблицы <code>имя_таблицы</code> базы данных <code>имя_базы_данных</code> . Этот синтаксис не применялся до версии MySQL 3.22.

Если любой компонент составного идентификатора требует заключения в кавычки, это следует делать индивидуально, а не для всего имени в целом. Например, `'my-table'.``'my-column'` правильно, в то время как `'my-table.my-column'` — нет.

Нет необходимости специфицировать префиксы *имя\_базы\_данных* или *имя\_таблицы* для ссылки на столбцы в операторах, если только ссылка не является неоднозначной. Предположим, что обе таблицы, и `t1`, и `t2`, имеют столбец с именем `c`, и выполняется чтение `c` с помощью оператора `SELECT`, который использует и `t1`, и `t2`. В этом случае имя `c` неоднозначно, потому что оно не уникально в таблицах, участвующих в запросе. Вы обязаны квалифицировать его именем таблицы, подобно `t1.c` или `t2.c`, чтобы указать, какая таблица имеется в виду. Подобным же образом, чтобы обратиться к таблице `t` базы данных `db1` и таблице `t` базы данных `db2` в одном операторе, вы должны ссылаться на столбцы этих таблиц: `db1.t.имя_столбца` и `db2.t.имя_столбца`.

Синтаксис `.имя_таблицы` означает таблицу *имя\_таблицы* в текущей базе данных. Этот синтаксис принят для совместимости с ODBC, поскольку некоторые программы ODBC снабжают имена таблиц префиксом `'.'`.

## 2.2.2. Чувствительность идентификаторов к регистру

В MySQL базы данных соответствуют подкаталогам каталога данных. Таблицы внутри базы данных соответствуют, как минимум, одному файлу, находящемуся внутри подкаталога (а возможно, и большему числу файлов, в зависимости от применяемого механизма хранения). Следовательно, зависимость от регистра имен файлов и каталогов базовой операционной системы определяет зависимость от регистра имен баз данных и таблиц. Это означает, что имена баз и таблиц не зависят от регистра в Windows, но зависят в большинстве вариантов Unix. Одним заслуживающим упоминания исключением является система Mac OS X, которая основана на Unix, но использует по умолчанию файловую систему (HFS+), имена в которой не зависят от регистра. Однако Mac OS X также поддерживает тома UFS, в которых имена чувствительны к регистру, как в любой системе Unix. См. раздел 1.8.1.

### На заметку!

Несмотря на то что имена баз данных и таблиц нечувствительны к регистру в некоторых платформах, вы не должны обращаться к базе данных или таблице, используя различные регистры, в пределах одного запроса. Следующий запрос не будет работать, потому что обращается к таблице и как к `my_table`, и как к `MY_TABLE`:

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

Имена столбцов, индексов и псевдонимы столбцов не зависят от регистра в любой платформе.

Псевдонимы таблиц были чувствительны к регистру до версии MySQL 4.1.1. Следующий запрос не будет работать, поскольку обращается к псевдониму и как к `a`, и как к `A`:

```
mysql> SELECT имя_столбца FROM имя_таблицы AS a
-> WHERE a.имя_столбца = 1 OR A.имя_столбца = 2;
```

Если у вас возникают сложности с запоминанием допустимых регистров для имен баз данных и таблиц, примите непротиворечивое соглашение, например, всегда создавать базы данных и таблицы с именами в нижнем регистре.

Как имена таблиц сохраняются на диске и используются MySQL – определено системной переменной `lower_case_table_names`, которую можно установить во время запуска `mysqld`.

`lower_case_table_names` может принимать одно из следующих значений:

Значение	Смысл
0	Имена баз данных и таблиц сохраняются на диске, используя регистр, указанный в операторе <code>CREATE TABLE</code> или <code>CREATE DATABASE</code> . При сравнении имен регистр учитывается. Это значение по умолчанию для систем Unix. Следует отметить, что в случае установки этого значения равным 0 с помощью опции <code>--lower_case_table_names=0</code> в нечувствительной к регистру операционной системе, и обращении к именам таблиц MySQL с использованием разных регистров символов могут повредиться индексы.
1	Имена таблиц сохраняются в нижнем регистре и сравнения имен нечувствительны к регистру. MySQL преобразует все имена таблиц к нижнему регистру при сохранении и поиске. Это поведение также относится к именам баз данных, начиная с версии MySQL 4.0.2, и псевдонимам имен таблиц, начиная с версии MySQL 4.1.1. Это значение по умолчанию в системах Windows и Mac OS X.
2	Имена баз данных и таблиц сохраняются на диске, используя регистр, указанный в операторе <code>CREATE TABLE</code> или <code>CREATE DATABASE</code> , но MySQL преобразует их к нижнему регистру при поиске. Сравнения имен нечувствительны к регистру. Следует отметить, что это работает <i>только</i> в файловых системах, имена в которых нечувствительны к регистру. Имена таблиц InnoDB сохраняются в нижнем регистре, как при <code>lower_case_table_names=1</code> . Установка <code>lower_case_table_names</code> равным 2 возможна, начиная с версии MySQL 4.0.18.

Если вы эксплуатируете MySQL только на одной платформе, обычно вам незачем изменять значение переменной `lower_case_table_names`. Однако вы можете столкнуться с трудностями при переносе таблицы между платформами, которые по-разному относятся к регистру символов в именах. Например, в Unix вы можете иметь две разные таблицы с именами `my_table` и `MY_TABLE`, тогда как в Windows эти имена рассматриваются как одно и то же. Чтобы избежать проблем переноса, вызванных регистром символов имен таблиц, существует два выбора:

- Использовать `lower_case_table_names=1` во всех системах. Главное неудобство, связанное с этим, состоит в том, что когда вы используете `SHOW TABLES` или `SHOW DATABASES`, то не увидите имена в их истинном регистре.
- Использовать `lower_case_table_names=0` в системах Unix и `lower_case_table_names=2` в системах Windows. Это предохранит регистр символов в именах баз и таблиц. Неудобство такого решения связано с необходимостью гарантирования, что запросы в Windows всегда обращаются к таблицам в правильном регистре. Если вы переносите запросы в среду Unix, где регистр имеет значение, они не будут работать, если регистр символов не тот.

Следует отметить, что перед установкой `lower_case_table_names` равным 1 в Unix вы должны сначала преобразовать старые имена баз и таблиц к нижнему регистру перед перезапуском `mysqld`.

## 2.3. Пользовательские переменные

MySQL поддерживает пользовательские переменные, начиная с версии 3.23.6. Вы можете сохранять значение в пользовательских переменных и обращаться к ним позже, что позволяет передавать значения из одного оператора в другой. Пользовательские переменные связаны с подключением. Это означает, что переменная, определенная одним клиентом, не может быть видима и используема другими. Все переменные клиентского соединения автоматически освобождаются, когда клиент отключается.

Пользовательские переменные записываются как `@имя_переменной`, где имя переменной `имя_переменной` может включать алфавитно-цифровые символы текущего набора символов, а также `'_'` и `'$'`. Набором символов по умолчанию является ISO-8859-1 (Latin1). Его можно изменить с помощью опции `mysqld --default-character-set`. Имена пользовательских переменных нечувствительны к регистру, начиная с версии MySQL 5.0. В более ранних версиях они зависят от регистра.

Один из способов установки пользовательских переменных предусматривает применение оператора SET:

```
SET @имя_переменной = выражение [, @имя_переменной = выражение] ...
```

В операторе SET для присвоения значений можно использовать `=` или `:=`. Выражение `выражение`, присваиваемое каждой переменной, может оцениваться как целое, действительное, строковое или NULL.

Вы также можете присваивать значения пользовательским переменным в операторах, отличных от SET. В этом случае операцией присваивания должна быть только `:=`, но не `=`, потому что `=` трактуется как операция сравнения в операторах, отличных от SET:

```
mysql> SET @t1=0, @t2=0, @t3=0;
mysql> SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
+-----+-----+-----+
| @t1:=(@t2:=1)+@t3:=4 | @t1 | @t2 | @t3 |
+-----+-----+-----+
| 5 | 5 | 1 | 4 |
+-----+-----+-----+
```

Пользовательские переменные могут быть использованы везде, где допускаются переменные. В настоящее время это не включает контексты, явно требующие числа, — такие как конструкция LIMIT оператора SELECT или IGNORE *число* операторов LOAD DATA.

Если вы обращаетесь к переменной, которая не была инициализирована, ее значение будет равно NULL.

### На заметку!

В операторе SELECT каждое выражение вычисляется только при отправке клиенту. Это значит, что в конструкциях HAVING, GROUP BY или ORDER BY нельзя ссылаться на выражение, которое включает переменные, устанавливаемые в списке SELECT. Например, следующий оператор не будет работать, как это ожидается:

```
mysql> SELECT (@aa:=id) AS a, (@aa+3) AS b FROM имя_таблицы HAVING b=5;
```

Ссылка на `b` в конструкции HAVING ссылается на псевдоним выражения в списке SELECT, который использует `@aa`. Это не работает, как ожидается: `@aa` не будет содержать значение текущей строки, а будет равно `id` из предыдущей извлеченной строки.



Основное правило гласит: нельзя одновременно присваивать и использовать переменную в одном и том же запросе.

Другое обстоятельство, касающееся установки переменной и использования ее в том же операторе, связано с тем, что тип возврата переменной по умолчанию основывается на типе переменной в начале оператора. Следующий пример иллюстрирует это:

```
mysql> SET @a='test';  
mysql> SELECT @a, (@a:=20) FROM имя_таблицы;
```

В этом операторе SELECT MySQL сообщит клиенту, что первый столбец строковый, и преобразует все обращения к @a в строки, даже несмотря на то, что ей присваивается число в следующей строке. После того, как SELECT будет выполнен, @a в следующем операторе будет восприниматься как число.

Во избежание проблем, связанных с таким поведением, либо не используйте переменную, установленную в том же операторе, или предварительно присваивайте переменной значение 0, 0.0 или '', чтобы указать тип при инициализации.

Переменные, которым не присвоено значение, содержат NULL и имеют строковый тип.

## 2.4. Системные переменные

Начиная с MySQL 4.0.3, мы предоставили лучший доступ к множеству системных переменных и сеансовых переменных соединений. Многие переменные можно изменять динамически – при работающем сервере. Это позволяет модифицировать поведение сервера без его останова и перезапуска.

Сервер mysqld поддерживает два типа переменных. Глобальные переменные влияют на все операции сервера. Сеансовые переменные влияют только на отдельное клиентское соединение.

Когда сервер стартует, он инициализирует все глобальные переменные значениями по умолчанию. Эти умолчания могут изменяться опциями, указанными в файле опций или в командной строке. После того, как сервер стартует, динамические глобальные переменные можно изменять, подключаясь к серверу и выполняя оператор SET GLOBAL *имя\_переменной*. Для изменения глобальных переменных нужно иметь привилегию SUPER.

Сервер также поддерживает набор сеансовых переменных для каждого подключенного клиента. Клиентские сеансовые переменные инициализируются во время установки соединения с сервером значениями, взятыми из соответствующих глобальных переменных.

Те сеансовые переменные, которые являются динамическими, клиент может изменять с помощью оператора SET SESSION *имя\_переменной*. Установка сеансовых переменных не требует специальных привилегий, но клиент может изменять только свои собственные сеансовые переменные, а никакого другого клиента.

Изменения в глобальных переменных видимы всем клиентам, которые имеют к ним доступ. Однако эти изменения затрагивают соответствующие сеансовые переменные, которые инициализируются значениями глобальных, только для клиентов, подключившихся после внесения изменений. Сеансовые переменные клиентов, подключившихся ранее, не изменяются (даже для того клиента, который выполняет оператор SET GLOBAL).

Глобальные или сеансовые переменные могут быть установлены и прочтены с применением одной из приведенных ниже синтаксических форм. В приведенных далее примерах задействована переменная `sort_buffer_size`.

Для установки глобальной переменной можно воспользоваться одним из следующих вариантов:

```
mysql> SET GLOBAL sort_buffer_size=значение;  
mysql> SET @@global.sort_buffer_size=значение;
```

Установить значение сеансовой переменной можно так:

```
mysql> SET SESSION sort_buffer_size=значение;  
mysql> SET @@session.sort_buffer_size=значение;  
mysql> SET sort_buffer_size=значение;
```

LOCAL – это синоним SESSION.

Если вы не указываете GLOBAL, SESSION или LOCAL при установке значения переменной, по умолчанию предполагается SESSION (см. раздел 6.5.3.1).

Чтобы получить значение глобальной переменной, используйте один из следующих операторов:

```
mysql> SELECT @@global.sort_buffer_size;  
mysql> SHOW GLOBAL VARIABLES like 'sort_buffer_size';
```

Получить значение сеансовой переменной можно одним из следующих способов:

```
mysql> SELECT @@sort_buffer_size;  
mysql> SELECT @@session.sort_buffer_size;  
mysql> SHOW SESSION VARIABLES like 'sort_buffer_size';
```

Здесь также LOCAL является синонимом SESSION.

Когда вы читаете значение переменной с помощью оператора SELECT *@@имя\_переменной* (то есть, не указывая global, session или local), MySQL возвращает значение сеансовой переменной, если она существует, и глобальной – в противном случае.

Для SHOW VARIABLES, если не указано ни SESSION, ни GLOBAL, ни LOCAL, MySQL возвращает SESSION.

Причина для обязательного указания слова GLOBAL при установке исключительно глобальных переменных состоит в том, чтобы избежать проблем в будущем. Если удалить SESSION-переменную с тем же именем, что и GLOBAL, то клиент с привилегией SUPER может нечаянно изменить GLOBAL-переменную вместо того, чтобы изменить только SESSION-переменную, принадлежащую его собственному сеансу. Если добавить GLOBAL-переменную с тем же именем, что и существующая SESSION-переменная, то клиент, желая изменить GLOBAL-переменную, изменит только свою SESSION-переменную.

Дополнительную информацию о стартовых опциях и системных переменных можно найти в книге *MySQL. Руководство администратора* (М.: Издательский дом “Вильямс”, 2005, ISBN 5-8459-0805-1). Там же приведен список переменных, которые можно изменять во время работы сервера.

## 2.4.1. Структурированные системные переменные

Структурированные системные переменные поддерживаются, начиная с MySQL 4.1.1. Структурированные переменные отличаются от обычных системных переменных в двух аспектах:

- Их значения являются структурами с компонентами, представляющими параметры сервера, которые близко связаны друг с другом.
- Может существовать несколько экземпляров данного типа структурированной переменной. Они имеют различные имена и ссылаются на различные ресурсы, поддерживаемые сервером.

В настоящий момент MySQL поддерживает только один тип структурированных переменных. Он специфицирует параметры, управляющие операциями кэшей ключей. Структурированные переменные кэш ключей имеют следующие компоненты:

- `key_buffer_size`
- `key_cache_block_size`
- `key_cache_division_limit`
- `key_cache_age_threshold`

Целью настоящего раздела является описание синтаксиса ссылок на структурированные переменные. Переменные кэш ключей используются для демонстрации примеров синтаксиса, а специфические детали о работе с ключевыми кэшами можно найти в книге *MySQL. Руководство администратора*.

Чтобы сослаться на компонент структурированной переменной, вы можете использовать составное имя в формате `имя_экземпляра.имя_компонента`. Вот примеры:

```
hot_cache.key_buffer_size
hot_cache.key_cache_block_size
cold_cache.key_cache_block_size
```

Для каждой структурированной системной переменной всегда существует предопределенный экземпляр с именем `default`. Если вы обращаетесь к компоненту структурированной переменной без имени экземпляра, используется экземпляр `default`. Поэтому `default.key_buffer_size` и `key_buffer_size` ссылаются на одну и ту же системную переменную.

Правила именования экземпляров и компонентов структурированных переменных формулируются следующим образом:

- Для заданного типа структурированных переменных каждый экземпляр должен иметь имя, уникальное среди переменных данного типа. Однако имена переменных не обязаны быть уникальными между разными типами переменных. Например, каждая структурированная переменная имеет экземпляр с именем `default`, таким образом, `default` – не уникальное имя между разными типами.
- Имена компонентов каждого типа структурированных переменных должны быть уникальны среди всех имен системных переменных. Если бы это было не так (то есть, два разных типа структурированных переменных имели бы компоненты с одинаковыми именами), было бы не ясно, на какую переменную `default` ссылается имя члена структурированной переменной, если оно не квалифицировано именем экземпляра.
- Если имя экземпляра структурированной переменной не является корректным в виде идентификатора без кавычек, обращайтесь к нему с использованием обратных одинарных кавычек. Например, `hot-cache` – недопустимое имя, но `'hot-cache'` – вполне законное.
- `global`, `session` и `local` не являются правильными именами экземпляров. Это позволяет избежать конфликтов с нотацией, таких как `@@global.имя_переменной`, для ссылки на неструктурированные системные переменные.

В настоящий момент первые два правила нарушить невозможно, потому что существует только один тип структурированных переменных для описания параметров кэш ключей. Эти правила обретут большее значение в будущем, когда появятся какие-то другие типы структурированных переменных.

Можно ссылаться на компоненты структурированных переменных, используя составные имена в любом контексте, где допустимо появление простых имен переменных. Например, вы можете присваивать значение структурированным переменным, используя опции командной строки:

```
shell> mysql --hot_cache.key_buffer_size=64K
```

В файле опций поступите следующим образом:

```
[mysql]  
hot_cache.key_buffer_size=64K
```

Если вы запускаете сервер с подобной опцией, он создает кэш ключей с именем `hot_cache` размером 64 Кбайт в дополнение к кэшу ключей по умолчанию, который имеет размер 8 Мбайт.

Предположим, что сервер запускается так:

```
shell> mysql --key_buffer_size=256K \  
--extra_cache.key_buffer_size=128K \  
--extra_cache.key_cache_block_size=2048
```

В этом случае сервер устанавливает размер кэша ключей по умолчанию в 256 Кбайт (можно также указать `--default.key_buffer_size=256K`). В дополнение сервер создает второй кэш ключей с именем `extra_cache`, который имеет размер 128 Кбайт и размер блока буферов для кэширования индексных блоков таблиц в 2048 байт.

В следующем примере сервер запускается с тремя различными кэшами ключей, которые имеют отношение размеров 3:1:1:

```
shell> mysql --key_buffer_size=6M \  
--hot_cache.key_buffer_size=2M \  
--cold_cache.key_buffer_size=2M
```

Значения структурированных переменных могут устанавливаться и извлекаться во время выполнения. Например, для установки размера кэша ключей с именем `hot_cache` равным 10 Мбайт, воспользуйтесь любым из следующих операторов:

```
mysql> SET GLOBAL hot_cache.key_buffer_size = 10*1024*1024;  
mysql> SET @@global.hot_cache.key_buffer_size = 10*1024*1024;
```

Чтобы получить размер кэша, сделайте так:

```
mysql> SELECT @@global.hot_cache.key_buffer_size;
```

Однако следующий оператор работать не будет. Переменная интерпретируется не как составное имя, а как простая строка в конструкции `LIKE` сопоставления с шаблоном:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'hot_cache.key_buffer_size';
```

Это единственное исключение из правила, которое гласит, что имя составной переменной может применяться везде, где допустимо появление простого имени.

## 2.5. Синтаксис комментариев

Сервер MySQL поддерживает три стиля комментариев:

- От символа `'#'` до конца строки.
- От последовательности `--` до конца строки. Этот стиль поддерживается, начиная с версии MySQL 3.23.3. Отметим, что стиль комментария с `--` требует, что-

бы за вторым тире следовал как минимум один пробел (или управляющий символ, такой как перевод строки). Этот синтаксис слегка отличается от стандартного стиля комментариев SQL (см. раздел 1.8.5.7).

- От последовательности `'/*'` до последовательности `'*/'`. Замыкающая последовательность не должна быть на той же строке, поэтому синтаксис позволяет писать многострочные комментарии.

В следующем примере представлены все три стиля комментариев:

```
mysql> SELECT 1+1; # Этот комментарий продолжается до конца строки
mysql> SELECT 1+1; -- Этот комментарий продолжается до конца строки
mysql> SELECT 1 /* это внутрискрочный комментарий */ + 1;
mysql> SELECT 1+
/*
А это
многострочный комментарий
*/
1;
```

Описанный выше синтаксис комментариев влияет на то, как сервер `mysqld` разбирает SQL-операторы. Клиентская программа `mysql` также выполняет некоторый предварительный анализ операторов, перед тем как отправлять их серверу. (Например, она делает это, чтобы определить границы операторов во входной строке с множеством операторов.) Однако, есть некоторые ограничения в том, как `mysql` разбирает комментарии `/* ... */`:

- Символы одинарной, двойной и обратной кавычки воспринимается для индикации начала помещенной в кавычки строки или идентификатора, даже внутри комментария. Если открывающей кавычке в теле комментария не соответствует закрывающая, анализатор не считает, что комментарий не закрыт. Если вы запускаете `mysql` интерактивно, то можете обнаружить эту ситуацию по изменению приглашения командной строки с `mysql>` на `'>`, `>` или `'>`. Упомянутая проблема была решена в MySQL 4.1.1.
- Точка с запятой внутри комментария воспринимается как завершение оператора SQL, и то, что за ней следует, является началом следующего оператора. Эта проблема была решена в MySQL 4.0.13.

В версиях MySQL, которые имеют эти недостатки, они проявляются как при интерактивном запуске `mysql`, так и тогда, когда вы помещаете команды в файл и применяете `mysql` в пакетном режиме с помощью команды `mysql < имя_файла`.

## 2.6. Трактовка зарезервированных слов MySQL

Общая проблема возникает при попытке использования идентификатора, такого как имя таблицы или столбца, совпадающего с именем встроенного в MySQL типа данных или именем функции, вроде `TIMESTAMP` или `GROUP`. Это делать разрешается (например, `ABS` допускается в качестве имени столбца). Однако по умолчанию не допускается наличие пробелов в вызове функции между именем функции и последующим символом `'('`. Это требование позволяет отличить вызов функции от ссылки на имя столбца.

Побочный эффект такого поведения заключается в том, что пропущенный пробел в некоторых контекстах заставляет идентификатор интерпретироваться как имя функции. Например, приведенный ниже оператор корректен:

```
mysql> CREATE TABLE abs (val INT);
```

Но если пропустить пробел после `abs`, возникнет синтаксическая ошибка, поскольку анализатор воспринимает это как вызов функции `ABS()`:

```
mysql> CREATE TABLE abs(val INT);
```

Если SQL-режим сервера включает значение `IGNORE_SPACE`, то в вызове функции разрешается оставлять пробел между ее именем и последующим символом `'('`. Это заставляет анализатор трактовать имена функций как зарезервированные слова. В результате идентификаторы, совпадающие с именами функций, должны быть взяты в кавычки, как описано в разделе 2.2.

Слова из представленной ниже таблицы явно зарезервированы в MySQL. Большинство из них запрещены стандартом SQL для применения в качестве имен столбцов и/или таблиц (например, `GROUP`). Некоторые слова зарезервированы потому, что MySQL нуждается в них и в настоящее время использует анализатор yacc. Зарезервированные слова можно использовать в качестве идентификаторов, помещая их в кавычки.

ADD	ALL	ALTER
ANALYZE	AND	AS
ASC	ASENSITIVE	AUTO_INCREMENT
BDB	BEFORE	BERKELEYDB
BETWEEN	BIGINT	BINARY
BLOB	BOTH	BY
CALL	CASCADE	CASE
CHANGE	CHAR	CHARACTER
CHECK	COLLATE	COLUMN
COLUMNS	CONDITION	CONNECTION
CONSTRAINT	CONTINUE	CREATE
CROSS	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURSOR	DATABASE
DATABASES	DAY_HOUR	DAY_MICROSECOND
DAY_MINUTE	DAY_SECOND	DEC
DECIMAL	DECLARE	DEFAULT
DELAYED	DELETE	DESC
DESCRIBE	DETERMINISTIC	DISTINCT
DISTINCTROW	DIV	DOUBLE
DROP	ELSE	ELSEIF
ENCLOSED	ESCAPED	EXISTS
EXIT	EXPLAIN	FALSE
FETCH	FIELDS	FLOAT
FOR	FORCE	FOREIGN
FOUND	FRAC_SECOND	FROM
FULLTEXT	GRANT	GROUP
HAVING	HIGH_PRIORITY	HOURL_MICROSECOND
HOURL_MINUTE	HOURL_SECOND	IF
IGNORE	IN	INDEX
INFILE	INNER	INNODB
INOUT	INSENSITIVE	INSERT
INT	INTEGER	INTERVAL
INTO	IO_THREAD	IS
ITERATE	JOIN	KEY
KEYS	KILL	LEADING

LEAVE	LEFT	LIKE
LIMIT	LINES	LOAD
LOCALTIME	LOCALTIMESTAMP	LOCK
LONG	LONGBLOB	LONGTEXT
LOOP	LOW_PRIORITY	MASTER_SERVER_ID
MATCH	MEDIUMBLOB	MEDIUMINT
MEDIUMTEXT	MIDDLEINT	MINUTE_MICROSECOND
MINUTE_SECOND	MOD	NATURAL
NOT	NO_WRITE_TO_BINLOG	NULL
NUMERIC	ON	OPTIMIZE
OPTION	OPTIONALLY	OR
ORDER	OUT	OUTER
OUTFILE	PRECISION	PRIMARY
PRIVILEGES	PROCEDURE	PURGE
READ	REAL	REFERENCES
REGEXP	RENAME	REPEAT
REPLACE	REQUIRE	RESTRICT
RETURN	REVOKE	RIGHT
RLIKE	SECOND_MICROSECOND	SELECT
SENSITIVE	SEPARATOR	SET
SHOW	SMALLINT	SOME
SONAME	SPATIAL	SPECIFIC
SQL	SQLEXCEPTION	SQLSTATE
SQLWARNING	SQL_BIG_RESULT	SQL_CALC_FOUND_ROWS
SQL_SMALL_RESULT	SQL_TSI_DAY	SQL_TSI_FRAC_SECOND
SQL_TSI_HOUR	SQL_TSI_MINUTE	SQL_TSI_MONTH
SQL_TSI_QUARTER	SQL_TSI_SECOND	SQL_TSI_WEEK
SQL_TSI_YEAR	SSL	STARTING
STRAIGHT_JOIN	STRIPED	TABLE
TABLES	TERMINATED	THEN
TIMESTAMPADD	TIMESTAMPDIFF	TINYBLOB
TINYINT	TINYTEXT	TO
TRAILING	TRUE	UNDO
UNION	UNIQUE	UNLOCK
UNSIGNED	UPDATE	USAGE
USE	USER_RESOURCES	USING
UTC_DATE	UTC_TIME	UTC_TIMESTAMP
VALUES	VARBINARY	VARCHAR
VARCHARACTER	VARYING	WHEN
WHERE	WHILE	WITH
WRITE	XOR	YEAR_MONTH
ZEROFILL		

Следующие ключевые слова разрешены в MySQL для использования в качестве имен таблиц и столбцов. Это связано с тем, что они представляют собой очень естественные слова и большинство людей применяют их.

ACTION	ENUM	TIME
BIT	NO	TIMESTAMP
DATE	TEXT	

## Поддержка наборов символов

Усовершенствованная поддержка для управления символьными наборами была добавлена в MySQL версии 4.1. Описанные здесь средства реализованы в MySQL 4.1.1. (MySQL 4.1.0 обладает некоторыми из них, но не всеми, и некоторые реализованы иначе.)

Настоящая глава посвящена обсуждению следующих вопросов:

- Что такое наборы символов и порядки сопоставления
- Многоуровневая система умолчаний.
- Новый синтаксис MySQL 4.1.
- Затронутые функции и операции.
- Поддержка Unicode.
- Значение каждого индивидуального набора символов и порядка сопоставления.

Поддержка символьных наборов в настоящее время включена в механизмы хранения MyISAM, MEMORY (HEAP) и (начиная с MySQL 4.1.2) InnoDB. Механизм хранения ISAM не поддерживает наборы символов, и это не планируется, потому что ISAM считается устаревшим.

### 3.1. Общие сведения о наборах символов и порядках сопоставления

**Символьный набор** (character set) – это набор символов и кодировок. **Порядок сопоставления** (collation) – это совокупность правил сравнения символов в наборе. Проясним разницу на примере воображаемого символьного набора.

Предположим, что у нас есть алфавит из четырех символов: 'А', 'В', 'а', 'б'. Присвоим каждому символу номер: 'А'=0, 'В'=1, 'а'=2, 'б'=3. Буква 'А' – это символ, число 0 – код 'А', а комбинация всех четырех символов и их кодов образует **символьный набор**.

Теперь предположим, что мы хотим сравнить два строковых значения, 'А' и 'В'. Простейший способ сделать это – посмотреть на их коды: 0 – для 'А' и 1 – для 'В'. То, что мы только что сделали – это применили порядок сопоставления к нашему символьному набору. Порядок сопоставления – это набор правил (в данном случае правило только од-



но): “сравнить коды”. Мы называем этот простейший из всех возможных порядков сопоставления **бинарным**.

Но что делать, если мы захотим сказать, что заглавные и прописные буквы эквивалентны? В этом случае мы имеем, по крайней мере, два правила: (1) трактовать прописные буквы ‘а’ и ‘б’ как эквивалентные ‘А’ и ‘В’; (2) сравнить коды. Мы называем это порядком сопоставления, не зависящим от регистра. Он немного сложнее, чем бинарный порядок.

В реальной жизни большинство символьных наборов имеют множество символов, не только ‘А’ и ‘В’, а полные алфавиты, иногда несколько алфавитов, или восточные иероглифические системы с тысячами символов, вместе с множеством специальных символов и знаков пунктуации. Как и в реальной жизни, большинство порядков сопоставления подчиняются множеству правил: не только нечувствительность к регистру, но также нечувствительность к диакритике (диакритика – это метка, добавляемая к символам, как в немецком ‘ö’) и многосимвольным отображениям (таким, как правило, что ‘ö’ = ‘oe’ в одной из систем сравнения символов немецкого языка).

MySQL 4.1 может выполнять следующие вещи:

- Сохранять строки с использованием различных символьных наборов.
- Сравнить строки с применением различных порядков сопоставления.
- Смешивать строки в различных символьных наборах или с разными порядками хранения на одном сервере, в одной базе данных и даже в одной таблице.
- Позволяет специфицировать символьные наборы и порядки сопоставления на любом уровне.

В этих отношениях MySQL 4.1 не только намного удобнее, чем MySQL 4.0, он также впереди многих других СУБД. Однако, чтобы использовать эти новые средства эффективно, вам следует изучить, какие доступны символьные наборы и порядки сопоставления, как изменять их умолчания, и какие строковые операторы применять к ним.

## 3.2. Символьные наборы и порядки сопоставления MySQL

Сервер MySQL может поддерживать множество символьных наборов. Для получения полного списка доступных символьных наборов воспользуйтесь оператором `SHOW CHARACTER SET`:

```
mysql> SHOW CHARACTER SET;
```

Charset	Description	Default collation
big5	Big5 Traditional Chinese	big5_chinese_ci
dec8	DEC West European	dec8_swedish_ci
cp850	DOS West European	cp850_general_ci
hp8	HP West European	hp8_english_ci
koi8r	KOI8-R Relcom Russian	koi8r_general_ci
latin1	ISO 8859-1 West European	latin1_swedish_ci
latin2	ISO 8859-2 Central European	latin2_general_ci

...

Вывод также включает другой столбец, который не приведен, дабы пример поместился на страницу.

Любой из символьных наборов всегда имеет, как минимум, один порядок сопоставления. Он может также иметь несколько порядков сопоставления.

Чтобы получить список порядков сопоставления для символьного набора, используйте оператор `SHOW COLLATION`. Например, чтобы увидеть порядки сопоставления для набора `latin1` ("ISO-8859-1 West European"), воспользуйтесь приведенным ниже оператором для поиска порядков сопоставления, которые начинаются с `'latin1'`:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-----+-----+-----+-----+-----+-----+
| Collation          | Charset | Id   | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+-----+
| latin1_german1_ci  | latin1  | 5    |         |         | 0        |
| latin1_swedish_ci  | latin1  | 8    | Yes     | Yes      | 1        |
| latin1_danish_ci   | latin1  | 15   |         |         | 0        |
| latin1_german2_ci  | latin1  | 31   |         | Yes      | 2        |
| latin1_bin         | latin1  | 47   |         | Yes      | 1        |
| latin1_general_ci  | latin1  | 48   |         |         | 0        |
| latin1_general_cs  | latin1  | 49   |         |         | 0        |
| latin1_spanish_ci  | latin1  | 94   |         |         | 0        |
+-----+-----+-----+-----+-----+-----+
```

Порядки сопоставления `latin1` имеют следующие назначения:

Порядок	Описание
<code>latin1_bin</code>	Бинарный, в соответствии с кодировкой <code>latin1</code> .
<code>latin1_danish_ci</code>	Датский/норвежский.
<code>latin1_general_ci</code>	Многоязычный.
<code>latin1_general_cs</code>	Многоязычный, чувствительный к регистру.
<code>latin1_german1_ci</code>	Немецкий DIN-1.
<code>latin1_german2_ci</code>	Немецкий DIN-2.
<code>latin1_spanish_ci</code>	Современный испанский.
<code>latin1_swedish_ci</code>	Шведский/финский.

Порядки сопоставления обладают следующими общими характеристиками:

- Два разных символьных набора не могут иметь один и тот же порядок сопоставления.
- Каждый символьный набор имеет один порядок сопоставления, называемый порядком по умолчанию. Например, порядком сопоставления по умолчанию для `latin1` является `latin1_swedish_ci`.
- Существует соглашение об именах порядков. Имена начинаются с имени символьного набора, с которым ассоциированы (обычно включают наименование языка), и завершаются на `_ci` (не зависящие от регистра), `_cs` (зависящие от регистра), `_bin` (бинарные), или `_uca` (Unicode Collation Algorithm – порядок сопоставления Unicode, см. <http://www.unicode.org/reports/tr10/>).

## 3.3. Определение символьного набора и порядка сопоставления по умолчанию

Настройки по умолчанию для символьных наборов и порядков сопоставления можно разделить на четыре уровня: сервер, база данных, таблица и соединение. Последующее описание может показаться сложным, но на практике обнаруживается, что многоуровневые умолчания приводят к естественным и вполне очевидным результатам.

### 3.3.1. Наборы символов и порядки сопоставления на уровне сервера

Сервер MySQL обладает серверным набором символов и порядком сопоставления, которые могут быть ненулевыми.

MySQL определяет серверный набор символов и порядок сопоставления следующим образом:

- В соответствии с установками опций при старте сервера.
- В соответствии со значениями, установленными во время выполнения.

На уровне сервера решение достаточно простое. Серверный набор символов и порядок сопоставления изначально зависит от опций, которые используются при запуске `mysqld`. Вы можете применить `--default-character-set` для установки символьного набора, и к нему добавить `--default-collation` для установки порядка сопоставления. Если вы не указываете символьный набор, это то же самое, что задать `--default-character-set=latin1`. Если вы указываете только символьный набор (например, `latin1`), но не порядок сопоставления, это то же самое, что задать `--default-character-set=latin1 --default-collation=latin1_swedish_ci`, поскольку `latin1_swedish_ci` — порядок сопоставления по умолчанию для набора `latin1`. Таким образом, следующие три команды дают одинаковый эффект:

```
shell> mysqld
shell> mysqld --default-character-set=latin1
shell> mysqld --default-character-set=latin1 \
            --default-collation=latin1_swedish_ci
```

Единственный способ изменить эту установку — перекомпилировать сервер. Если вы хотите изменить символьный набор сервера по умолчанию и его порядок сопоставления при сборке сервера из исходных текстов, используйте аргументы сценария `configure`: `--with-charset` и `--with-collation`. Например:

```
shell> ./configure --with-charset=latin1
```

или:

```
shell> ./configure --with-charset=latin1 \
            --with-collation=latin1_german1_ci
```

Как `mysqld`, так и `configure` проверяют, чтобы комбинация символьного набора и порядка сопоставления была корректной. Если это не так, каждая из программ выдает сообщение об ошибке и завершается.

Текущий символьный набор и порядок сопоставления, действующие на сервере, доступны через значения системных переменных `character_set_server` и `collation_server`. Их значение можно изменять во время работы сервера.

### 3.3.2. Наборы символов и порядки сопоставления на уровне базы данных

Каждая база данных имеет символьный набор и порядок сопоставления, которые могут быть ненулевыми.

Операторы CREATE DATABASE и ALTER DATABASE имеют необязательные конструкции для спецификации набора символов и порядка сопоставления для базы данных:

```
CREATE DATABASE имя_базы_данных
  [[DEFAULT] CHARACTER SET имя_набора]
  [[DEFAULT] COLLATE имя_порядка_сопоставления]
ALTER DATABASE имя_базы_данных
  [[DEFAULT] CHARACTER SET имя_набора]
  [[DEFAULT] COLLATE имя_порядка_сопоставления]
```

Пример:

```
CREATE DATABASE имя_базы_данных
  DEFAULT CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

MySQL выбирает символьный набор и порядок сопоставления для базы данных следующим образом:

- Если указаны и CHARACTER SET X, и COLLATE Y, принимается набор символов X и порядок сопоставления Y.
- Если CHARACTER SET X указан без COLLATE, принимается набор символов X и его порядок сопоставления по умолчанию.
- В противном случае принимается символьный набор и порядок сопоставления, установленные на сервере.

Синтаксис оператора MySQL CREATE DATABASE...DEFAULT CHARACTER SET... аналогичен стандартному синтаксису SQL CREATE SCHEMA...CHARACTER SET...

Таким образом, существует возможность создавать базы данных с различными наборами символов и порядками сопоставления на одном сервере MySQL.

Набор символов и порядок сопоставления по умолчанию, установленный для базы данных, можно прочитать через системные переменные character\_set\_database и collation\_database. Сервер устанавливает значение этих переменных всякий раз, когда меняется база данных по умолчанию. Если в данный момент нет базы по умолчанию, эти переменные принимают значение, равное значению соответствующих переменных уровня сервера, — character\_set\_server и collation\_server.

### 3.3.2. Наборы символов и порядки сопоставления на уровне таблицы

Каждая таблица имеет свой набор символов и порядок сопоставления, которые могут быть ненулевыми. Операторы CREATE TABLE и ALTER TABLE имеют необязательные конструкции, специфицирующие набор символов и порядок сопоставления:

```
CREATE TABLE имя_таблицы (column_list)
  [[DEFAULT] CHARACTER SET имя_набора_символов [COLLATE имя_порядка_сопоставления]]
ALTER TABLE имя_таблицы
  [[DEFAULT] CHARACTER SET имя_набора_символов] [COLLATE имя_порядка_сопоставления]
```

Пример:

```
CREATE TABLE t1 ( ... )  
  DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

MySQL выбирает набор символов и порядок сопоставления для таблицы следующим образом:

- Если указанные и CHARACTER SET *X*, и COLLATE *Y*, принимается набор символов *X* и порядок сопоставления *Y*.
- Если CHARACTER SET *X* указан без COLLATE, принимается набор символов *X* и его порядок сопоставления по умолчанию.
- В противном случае принимается символьный набор и порядок сопоставления, установленные в базе данных.

Символьный набор и порядок сопоставления таблицы используются как значения по умолчанию, если они не указаны в индивидуальных определениях столбцов. Символьный набор и порядок сопоставления уровня таблицы являются расширением MySQL; в стандарте SQL они не предусмотрены.

### 3.3.4. Наборы символов и порядки сопоставления на уровне столбца

Каждый символьный столбец (то есть, столбцы типа CHAR, VARCHAR или TEXT) имеет свой набор символов и порядок сопоставления, которые могут быть ненулевыми. Синтаксис определения столбца предусматривает необязательные конструкции для указания набора символов и порядка сопоставления:

```
имя_столбца {CHAR | VARCHAR | TEXT} {длина_столбца}  
  [CHARACTER SET имя_набора_символов [COLLATE имя_порядка_сопоставления]]
```

Пример:

```
CREATE TABLE Table1  
(  
  column1 VARCHAR(5) CHARACTER SET latin1 COLLATE latin1_german1_ci  
);
```

MySQL выбирает набор символов и порядок сопоставления для столбца следующим образом:

- Если указанные и CHARACTER SET *X*, и COLLATE *Y*, принимается набор символов *X* и порядок сопоставления *Y*.
- Если CHARACTER SET *X* указан без COLLATE, принимается набор символов *X* и его порядок сопоставления по умолчанию.
- В противном случае принимается символьный набор и порядок сопоставления, установленные для таблицы.

Конструкции CHARACTER SET и COLLATE определены в стандарте SQL.

### 3.3.5. Примеры назначения символьного набора и порядка сопоставления

Следующие примеры демонстрируют, как MySQL определяет наборы символов и порядки сопоставления по умолчанию.

**Пример 1: таблица + определение столбца**

```
CREATE TABLE t1
(
  c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

Здесь мы имеем столбец с набором символов `latin1` и порядком сопоставления `latin1_german1_ci`. Определение явное, поэтому прямое. Следует отметить, что нет никаких проблем с сохранением столбца с `latin2` в таблице с `latin1`.

**Пример 2: таблица + определение столбца**

```
CREATE TABLE t1
(
  c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

На этот раз мы имеем столбец с набором символов `latin1` и порядком сопоставления по умолчанию. Здесь, несмотря на то, что это может показаться естественным, порядок сопоставления по умолчанию не берется с уровня таблицы. Вместо этого порядок сопоставления по умолчанию для `latin1` всегда принимается `latin1_swedish_ci`; столбец `c1` будет иметь порядок сопоставления `latin1_swedish_ci` (а не `latin1_danish_ci`).

**Пример 3: таблица + определение столбца**

```
CREATE TABLE t1
(
  c1 CHAR(10)
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

Имеем столбец с набором символов по умолчанию. При таких обстоятельствах MySQL обращается на уровень таблицы в поисках набора символов и порядка сопоставления для столбца. Поэтому символьным набором для столбца `c1` будет `latin1`, а порядком сопоставления – `latin1_danish_ci`.

**Пример 4: база данных + таблица + определение столбца**

```
CREATE DATABASE d1
  DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
USE d1;
CREATE TABLE t1
(
  c1 CHAR(10)
);
```

Мы определяем столбец без указания набора символов и порядка сопоставления. Мы также не указываем их на уровне таблицы. В этом случае MySQL обращается на уровень базы данных. (Установки базы данных становятся установками таблицы, и, как следствие, установками для столбца.) Поэтому символьным набором для столбца `c1` будет `latin2`, а порядком сопоставления – `latin2_czech_ci`.

### 3.3.6. Наборы символов и порядки сопоставления на уровне соединения

Некоторые переменные набора символов и порядка сопоставления имеют отношение к взаимодействию клиента с сервером. О части из них уже упоминались в предыдущих разделах:

- Серверный набор символов и порядок сопоставления доступны через системные переменные `character_set_server` и `collation_server`.
- Набор символов и порядок сопоставления базы данных по умолчанию доступны через системные переменные `character_set_database` и `collation_database`.

Дополнительные символические наборы и порядки сопоставления включаются в управление трафиком соединения клиента и сервера. Каждый клиент имеет набор символов и порядок сопоставления, связанные с соединением.

Рассмотрим, что такое “соединение”. “Соединение” – это то, что вы получаете при подключении к серверу. Клиент посылает SQL-операторы, подобные запросам, через соединение серверу. Сервер клиенту отправляет ответы, такие как результирующие наборы, по тому же соединению. Это вызывает некоторые вопросы об управлении символическим набором и порядком сопоставления для клиентских соединений, на каждый из которых можно ответить в терминах системных переменных:

- Какой набор символов имеет запрос, когда он покидает клиента?  
Сервер берет переменную `character_set_client` в качестве набора символов принимаемых от клиента запросов.
- В какой набор символов должен сервер транслировать запрос после его получения?  
Для этого сервер использует переменные `character_set_connection` и `collation_connection`. Он преобразует отправленные клиентом запросы из набора `character_set_client` в `character_set_connection` (за исключением строковых литералов, представленных как `_latin1` или `_utf8`). Переменная `collation_connection` важна для сравнения литеральных строк. Для сравнения строк со значениями столбцов это не имеет значения, поскольку порядок сопоставления столбцов имеет более высокий приоритет.
- В какой символический набор должен сервер транслировать результирующий набор или сообщение об ошибке перед отправкой их клиенту?  
Переменная `character_set_results` задает символический набор, в котором сервер возвращает результаты клиенту. Это включает в себя результирующие данные, такие как значения столбцов и метаданные результата, подобные именам столбцов.

Вы можете осуществить тонкую настройку установок этих переменных либо положиться на умолчания (в этом случае можно пропустить настоящий раздел).

Существуют два оператора, влияющие на набор символов соединения:

```
SET NAMES 'имя_набора_символов'  
SET CHARACTER SET имя_набора_символов
```

`SET NAMES` задает ожидаемый набор символов, в котором клиент будет отправлять SQL-операторы. То есть, `SET NAMES 'cp1251'` сообщает серверу, что “входящие сообщения от этого клиента будут в наборе символов `cp1251`”. Это также определяет набор символов для результатов, которые сервер будет отправлять обратно клиенту. (Напри-

мер, подобным образом задается набор символов для значений столбцов, возвращаемых оператором SELECT.)

Оператор SET NAMES 'x' эквивалентен следующим трем операторам:

```
mysql> SET character_set_client = x;  
mysql> SET character_set_results = x;  
mysql> SET character_set_connection = x;
```

Установка значения character\_set\_connection в x также устанавливает значение collation\_connection равным collation по умолчанию для x.

SET CHARACTER SET похож, однако устанавливает набор символов и порядок сопоставления для соединения такими же, как у базы данных по умолчанию. Оператор SET CHARACTER SET x эквивалентен следующим трем операторам:

```
mysql> SET character_set_client = x;  
mysql> SET character_set_results = x;  
mysql> SET collation_connection = @@collation_database;
```

Когда клиент подключается, он посылает серверу имя символьного набора, который желает использовать. Сервер устанавливает в переменных character\_set\_client, character\_set\_results и character\_set\_connection имя этого символьного набора. (В сущности, сервер выполняет операцию SET NAMES, используя этот набор.)

При работе с клиентской программой mysql нет необходимости выполнять SET NAMES каждый раз в начале работы, если вы хотите установить набор символов, отличный от принятого по умолчанию. Вы можете добавить опцию --default-character-set в командную строку вызова mysql либо в файл опций. Например, следующие установки файла опций изменяют значение трех переменных символьного набора на koi8r при каждом запуске mysql:

```
[mysql]  
default-character-set=koi8r
```

Рассмотрим пример. Предположим, что столбец column1 определен как CHAR(5) CHARACTER SET latin2.

Если вы не выполняете SET NAMES или SET CHARACTER SET, то, выполнив SELECT column1 FROM t, сервер отправит обратно все значения column1, используя символьный набор, указанный при подключении. С другой стороны, если вы установите SET NAMES 'latin1' или SET CHARACTER SET latin1, то непосредственно перед отправкой результата сервер будет преобразовывать значения, представленные в latin2, в latin1. Преобразование может привести к потерям, если в одном наборе окажутся символы, отсутствующие в другом наборе.

Если вы не хотите, чтобы сервер выполнял какие-то преобразования, установите character\_set\_results в NULL:

```
mysql> SET character_set_results = NULL;
```

### 3.3.7. Набор символов и порядок сопоставления строковых литералов

Каждый символьный строковый литерал имеет набор символов и порядок сопоставления, которые могут быть ненулевыми.



Строковый литерал может иметь необязательное представление набора символов и конструкцию COLLATE для указания порядка сопоставления:

```
{_имя_набора_символов}'строка' [COLLATE имя_порядка_сопоставления]
```

Примеры:

```
SELECT 'строка';  
SELECT _latin1'строка';  
SELECT _latin1'строка' COLLATE latin1_danish_ci;
```

Для отдельного оператора SELECT 'строка' аргумент строка имеет символьный набор и порядок сопоставления, определенный системными переменными character\_set\_connection и collation\_connection.

Выражение *имя\_набора\_символов* формально называют *“представителем”* (introducer). Он сообщает анализатору, что “строка, которая следует, имеет символьный набор X”. Поскольку это часто смущало пользователей в прошлом, мы подчеркиваем, что “представитель” сам по себе не выполняет никакого преобразования, а просто является строгим сигналом, который не меняет значения строки. “Представитель” также может проставляться перед стандартным шестнадцатеричным литералом и числовым представлением шестнадцатеричного литерала (x'литерал' или 0хnnnn), а также перед ? (местом подстановки параметров при использовании предварительно подготовленных операторов в интерфейсе языка программирования).

Ниже приведены примеры:

```
SELECT _latin1 x'AABBCC';  
SELECT _latin1 0xAABBCC;  
SELECT _latin1 ?;
```

MySQL определяет символьный набор и порядок сопоставления литералов следующим образом:

- Если указано и *\_X*, и COLLATE *Y*, применяется символьный набор *X* и порядок сопоставления *Y*.
- Если *\_X* указано, а COLLATE нет, используется набор *X* и его порядок сопоставления по умолчанию.
- В противном случае применяется набор символов и порядок сопоставления, заданные системными переменными character\_set\_connection и collation\_connection.

Примеры:

- Строка с набором символов latin1 и порядком сопоставления latin1\_german\_ci:  
SELECT \_latin1'Müller' COLLATE latin1\_german\_ci;
- Строка с набором символов latin1 и порядком сопоставления по умолчанию (то есть, latin1\_swedish\_ci):  
SELECT \_latin1'Müller';
- Строка с набором символов и порядком сопоставления по умолчанию:  
SELECT 'Müller';

“Представители” наборов символов и конструкция COLLATE реализованы в соответствии с требованиями стандарта SQL.

### 3.3.8. Применение COLLATE в операторах SQL

С помощью конструкции COLLATE можно переопределить любое поведение, связанное со сравнением строк. COLLATE может использоваться в различных частях SQL-операторов. Ниже представлены некоторые примеры:

- C ORDER BY:  

```
SELECT k  
FROM t1  
ORDER BY k COLLATE latin1_german2_ci;
```
- C AS:  

```
SELECT k COLLATE latin1_german2_ci AS k1  
FROM t1  
ORDER BY k1;
```
- C GROUP BY:  

```
SELECT k  
FROM t1  
GROUP BY k COLLATE latin1_german2_ci;
```
- С агрегатными функциями:  

```
SELECT MAX(k COLLATE latin1_german2_ci)  
FROM t1;
```
- C DISTINCT:  

```
SELECT DISTINCT k COLLATE latin1_german2_ci  
FROM t1;
```
- C WHERE:  

```
SELECT *  
FROM t1  
WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;
```
- C HAVING:  

```
SELECT k  
FROM t1  
GROUP BY k  
HAVING k = _latin1 'Müller' COLLATE latin1_german2_ci;
```

### 3.3.9. Приоритет конструкции COLLATE

Конструкция COLLATE имеет высокий приоритет (выше, чем ||), поэтому следующие два выражения эквивалентны:

```
x || y COLLATE z  
x || (y COLLATE z)
```

### 3.3.10. Операция BINARY

Операция BINARY – это сокращение для COLLATE. BINARY 'x' является эквивалентом для 'x' COLLATE y, где y – бинарный порядок сопоставления для символического набора 'x'. Каждый символический набор имеет бинарный порядок сопоставления. Например, бинарным порядком сопоставления для набора символов latin1 будет latin1\_bin, по-

этому для столбца `a` установлен символический набор `latin1`, и следующие два оператора производят одинаковый эффект:

```
SELECT * FROM t1 ORDER BY BINARY a;  
SELECT * FROM t1 ORDER BY a COLLATE latin1_bin;
```

### 3.3.11. Специальные случаи, в которых определение порядка сопоставления сложно

В огромном большинстве запросов вполне очевидно, какой порядок использует MySQL для выполнения операций сравнения. Например, в следующих примерах должно быть ясно, что порядок будет “порядком сопоставления, установленным для столбца `x`”:

```
SELECT x FROM T ORDER BY x;  
SELECT x FROM T WHERE x = x;  
SELECT DISTINCT x FROM T;
```

Однако когда в запросе присутствует несколько операндов, это может быть не столь однозначно, например:

```
SELECT x FROM T WHERE x = 'Y';
```

Должен ли запрос использовать порядок сопоставления столбца `x` или литеральной строки `'Y'`?

Стандарт SQL разрешает такие вопросы, используя то, что можно назвать правилами “принуждения” (coercibility), сущность которых состоит в следующем: поскольку и `x`, и `'Y'` имеют свои порядки сопоставления, чей порядок имеет приоритет? Это сложно, однако следующие правила разрешают большинство ситуаций:

- Явное указание конструкции `COLLATE` имеет степень принуждения 0 (то есть, принуждение не применяется).
- Конкатенация двух строк с разными порядками сопоставления имеет принуждение 1.
- Порядок сопоставления столбца имеет степень принуждения 2.
- Порядок сопоставления литерала имеет степень принуждения 3.

Приведенные ниже правила разрешают неоднозначность:

- Использовать порядок сопоставления с наименьшим значением степени принуждения.
- Если обе стороны имеют одинаковое значение степени принуждения, возникает ошибка, если используются разные символические наборы.

Примеры:

<code>column1 = 'A'</code>	Использовать порядок сопоставления <code>column1</code>
<code>column1 = 'A' COLLATE x</code>	Использовать порядок сопоставления <code>'A'</code>
<code>column1 COLLATE x = 'A' COLLATE y</code>	Ошибка

С помощью функции `COERCIBILITY()` можно определить степень принуждения строкового выражения:

```
mysql> SELECT COERCIBILITY('A' COLLATE latin1_swedish_ci);  
-> 0  
mysql> SELECT COERCIBILITY('A');  
-> 3
```

См. раздел 5.8.3.

### 3.3.12. Порядок сопоставления должен подходить набору символов

Вернемся к тому, что каждый символьный набор имеет соответствующий ему один или несколько порядков сопоставления. Поэтому следующий оператор вызовет сообщение об ошибке, так как `latin2_bin` не является допустимым порядком сопоставления для набора символов `latin1`:

```
mysql> SELECT _latin1 'x' COLLATE latin2_bin;
ERROR 1251: COLLATION 'latin2_bin' is not valid
for CHARACTER SET 'latin1'
```

В некоторых случаях выражения, которые работали до MySQL 4.1, терпят неудачу в версиях от 4.1 и выше, если не принимать во внимание набор символов и порядок сопоставления. Например, в версиях, предшествующих 4.1, следующий оператор работает нормально:

```
mysql> SELECT SUBSTRING_INDEX(USER(), '@', 1);
+-----+
| SUBSTRING_INDEX(USER(), '@', 1) |
+-----+
| root                             |
+-----+
```

После обновления до версии 4.1 он выдает сбой:

```
mysql> SELECT SUBSTRING_INDEX(USER(), '@', 1);
ERROR 1267 (HY000): Illegal mix of collations
(utf8_general_ci,IMPLICIT) and (latin1_swedish_ci,COERCIBLE)
for operation 'substr_index'
```

Причина состоит в том, что имена пользователей сохраняются в кодировке UTF8 (см. раздел 3.6). В результате функция `USER()` и литерал `'@'` имеют разные наборы символов (а потому и разные порядки сопоставления).

```
mysql> SELECT COLLATION(USER()), COLLATION('@');
+-----+-----+
| COLLATION(USER()) | COLLATION('@') |
+-----+-----+
| utf8_general_ci   | latin1_swedish_ci |
+-----+-----+
```

Один способ обойти это, состоит в том, чтобы заставить MySQL интерпретировать литеральную строку как `utf8`:

```
mysql> SELECT SUBSTRING_INDEX(USER(), _utf8'@', 1);
+-----+
| SUBSTRING_INDEX(USER(), _utf8'@', 1) |
+-----+
| root                             |
+-----+
```

Другой способ предполагает изменение набора символов и порядок сопоставления для соединения на `utf8`. Вы можете сделать это либо с помощью оператора `SET NAMES 'utf8'`, либо установкой значений системных переменных `character_set_connection` и `collation_connection` напрямую.

### 3.3.13. Пример эффекта от порядка сопоставления

Предположим, что столбец X таблицы T содержит следующие значения в наборе latin1:

Muffler  
Müller  
MX Systems  
MySQL

Предположим также, что значения столбца извлекаются с помощью такого оператора:

```
SELECT X FROM T ORDER BY X COLLATE имя_набора_сопоставления;
```

Результирующий список значений при различных порядках сопоставления представлен в табл. 3.1.

**Таблица 3.1.** Результирующие значения при различных порядках сопоставления

latin1_swedish_ci	latin1_german1_ci	latin1_german2_ci
Muffler	Muffler	Müller
MX Systems	Müller	Muffler
Müller	MX Systems	MX Systems
MySQL	MySQL	MySQL

Таблица 3.1 является примером того, какое влияние могут оказывать различные порядки сопоставления на конструкцию ORDER BY. Символ, из-за которого изменяется порядок сортировки, – это U с двумя точками сверху, который немцы называют “U-умляут”, а мы назовем “U-диерезис” (U-dieresis).

- Первый столбец показывает результат выполнения оператора SELECT при использовании правил шведско-финского порядка сопоставления, в котором указано, что U с двумя точками сортируется как Y.
- Второй столбец показывает результат выполнения оператора SELECT при использовании правил немецкого стандарта DIN-1, который указывает, что U с двумя точками сортируется как U.
- Третий столбец показывает результат выполнения оператора SELECT при использовании правил немецкого стандарта DIN-2, который указывает, что U с двумя точками сортируется как UE.

Три разных порядка сопоставления дают три разных результата. Как раз этим и управляет здесь MySQL. Применяя соответствующий порядок сопоставления, вы можете выбрать желаемый порядок сортировки.

## 3.4. Операции, на которые влияет поддержка наборов символов

В этом разделе описаны операции, которые принимают во внимание информацию о наборах символов, начиная с версии MySQL 4.1.

### 3.4.1. Результирующие строки

В MySQL предусмотрено множество операций и функций, которые возвращают строки. Этот раздел отвечает на вопрос: каковы наборы символов и порядки сопоставления для этих строк?

Для простых функций, которые принимают аргумент-строку и возвращают строку, выходной набор символов и порядок сопоставления – те же самые, что и у входного значения. Например, `UPPER(X)` возвращает строку, набор символов и порядок сопоставления которой те же, что и у `X`. То же касается функций `INSTR()`, `LCASE()`, `LOWER()`, `LTRIM()`, `MID()`, `REPEAT()`, `REPLACE()`, `REVERSE()`, `RIGHT()`, `RPAD()`, `RTRIM()`, `SOUNDEX()`, `SUBSTRING()`, `TRIM()`, `UCASE()` и `UPPER()`. (Также следует отметить, что функция `REPLACE()`, в отличие от прочих, игнорирует порядок сопоставления входных строк и всякий раз выполняет не зависящее от регистра сравнение.)

Для операций, которые комбинируют множество входных строк и возвращают единственную выходную строку, применяются “агрегатные правила” стандарта SQL:

- Если указано явно `COLLATION X`, использовать `X`.
- Если указано явно `COLLATION X` и `COLLATION Y`, генерировать ошибку.
- Иначе, если все порядки сопоставления `X`, использовать `X`.
- Иначе результат не имеет порядка сопоставления.

Например, для `CASE...WHEN a THEN b WHEN b THEN c COLLATE X END`, результирующим порядком сопоставления будет `X`. То же относится к `CASE`, `UNION`, `||`, `CONCAT()`, `ELT()`, `GREATEST()`, `IF()` и `LEAST()`.

Для операций, которые выполняют преобразование к символьным данным, набор символов и порядок сопоставления результирующих строк определяются системными переменными `character_set_connection` и `collation_connection`. Это касается функций `CAST()`, `CHAR()`, `CONV()`, `FORMAT()`, `HEX()` и `SPACE()`.

### 3.4.2. CONVERT()

`CONVERT()` представляет способ преобразования данных между различными символьными наборами. Синтаксис выглядит следующим образом:

```
CONVERT(выражение USING имя_кодировки)
```

В MySQL имена кодировок трансляции такие же, как соответствующие имена символьных наборов.

Примеры:

```
SELECT CONVERT(_latin1'Müller' USING utf8);
INSERT INTO utf8table (utf8column)
  SELECT CONVERT(latin1field USING utf8) FROM latin1table;
```

`CONVERT(...USING...)` реализована в соответствии со спецификациями стандарта SQL.

### 3.4.3. CAST()

Для преобразования строки в другой символьный набор можно также использовать функцию CAST(), синтаксис которой показан ниже:

```
CAST(символьная_строка AS символьный_тип_данных
      CHARACTER SET имя_набора_символов)
```

Пример:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8);
```

Если вы используете CAST() без CHARACTER SET, результирующий набор символов и порядок сопоставления определяются системными переменными character\_set\_connection и collation\_connection. Если же вы используете CAST() с CHARACTER SET X, результирующим набором символов будет X, а порядком сопоставления – выбранный по умолчанию для X.

Нельзя указывать конструкцию COLLATE внутри CAST(), но можно за пределами CAST(). То есть CAST(... COLLATE...) – неверно, тогда как CAST(...) COLLATE... – верно.

Пример:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8) COLLATE utf8_bin;
```

### 3.4.4. Операторы SHOW

Некоторые операторы SHOW в MySQL 4.1 добавлены, некоторые изменены для предоставления дополнительной информации о символьных наборах. Добавлены операторы SHOW CHARACTER SET, SHOW COLLATION и SHOW CREATE DATABASE. Операторы SHOW CREATE TABLE и SHOW COLUMNS модифицированы.

Команда SHOW CHARACTER SET отображает все доступные символьные наборы. Она предусматривает необязательную конструкцию LIKE, чтобы показывать только символьные наборы, соответствующие заданному шаблону, например:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+-----+-----+-----+-----+
| Charset | Description | Default collation | Maxlen |
+-----+-----+-----+-----+
| latin1  | ISO 8859-1 West European | latin1_swedish_ci | 1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci | 1 |
| latin5  | ISO 8859-9 Turkish | latin5_turkish_ci | 1 |
| latin7  | ISO 8859-13 Baltic | latin7_general_ci | 1 |
+-----+-----+-----+-----+
```

См. раздел 6.5.3.2.

Вывод SHOW COLLATION включает все допустимые символьные наборы. Здесь также предусмотрена конструкция LIKE для выбора порядков сортировки, имена которых совпадают с заданным шаблоном, например:

```
mysql> SHOW COLLATION LIKE 'latin%';
+-----+-----+-----+-----+-----+-----+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+-----+
| latin1_german1_ci | latin1 | 5 | | | 0 |
| latin1_swedish_ci | latin1 | 8 | Yes | Yes | 0 |
+-----+-----+-----+-----+-----+-----+
```

latin1_danish_ci	latin1	15			0	
latin1_german2_ci	latin1	31		Yes		2
latin1_bin	latin1	47		Yes		0
latin1_general_ci	latin1	48				0
latin1_general_cs	latin1	49				0
latin1_spanish_ci	latin1	94				0

См. раздел 6.5.3.3.

SHOW CREATE DATABASE отображает операторы CREATE DATABASE, которые создадут заданную базу данных. Результат включает все опции базы данных. Поддерживаются DEFAULT CHARACTER SET и COLLATE. Все опции базы сохраняются в текстовом файле db.opt, который можно найти в каталоге данных.

```
mysql> SHOW CREATE DATABASE a\G
***** 1. row *****
Database: a
Create Database: CREATE DATABASE `a`
/*!40100 DEFAULT CHARACTER SET macce */
```

См. раздел 6.5.3.5.

Оператор SHOW CREATE TABLE похож, но отображает оператор CREATE TABLE, который создает заданную таблицу. Определение столбца теперь показывает все спецификации символьных наборов, а опции таблицы – информацию о символьном наборе.

См. раздел 6.5.3.5.

Оператор SHOW FULL COLUMNS отображает порядки сопоставления столбцов таблицы, если его вызывать как SHOW FULL COLUMNS. Столбцы типов CHAR, VARCHAR или TEXT имеют ненулевые порядки сопоставления. Числовые и другие несимвольные типы имеют порядок сопоставления NULL. Например:

```
mysql> SHOW FULL COLUMNS FROM t;
+-----+-----+-----+-----+-----+-----+-----+
| Field | Type   | Collation | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+-----+
| a     | char(1) | latin1_bin | YES  |     | NULL    |       |
| b     | int(11) | NULL      | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+-----+
```

Имя набора символов не отображается (оно включено в имя порядка сопоставления).

См. раздел 6.5.3.3.

## 3.5. Поддержка Unicode

Начиная с версии MySQL 4.1, предусмотрено два новых символьных набора для хранения данных в кодировке Unicode:

- ucs2 – набор символов Unicode.
- utf8 – кодировка UTF8 набора символов Unicode.

В UCS-2 (бинарном представлении Unicode) каждый символ представлен двухбайтовым Unicode-кодом, с самым старшим байтом в начале. Например, "LATIN CAPITAL LETTER A" имеет код 0x0041, и хранится как последовательность байтов 0x00 0x41. "CYRILLIC SMALL LETTER YERU" (Unicode 0x044B) хранится как двухбайтовая по-



следовательность 0x04 0x4B. Информацию о символах Unicode и их кодах можно найти на домашней странице Unicode по адресу <http://www.unicode.org>.

Существует временное ограничение на использование UCS-2 в качестве клиентского символьного набора. Это значит, что SET NAMES 'ucs2' работать не будет.

Символьный набор UTF8 (трансформированное представление Unicode) – это альтернативный способ хранения данных в кодировке Unicode. Он реализован в соответствии с документом RFC2279. Идея символьного набора UTF8 состоит в том, что различные символы Unicode подгоняются к байтовым последовательностям различной длины.

- Основные латинские буквы, цифры и знаки пунктуации занимают один байт.
- Большинство букв европейских и средневековых языков занимают два байта: расширенные латинские, кириллические, греческие, армянские, сирийские, арабские, символы иврита и другие.
- Корейские, китайские и японские иероглифы помещаются в трехбайтовые последовательности.

В настоящее время поддержка MySQL набора UTF8 не включает четырехбайтовых последовательностей.

#### Совет

Чтобы сэкономить место, применяя UTF8, используйте тип VARCHAR вместо CHAR. В противном случае MySQL резервирует 30 байт для столбца CHAR(10) CHARACTER SET utf8, поскольку это максимально возможная длина.

## 3.6. UTF8 для метаданных

Метаданные – это данные о данных. Все, что описывает базу данных, в отличие от самого содержимого базы данных, является метаданными. То есть имена столбцов, имена баз данных, имена пользователей, имена версий и большинство строчковых результатов операторов SHOW являются метаданными.

Представление метаданных должно удовлетворять следующим требованиям:

- Все метаданные должны быть представлены в одном символьном наборе. В противном случае SHOW не будет правильно работать, поскольку разные строки в одном и том же столбце будут иметь разные символьные наборы.
- Метаданные должны включать все символы всех языков. Иначе пользователи не смогут называть столбцы и таблицы на их собственных языках.

С тем чтобы удовлетворить обоим требованиям, MySQL сохраняет метаданные в символьном наборе Unicode, называемом UTF8. Это не приводит ни к каким последствиям, если вы не используете диакритических знаков. Но если упомянутые знаки применяются, вам стоит помнить, что метаданные хранятся в наборе UTF8.

Это значит, что функции USER(), CURRENT\_USER() и VERSION() будут по умолчанию возвращать значения в наборе символов UTF8. То же относится и ко всем их синонимам, таким как синонимы функции USER() – SESSION\_USER() и SYSTEM\_USER().

Сервер устанавливает значение системной переменной character\_set\_system равным имени символьного набора метаданных:

```
mysql> SHOW VARIABLES LIKE 'character_set_system';
+-----+-----+
| Variable_name | Value |
```

```
+-----+-----+
| character_set_system | utf8 |
+-----+-----+
```

Сохранение метаданных в Unicode не означает, что заголовки столбцов и результаты функций DESCRIBE будут по умолчанию в символьном наборе `character_set_system`. Когда вы выдадите `SELECT column1 FROM t`, имя `column1` само по себе будет возвращено с сервера клиенту в том символьном наборе, который определен оператором SET NAMES. Более точно используемый набор символов определяется значением системной переменной `character_set_results`. Если эта переменная установлена в NULL, никакого преобразования не происходит и сервер возвращает метаданные, используя оригинальный набор символов (заданный в `character_set_system`).

Если вы не хотите, чтобы сервер присылал метаданные обратно в символьном наборе, отличном от UTF8, применяйте SET NAMES для принудительного преобразования на сервере (см. раздел 3.3.6), либо выполняйте преобразование на клиенте. Всегда более эффективно переносить преобразование на сторону клиента, однако этот выбор не доступен для многих клиентов вплоть до серии продуктов MySQL 4.x.

Если вы используете, например, только функцию USER() для сравнения или присваивания внутри отдельного оператора, то беспокоиться не следует. MySQL выполнит некоторое автоматическое преобразование сам.

```
SELECT * FROM Table1 WHERE USER() = latin1_column;
```

Это работает потому, что значение `latin1_column` перед сравнением автоматически преобразуется в UTF8.

```
INSERT INTO Table1 (latin1_column) SELECT USER();
```

Это работает по той причине, что значение USER() перед присваиванием автоматически преобразуется в latin1. Автоматическое преобразование пока не полностью реализовано, но будет работать правильно в более поздних версиях.

Несмотря на то что автоматическое преобразование не включено в стандарт SQL, документация по стандарту SQL утверждает, что каждый символьный набор является (в смысле поддерживаемых символов) “подмножеством” Unicode. Принимая во внимание известный принцип, который состоит в том, что “все, определенное для супермножества, касается и подмножества”, мы уверены, что порядок сопоставления для Unicode может применяться для сравнений строк в кодировке, отличной от Unicode.

## 3.7. Совместимость с другими системами управления базами данных

Для достижения совместимости с MaxDB следующие два оператора эквивалентны:

```
CREATE TABLE t1 (f1 CHAR(n) UNICODE);
CREATE TABLE t1 (f1 CHAR(n) CHARACTER SET ucs2);
```

## 3.8. Новый формат файлов определения символьных наборов

В MySQL 4.1 конфигурация символьных наборов хранится в XML-файлах, по одному на каждый символьный набор. В предыдущих версиях эта информация хранилась в файлах `.conf`.

## 3.9. Национальный набор символов

До MySQL 4.1 типы `NCHAR` и `CHAR` были синонимами. Стандарт ANSI определяет `NCHAR` или `NATIONAL CHAR` как способ указания, что столбец `CHAR` должен использовать некоторый предопределенный символьный набор. Версия MySQL 4.1 и выше использует `utf8` в качестве этого предопределенного символьного набора. Например, следующие объявления типов столбцов эквивалентны:

```
CHAR(10) CHARACTER SET utf8
NATIONAL CHARACTER(10)
NCHAR(10)
```

Эквивалентны также и следующие объявления:

```
VARCHAR(10) CHARACTER SET utf8
NATIONAL VARCHAR(10)
NCHAR VARCHAR(10)
NATIONAL CHARACTER VARYING(10)
NATIONAL CHAR VARYING(10)
```

Вы можете использовать `N` литерал для создания строки в национальном символьном наборе. Следующие два оператора эквивалентны:

```
SELECT N'некоторый текст';
SELECT _utf8'некоторый текст';
```

## 3.10. Обновление символьных наборов от версии MySQL 4.0

Итак, что же насчет обновления более старых версий MySQL? MySQL 4.1 почти совместим сверху вниз с MySQL 4.0 и более ранними версиями по той простой причине, что практически все эти средства являются новыми, а поэтому нечему конфликтовать со старыми версиями. Однако имеются некоторые отличия и несколько моментов, которые следует принимать во внимание.

Наиболее важный момент: “набор символов MySQL 4.0” включал в себя свойства и “набора символов MySQL 4.1”, и “порядка сопоставления MySQL 4.1”. Вам придется забыть об этом. Отныне мы не связываем символьный набор и порядок сопоставления в один смешанный объект.

Национальные символьные наборы трактуются в MySQL 4.1 специальным образом. `NCHAR` — это не то же самое, что `CHAR`, а литералы `N'...'` — не то же самое, что литералы `'...'`.

И, наконец, изменился формат файлов, хранящих информацию о символьных наборах и порядках сопоставления. Убедитесь, что вы переустановили каталог `/share/mysql/charsets`, в котором находятся новые конфигурационные файлы.

Если вы хотите запустить `mysqld` из дистрибутива 4.1.x с данными, созданными в MySQL 4.0, то вы должны стартовать сервер с тем же набором символов и порядком сопоставления. В этом случае не придется выполнять переиндексацию ваших данных.

Есть два способа сделать это:

```
shell> ./configure --with-charset=... --with-collation=...
shell> ./mysqld --default-character-set=... --default-collation=...
```

Если вам нужен `mysqld`, например, с символьным набором MySQL 4.0 `danish`, то вы должны теперь использовать символьный набор `latin1` и порядок сопоставления `latin1_danish_ci`:

```
shell> ./configure --with-charset=latin1 \
--with-collation=latin1_danish_ci
shell> ./mysqld --default-character-set=latin1 \
--default-collation=latin1_danish_ci
```

Используйте табл. 3.2, приведенную в разделе 3.10.1, чтобы найти старые имена наборов символов MySQL 4.0 и их соответствие новым парам набор/порядок.

Если в таблице `latin1` версии 4.0 у вас хранятся данные не в наборе `latin1`, и вы хотите преобразовать определения столбцов таблицы так, чтобы они отражали действительный символьный набор данных, обратитесь к инструкциям, представленным в разделе 3.10.2.

### 3.10.1. Символьные наборы и соответствующие пары “символьный набор/порядок сопоставления” версии 4.1

Таблица 3.2. Символьные наборы и соответствующие пары  
“символьный набор/порядок сопоставления” версии 4.1

ID	Набор символов 4.0	Набор символов 4.1	Порядок сопоставления 4.1
1	big5	big5	big5_chinese_ci
2	czech	latin2	latin2_czech_ci
3	dec8	dec8	dec8_swedish_ci
4	dos	cp850	cp850_general_ci
5	german1	latin1	latin1_german1_ci
6	hp8	hp8	hp8_english_ci
7	koi8_ru	koi8r	koi8r_general_ci
8	latin1	latin1	latin1_swedish_ci
9	latin2	latin2	latin2_general_ci
10	swe7	swe7	swe7_swedish_ci
11	usa7	ascii	ascii_general_ci
12	ujis	ujis	ujis_japanese_ci
13	sjis	sjis	sjis_japanese_ci
14	cp1251	cp1251	cp1251_bulgarian_ci
15	danish	latin1	latin1_danish_ci

Окончание табл. 3.2

ID	Набор символов 4.0	Набор символов 4.1	Порядок сопоставления 4.1
16	hebrew	hebrew	hebrew_general_ci
17	win1251	(изъят)	(изъят)
18	tis620	tis620	tis620_thai_ci
19	euc_kr	euckr	euckr_korean_ci
20	estonia	latin7	latin7_estonian_ci
21	hungarian	latin2	latin2_hungarian_ci
22	koi8_ukr	koi8u	koi8u_ukrainian_ci
23	win1251ukr	cp1251	cp1251_ukrainian_ci
24	gb2312	gb2312	gb2312_chinese_ci
25	greek	greek	greek_general_ci
26	win1250	cp1250	cp1250_general_ci
27	croat	latin2	latin2_croatian_ci
28	gbk	gbk	gbk_chinese_ci
29	cp1257	cp1257	cp1257_lithuanian_ci
30	latin5	latin5	latin5_turkish_ci
31	latin1_de	latin1	latin1_german2_ci

### 3.10.2. Преобразование символьных столбцов версии 4.0. в формат версии 4.1

Обычно сервер при запуске по умолчанию использует символьный набор latin1. Если у вас данные столбцы хранятся в другом символьном наборе, который сервер 4.1 теперь поддерживает непосредственно, вы можете преобразовать такие столбцы. Однако вы должны избегать прямого преобразования latin1 в “реальный” символьный набор. Это может привести к потере данных. Вместо этого преобразуйте столбцы сначала в бинарный тип, а затем из бинарного в небинарный с требуемым символьным набором. Преобразование в бинарный тип и из бинарного типа не пытается преобразовывать значения символов и предохраняет ваши данные. Например, предположим, что у вас есть таблица из версии 4.0 с тремя столбцами, которые применяются для хранения значений в latin1, latin2 и utf8:

```
CREATE TABLE t
(
    latin1_col CHAR(50),
    latin2_col CHAR(100),
    utf8_col CHAR(150)
);
```

После обновления сервера до версии MySQL 4.1, вы хотите преобразовать таблицу так, чтобы оставить без изменений столбец latin1\_col, но изменить столбцы latin2\_col и utf8\_col так, чтобы они имели наборы символов latin2 и utf8.

Сначала создайте резервную копию таблицы, а затем преобразуйте столбцы следующим образом:

```
ALTER TABLE t MODIFY latin2_col BINARY(100);
ALTER TABLE t MODIFY utf8_col BINARY(150);
ALTER TABLE t MODIFY latin2_col CHAR(100) CHARACTER SET latin2;
ALTER TABLE t MODIFY utf8_col CHAR(150) CHARACTER SET utf8;
```

Первые два оператора “убирают” информацию о наборе символов из столбцов `latin2_col` и `utf8_col`. Вторые два оператора назначают им правильные символьные наборы.

Если хотите, можете скомбинировать преобразование в бинарный набор и обратно в объединенные операторы:

```
ALTER TABLE t
  MODIFY latin2_col BINARY(100),
  MODIFY utf8_col BINARY(150);
ALTER TABLE t
  MODIFY latin2_col CHAR(100) CHARACTER SET latin2,
  MODIFY utf8_col CHAR(150) CHARACTER SET utf8;
```

## 3.11. Наборы символов и порядки сопоставления, которые поддерживает MySQL 4.1

Ниже приведен аннотированный список наборов символов и порядков сопоставления, которые поддерживает MySQL. Поскольку опции и параметры установки отличаются, некоторые сервера могут иметь не все из перечисленных позиций, а некоторые могут иметь позиции, которые здесь не приведены.

MySQL поддерживает более 70 порядков сопоставления и более 30 наборов символов. Символьные наборы и их порядки сопоставления по умолчанию отображаются оператором `SHOW CHARACTER SET` (вывод включает и другие столбцы, не показанные здесь).

```
mysql> SHOW CHARACTER SET;
```

Charset	Description	Default collation
big5	Big5 Traditional Chinese	big5_chinese_ci
dec8	DEC West European	dec8_swedish_ci
cp850	DOS West European	cp850_general_ci
hp8	HP West European	hp8_english_ci
koi8r	KOI8-R Relcom Russian	koi8r_general_ci
latin1	ISO 8859-1 West European	latin1_swedish_ci
latin2	ISO 8859-2 Central European	latin2_general_ci
swe7	7bit Swedish	swe7_swedish_ci
ascii	US ASCII	ascii_general_ci
ujis	EUC-JP Japanese	ujis_japanese_ci
sjis	Shift-JIS Japanese	sjis_japanese_ci
cp1251	Windows Cyrillic	cp1251_bulgarian_ci
hebrew	ISO 8859-8 Hebrew	hebrew_general_ci
tis620	TIS620 Thai	tis620_thai_ci
euckr	EUC-KR Korean	euckr_korean_ci

koi8u	KOI8-U Ukrainian	koi8u_general_ci	
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci	
greek	ISO 8859-7 Greek	greek_general_ci	
cp1250	Windows Central European	cp1250_general_ci	
gbk	GBK Simplified Chinese	gbk_chinese_ci	
latin5	ISO 8859-9 Turkish	latin5_turkish_ci	
armSCII8	ARMSCII-8 Armenian	armSCII8_general_ci	
utf8	UTF-8 Unicode	utf8_general_ci	
ucs2	UCS-2 Unicode	ucs2_general_ci	
cp866	DOS Russian	cp866_general_ci	
keybcs2	DOS Kamenicky Czech-Slovak	keybcs2_general_ci	
macce	Mac Central European	macce_general_ci	
macroman	Mac West European	macroman_general_ci	
cp852	DOS Central European	cp852_general_ci	
latin7	ISO 8859-13 Baltic	latin7_general_ci	
cp1256	Windows Arabic	cp1256_general_ci	
cp1257	Windows Baltic	cp1257_general_ci	
binary	Binary pseudo charset	binary	
geostd8	GEOSTD8 Georgian	geostd8_general_ci	

### 3.11.1. Символьные наборы Unicode

MySQL поддерживает два набора символов Unicode. С использованием этих символьных наборов можно сохранять тексты примерно на 650 языках. Мы пока не добавляли большое количество порядков сопоставления для этих двух наборов, но это случится скоро. В настоящее время они имеют не зависящие от регистра, нечувствительные к диакритическим знакам порядки сопоставления, плюс бинарный порядок.

На данный момент порядок сопоставления `ucs2_general_uca` реализует только частичную поддержку алгоритма Unicode Collation Algorithm. Некоторые символы пока не поддерживаются.

■ Порядки сопоставления для `ucs2` (UCS2 Unicode):

- `ucs2_bin`
- `ucs2_general_ci` (по умолчанию)
- `ucs2_general_uca`

■ Порядки сопоставления для `utf8` (UTF8 Unicode):

- `utf8_bin`
- `utf8_general_ci` (по умолчанию)

### 3.11.2. Западноевропейские наборы символов

Западноевропейские наборы символов покрывают большинство западноевропейских языков, среди которых французский, испанский, каталонский, баскский, португальский, итальянский, албанский, датский, немецкий, шведский, норвежский, финский, фарерский, исландский, ирландский, шотландский и английский.

■ `ascii` (ASCII США):

- `ascii_bin`
- `ascii_general_ci` (по умолчанию)

- **cp850** (западноевропейские для DOS):
  - `cp850_bin`
  - `cp850_general_ci` (по умолчанию)
- **dec8** (западноевропейские для компьютеров DEC):
  - `dec8_bin`
  - `dec8_swedish_ci` (по умолчанию):
- **hp8** (западноевропейские для компьютеров HP):
  - `hp8_bin`
  - `hp8_english_ci` (по умолчанию)
- **latin1** (западноевропейские по стандарту ISO 8859-1):
  - `latin1_bin`
  - `latin1_danish_ci`
  - `latin1_general_ci`
  - `latin1_general_cs`
  - `latin1_german1_ci`
  - `latin1_german2_ci`
  - `latin1_spanish_ci`
  - `latin1_swedish_ci` (по умолчанию)

`latin1` является набором символов по умолчанию. Порядок сопоставления `latin1_swedish_ci` – порядок по умолчанию, который, предположительно, использует большинство клиентов MySQL. Постоянно утверждается, что он основан на шведско-финских правилах сравнения, но вы наверняка найдете немало шведов и финнов, которые с этим не согласны.

`latin1_german1_ci` и `latin1_german2_ci` базируются на стандартах DIN-1 и DIN-2. DIN (Немецкий Институт стандартов) – это немецкий аналог ANSI. DIN-1 называют порядком словарей, а DIN-2 – порядком телефонных книг.

- Правила `latin1_german1_ci` (словарь):  
 'Ä' = 'A', 'Ö' = 'O', 'Ü' = 'U', 'ß' = 's'
- Правила `latin1_german2_ci` (телефонная книга):  
 'Ä' = 'AE', 'Ö' = 'OE', 'Ü' = 'UE', 'ß' = 'ss'

В порядке сопоставления `latin1_spanish_ci` 'Ñ' ('N' с тильдой) – отдельная буква между 'N' и 'O'.

- **macroman** (западноевропейские для компьютеров Mac):
  - `macroman_bin`
  - `macroman_general_ci` (по умолчанию)
- **swe7** (7-битные шведские):
  - `swe7_bin`
  - `swe7_swedish_ci` (по умолчанию).



### 3.11.3. Центральноевропейские наборы символов

Мы предоставляем некоторую поддержку символьных наборов, используемых в Чехии, Словакии, Венгрии, Румынии, Словении, Хорватии и Польше.

- **cp1250 (Windows, центральная Европа):**
  - `cp1250_bin`
  - `cp1250_czech_ci`
  - `cp1250_general_ci` (по умолчанию)
- **cp852 (DOS, центральная Европа):**
  - `cp852_bin`
  - `cp852_general_ci` (по умолчанию)
- **keybcs2 (DOS Kamenicky, Чехословакия):**
  - `keybcs2_bin`
  - `keybcs2_general_ci` (по умолчанию)
- **latin2 (Центрально-европейские по стандарту ISO 8859-2):**
  - `latin2_bin`
  - `latin2_croatian_ci`
  - `latin2_czech_ci`
  - `latin2_general_ci` (по умолчанию)
  - `latin2_hungarian_ci`
- **macce (Центрально-европейские, для компьютеров Mac):**
  - `macce_bin`
  - `macce_general_ci` (по умолчанию).

### 3.11.4. Южно-европейские и средневосточные наборы символов

- **armscii8 (ARMScii-8, армянский):**
  - `armscii8_bin`
  - `armscii8_general_ci` (по умолчанию)
- **cp1256 (Windows, арабский):**
  - `cp1256_bin`
  - `cp1256_general_ci` (по умолчанию)
- **geostd8 (GEOSTD8, грузинский):**
  - `geostd8_bin`
  - `geostd8_general_ci` (по умолчанию)
- **greek (ISO 8859-7, греческий):**
  - `greek_bin`
  - `greek_general_ci` (по умолчанию)

- **hebrew (ISO 8859-8, иврит):**
  - `hebrew_bin`
  - `hebrew_general_ci` (по умолчанию)
- **latin5 (ISO 8859-9, турецкий):**
  - `latin5_bin`
  - `latin5_turkish_ci` (по умолчанию).

### 3.11.5. Балтийские наборы символов

Балтийские наборы символов покрывают эстонский, литовский и латышский языки. В настоящее время поддерживаются два балтийских набора символов:

- **cp1257 (Windows, Балтия):**
  - `cp1257_bin`
  - `cp1257_general_ci` (по умолчанию)
  - `cp1257_lithuanian_ci`
- **latin7 (ISO 8859-13, Балтия):**
  - `latin7_bin`
  - `latin7_estonian_cs`
  - `latin7_general_ci` (по умолчанию)
  - `latin7_general_cs`

### 3.11.6. Кириллические наборы символов

Ниже представлены кириллические наборы символов для использования с белорусским, болгарским, русским и украинским языками:

- **cp1251 (Windows, кириллица):**
  - `cp1251_bin`
  - `cp1251_bulgarian_ci`
  - `cp1251_general_ci` (по умолчанию)
  - `cp1251_general_cs`
  - `cp1251_ukrainian_ci`
- **cp866 (DOS, русский):**
  - `cp866_bin`
  - `cp866_general_ci` (по умолчанию)
- **koi8r (KOI8-R, русский, Релком):**
  - `koi8r_bin`
  - `koi8r_general_ci` (по умолчанию)
- **koi8u (KOI8-U, украинский):**
  - `koi8u_bin`
  - `koi8u_general_ci` (по умолчанию).

### 3.11.7. Азиатские наборы символов

В состав азиатских наборов символов, которые мы поддерживаем, входят китайский, японский, корейский и тайский. Они могут быть сложными. Например, китайские наборы должны включать тысячи различных символов.

- **big5** (Big5, китайский традиционный):
  - `big5_bin`
  - `big5_chinese_ci` (по умолчанию)
- **euckr** (EUC-KR, корейский):
  - `euckr_bin`
  - `euckr_korean_ci` (по умолчанию)
- **gb2312** (GB2312, китайский упрощенный):
  - `gb2312_bin`
  - `gb2312_chinese_ci` (по умолчанию)
- **gbk** (GBK, китайский упрощенный):
  - `gbk_bin`
  - `gbk_chinese_ci` (по умолчанию)
- **sjis** (Shift-JIS японский):
  - `sjis_bin`
  - `sjis_japanese_ci` (по умолчанию)
- **tis620** (TIS620, тайский):
  - `tis620_bin`
  - `tis620_thai_ci` (по умолчанию)
- **ujis** (EUC-JP, японский):
  - `ujis_bin`
  - `ujis_japanese_ci` (по умолчанию).

## Типы столбцов

**В** MySQL поддерживается множество типов столбцов различных категорий: числовые типы, типы даты и времени и строковые (символьные) типы. В настоящей главе приведен общий обзор этих типов столбцов, затем дается более детальное описание свойств типов в каждой категории, а также итоговая информация об их требованиях к системе хранения. Представленный обзор типов является поверхностным. Для получения дополнительной информации о конкретном типе, такой как допустимые форматы значений столбцов, следует ознакомиться с его детальным описанием.

MySQL версии 4.1 и выше поддерживает расширения для управления пространственными данными. Информация о пространственных (spatial) типах приведена в главе 7.

При описании типов используются следующие соглашения:

- *M*. Обозначает максимальный показываемый размер. Максимально допустимый составляет 255.
- *D*. Используется в описании числовых типов с фиксированной и плавающей запятой и обозначает количество знаков после запятой. Максимально возможное значение равно 30, но должно быть не более чем  $M - 2$ .
- Квадратные скобки ('[' и ']') выделяют необязательные части в спецификации типов.

### 4.1. Обзор типов столбцов

#### 4.1.1. Обзор числовых типов

Ниже приведена общая информация о числовых типах столбцов. Дополнительную информацию можно найти в разделе 4.2. Требования по хранению столбцов приведены в разделе 4.5.

Если вы указываете атрибут `ZEROFILL` для числового столбца, MySQL автоматически добавит атрибут `UNSIGNED`.

#### Внимание!

Следует помнить, что когда выполняется вычитание между двумя целыми, одно из которых без знака, то результат тоже будет без знака! См. раздел 5.7.

- `TINYINT[(M)] [UNSIGNED] [ZEROFILL]`

Очень короткое целое. Диапазон для знаковых значений: от -128 до 127. Для беззнаковых — от 0 до 255.

- BIT, BOOL, BOOLEAN

Синоним для TINYINT(1). Синоним BOOLEAN появился в версии MySQL 4.1.0. Нулевое значение рассматривается, как “ложь”, а ненулевое – как “истина”.

В будущем планируется ввести полную поддержку булевских типов в соответствии со стандартом SQL.

- SMALLINT[(M)] [UNSIGNED] [ZEROFILL]

Короткое целое. Диапазон для знаковых значений: от -32768 до 32767, а для беззнаковых – от 0 до 65535.

- MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]

Целое среднего размера. Диапазон знаковых значений: от -8388608 до 8388607. Для беззнаковых – от 0 до 16777215.

- INT[(M)] [UNSIGNED] [ZEROFILL]

Обычное целое. Диапазон знаковых значений: от -2147483648 до 2147483647. Для беззнаковых – от 0 до 4294967295.

- INTEGER[(M)] [UNSIGNED] [ZEROFILL]

Синоним для INT.

- BIGINT[(M)] [UNSIGNED] [ZEROFILL]

Длинное целое. Диапазон знаковых значений: от -9223372036854775808 до 9223372036854775807. Для беззнаковых – от 0 до 18446744073709551615.

Ниже представлены некоторые вещи, о которых следует знать при обращении с столбцами BIGINT:

- Все арифметические действия выполняются с использованием значений BIGINT или DOUBLE, поэтому вы не должны применять беззнаковых целых, больших чем 9223372036854775807 (63 бита), кроме как с битовыми функциями! Если это сделать, некоторые их последних разрядов результата могут оказаться неверными из-за ошибок округления при преобразовании BIGINT в DOUBLE.

- MySQL 4.1 может работать с BIGINT в следующих случаях:

При сохранении больших беззнаковых значений в столбцах типа BIGINT.

В функциях MIN(имя\_столбца) или MAX(имя\_столбца), когда имя\_столбца ссылается на столбец типа BIGINT.

При использовании операций (+, -, \* и так далее), когда оба операнда являются целыми.

- Вы всегда можете хранить точное целое значение столбца BIGINT, сохраняя его в виде строки. В этом случае MySQL выполняет преобразование строки в число, которое не использует промежуточного представления в виде числа двойной точности.
- Операции -, + и \* используют арифметику BIGINT, когда оба операнда представляют собой целые значения! Это означает, что если вы перемножаете два больших целых числа (или результата функций, возвращающих целые), то можете получить неожиданный результат, если он превысит значение 9223372036854775807.

■ `FLOAT(p) [UNSIGNED] [ZEROFILL]`

Число с плавающей точкой. Здесь *p* задает точность, которая может быть от 0 до 24 для числа с плавающей точкой одинарной точности, и от 25 до 53 для числа с плавающей точкой двойной точности. Эти типы похожи на `FLOAT` и `DOUBLE`, которые описываются далее. `FLOAT(p)` имеет те же диапазоны допустимых значений, что и соответствующие типы `FLOAT` и `DOUBLE`, но ширина отображения и количество разрядов не определено. Начиная с версии MySQL 3.23, это полноценный тип с плавающей точкой. В более ранних версиях `FLOAT(p)` всегда имел 2 разряда.

Этот синтаксис предусмотрен для совместимости с ODBC.

Применение `FLOAT` может привести к некоторым неожиданным проблемам, поскольку все вычисления в MySQL выполняются с двойной точностью. См. раздел A.1.7.

■ `FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]`

Короткий (одинарной точности) тип числа с плавающей точкой. Допустимые значения – от  $-3,402823466E+38$  до  $-1,175494351E-38$ , 0 и от  $1,175494351E-38$  до  $3,402823466E+38$ . Если указано `UNSIGNED`, отрицательные значения не допускаются. *M* – отображаемая ширина и *D* – количество цифр после десятичной точки. `FLOAT` без аргументов, или `FLOAT(p)` (при *p* от 0 до 24), представляют числа с плавающей точкой одинарной точности.

■ `DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]`

Тип числа с плавающей точкой нормальной длины (двойной точности). Допустимые значения – от  $-1,7976931348623157E+308$  до  $-2,2250738585072014E-308$ , 0 и от  $2,2250738585072014E-308$  до  $1,7976931348623157E+308$ . Если указано `UNSIGNED`, отрицательные значения не допускаются. *M* – отображаемая ширина и *D* – количество цифр после десятичной точки. `DOUBLE` без аргументов, или `DOUBLE(p)` (при *p* от 25 до 53), представляют числа с плавающей точкой двойной точности.

■ `DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL]`

`REAL[(M,D)] [UNSIGNED] [ZEROFILL]`

Это синонимы для `DOUBLE`. Исключение: если SQL-режим сервера включает опцию `REAL_AS_FLOAT`, `REAL` становится синонимом `FLOAT`, а не `DOUBLE`.

■ `DECIMAL[(M,D)] [UNSIGNED] [ZEROFILL]`

Неупакованное число с фиксированной точкой. Ведет себя подобно столбцу `CHAR`. “Неупакованное” означает, что число хранится в виде строки, используя один символ на каждый десятичный разряд. *M* – общее количество разрядов и *D* – количество разрядов после десятичной точки. Десятичная точка и знак ‘-’ (для отрицательных значений) не считаются в *M*, хотя место для них резервируется. Если *D* равно 0, значение не содержит десятичной точки или дробной части. Максимальный диапазон значений `DECIMAL` такой же, как и `DOUBLE`, но действительный диапазон для конкретного столбца `DECIMAL` ограничивается заданными *M* и *D*. Если указано `UNSIGNED`, отрицательные значения не допускаются. Если *D* не указано, принимается по умолчанию 0. Если *M* пропущено, по умолчанию принимается 10. До MySQL 3.23 аргумент *M* должен был быть достаточным, чтобы вместить десятичную точку и знак.

- DEC({M[,D]}) [UNSIGNED] [ZEROFILL]  
NUMERIC({M[,D]}) [UNSIGNED] [ZEROFILL]  
FIXED({M[,D]}) [UNSIGNED] [ZEROFILL]

Синонимы для DECIMAL. Синоним FIXED был добавлен в версии MySQL 4.1.0 для совместимости с другими серверами.

## 4.1.2. Обзор типов даты и времени

Ниже приведена общая информация о типах столбцов, предусмотренных для хранения значений времени и даты. Дополнительную информацию можно найти в разделе 4.3. Требования по хранению столбцов представлены в разделе 4.5.

- DATE. Дата. Поддерживаемый диапазон — от '1000-01-01' до '9999-12-31'. MySQL выводит значения типа DATE в формате 'ГГГГ-ММ-ДД', но позволяет присваивать значения столбцам типа DATE как в виде строк, так и в форме чисел.
- DATETIME. Комбинация даты и времени. Поддерживаемый диапазон — от '1000-01-01 00:00:00' до '9999-12-31 23:59:59'. MySQL выводит значения типа DATETIME в формате 'ГГГГ-ММ-ДД ЧЧ:ММ:СС', но позволяет присваивать значения столбцам типа DATETIME как в виде строк, так и в форме чисел.
- TIMESTAMP(M). Временная метка. Диапазон — от '1970-01-01 00:00:00' до части 2037 года. Столбцы TIMESTAMP применимы для регистрации времени и даты операций INSERT и UPDATE. Значение первого столбца типа TIMESTAMP в таблице автоматически устанавливается равным времени и дате самой последней операции, если только вы не присваиваете его явно. Вы можете также установить значение текущего времени и даты в любом столбце типа TIMESTAMP, присвоив им NULL.

Начиная с MySQL 4.1 и выше, TIMESTAMP возвращается в виде строки в формате 'ГГГГ-ММ-ДД ЧЧ:ММ:СС'. Если вы хотите получить значение в виде числа, то должны добавить 0 к столбцу TIMESTAMP. Разная ширина отображения значений временной метки не поддерживается.

В MySQL 4.0 и более ранних версиях значения TIMESTAMP отображаются в форматах ГГГГММДДЧЧММСС, ГГММДДЧЧММСС, ГГГГММДД или ГГММДД, в зависимости от того, равен ли параметр M значению 14 (или не указан), 12, 8 или 6, но при этом присваивать значения столбцам TIMESTAMP можно и в виде строки и в виде числа. Параметр M влияет только на то, как значение отображается, а не хранится. Хранятся же значения TIMESTAMP всегда в четырех байтах. В MySQL 4.0.12, предусмотрена опция сервера --new, которая заставляет его вести себя так, как MySQL 4.1.

Следует отметить, что столбцы TIMESTAMP(M), где M равно 8 или 14, описываются как числовые, в то время как при других значениях они описываются строками. Это сделано только для гарантии правильной выгрузки в дампы и последующей загрузки таблиц со столбцами такого типа.

- TIME. Время. Диапазон — от '-838:59:59' до '838:59:59'. MySQL показывает значения времени в формате 'ЧЧ:ММ:СС', но позволяет присваивать значения столбцам типа TIME в числовом и строковом виде.
- YEAR({2|4}). Год в двузначном или четырехзначном формате. По умолчанию используется четырехзначный. В четырехзначном формате допустимы значения от 1901 до 2155, и 0000. В двузначном формате допустимы значения от 1970 до 2069.

MySQL отображает значения YEAR в формате YYYY, однако позволяет присваивать столбцам типа YEAR и числовые и строковые значения.

### 4.1.3. Обзор строковых типов

Ниже приведена общая информация о типах столбцов, предусмотренных для хранения строковых значений. Дополнительную информацию можно найти в разделе 4.4. Требования по хранению столбцов представлены в разделе 4.5.

В некоторых случаях MySQL может заменить строковый столбец типом, отличающимся от указанного в операторе CREATE TABLE или ALTER TABLE (см. раздел 6.2.5.2).

Изменение, которое касается многих строковых типов столбцов, состоит в том, что, начиная с MySQL 4.1, определение символьных столбцов может включать атрибут CHARACTER SET для указания символьного набора и, необязательно, порядка сопоставления. Это касается типов CHAR, VARCHAR, TEXT, ENUM и SET. Например:

```
CREATE TABLE t
(
  c1 CHAR(20) CHARACTER SET utf8,
  c2 CHAR(20) CHARACTER SET latin1 COLLATE latin1_bin
);
```

Показанное выше определение таблицы создает столбец C1, который имеет набор символов utf8 с порядком сопоставления по умолчанию для этого символьного набора, и столбец C2 с набором символов latin1 и бинарным порядком сопоставления. Бинарное сопоставление нечувствительно к регистру.

Сортировка и порядок сравнения значений символьных столбцов базируется на символьном наборе, назначенном столбцу. До MySQL 4.1 сортировка и сравнение основывались на порядке сопоставления, заданном набором символов сервера. Для столбцов CHAR и VARCHAR вы можете объявить атрибут BINARY, чтобы сделать порядок сопоставления и сравнения независимым от регистра, с использованием значений кодов символов вместо лексического.

Более подробную информацию можно найти в главе 3.

Кроме того, начиная с версии MySQL 4.1, спецификация длины строковых столбцов указывается в символах (в ранних версиях она интерпретировалась в байтах).

- [NATIONAL] CHAR(M) [BINARY | ASCII | UNICODE]

Строка фиксированной длины, всегда при сохранении дополняемая справа пробелами до указанной длины. M представляет длину столбца. Диапазон значений M – от 0 до 255 символов (до версии MySQL 3.23 диапазон включал от 1 до 255).

#### На заметку!

Завершающие пробелы удаляются при извлечении значений CHAR.

Начиная с MySQL 4.1.0, столбцы CHAR длиной свыше 255 символов преобразуются к наименьшему типу TEXT, который может вместить значения указанной длины. Например, CHAR(500) преобразуется в TEXT, а CHAR(200000) – в MEDIUMTEXT. Данная возможность обеспечивает совместимость. Однако это преобразование делает тип столбца переменной длины, а также приводит к удалению завершающих пробелов.

CHAR представляет собой сокращение от CHARACTER. NATIONAL CHAR (или его эквивалент в сокращенной форме, NCHAR) – это стандартный способ, с помощью кото-



рого в языке SQL можно определить, что столбец должен использовать набор символов по умолчанию. Это установлено в MySQL по умолчанию.

Атрибут `BINARY` делает сортировку и сравнения строк независимыми от регистра.

Начиная с MySQL 4.1.0, можно указывать атрибут `ASCII`. Он назначает столбцу `CHAR` набор символов `latin1`.

Начиная с MySQL 4.1.0, можно также указывать атрибут `UNICODE`. Он назначает столбцу `CHAR` набор символов `ucs2`.

MySQL позволяет объявить столбец типа `CHAR(0)`. В основном это применимо в тех случаях, когда нужно обеспечить совместимость со старыми приложениями, которые зависят от наличия таких столбцов, но не используют их значения. Это также неплохо, если вам нужно иметь столбец, принимающий только два значения: столбец `CHAR(0)`, не объявленный с атрибутом `NOT NULL`, занимает только один бит и может принимать два значения: `NULL` и `''` (пустая строка).

■ **CHAR**

Это синоним для `CHAR(1)`.

■ **[NATIONAL] VARCHAR(M) [BINARY]**

Строка переменной длины. *M* представляет максимальную длину столбца. Диапазон значений *M* — от 0 до 255 символов (до MySQL 3.23 был от 1 до 255).

**На заметку!**

При сохранении значения столбца завершающие пробелы удаляются, что отличается от стандартной спецификации SQL.

Начиная с MySQL 4.1.0, столбцы этого типа с длиной, превышающей 255 символов, преобразуются к наименьшему типу `TEXT`, который может вместить значения указанной длины. Например, `VARCHAR(500)` преобразуется в `TEXT`, а `VARCHAR(200000)` — в `MEDIUMTEXT`. Данная возможность обеспечивает совместимость. Однако это преобразование затрагивает удаление завершающих пробелов. `VARCHAR` представляет собой сокращение от `CHARACTER VARYING`.

Атрибут `BINARY` делает сортировку и сравнения строк независимыми от регистра.

■ **TINYBLOB, TINYTEXT**

Столбец `BLOB` или `TEXT` с максимальной длиной  $2^8 - 1$  символов.

■ **BLOB, TEXT**

Столбец `BLOB` или `TEXT` с максимальной длиной  $2^{16} - 1$  символов.

■ **MEDIUMBLOB, MEDIUMTEXT**

Столбец `BLOB` или `TEXT` с максимальной длиной  $2^{24} - 1$  символов.

■ **LONGBLOB, LONGTEXT**

Столбец `BLOB` или `TEXT` с максимальной длиной 4 294 967 295 или 4 Гбайт ( $2^{32} - 1$ ) символов. До версии MySQL 3.23 протокол клиент-серверного обмена и таблицы `MyISAM` накладывали ограничение в 16 Мбайт на длину коммуникационного пакета и длину строки. Начиная с MySQL 4.0, максимально допустимая длина `LONGBLOB` или `LONGTEXT` зависит от настроенного максимального размера пакета в клиент-серверном протоколе и наличия свободной памяти.

- `ENUM('значение1', 'значение2', ...)`

Перечисление. Строковый объект, который может иметь только одно значение из списка возможных 'значение1', 'значение2', ..., NULL или специальное значение ошибки ''. Столбец `ENUM` может иметь до 65535 различных значений. Имеет внутреннее представление в виде целого числа.

- `SET('значение1', 'значение2', ...)`

Набор. Строковый объект, который может иметь нуль и более значений, каждое из которых выбирается из списка допустимых 'значение1', 'значение2', ... Столбец типа `SET` может иметь максимум 64 члена. Внутреннее представление – целое число.

## 4.2. Числовые типы

MySQL поддерживает все стандартные числовые типы SQL. Эти типы включают в себя точные числовые типы (`INTEGER`, `SMALLINT`, `DECIMAL` и `NUMERIC`), а также приближенные числовые типы (`FLOAT`, `REAL` и `DOUBLE PRECISION`). Ключевое слово `INT` – это синоним для `INTEGER`, а `DEC` – синоним для `DECIMAL`.

В качестве расширения стандарта SQL, MySQL также поддерживает целые типы `TINYINT`, `MEDIUMINT` и `BIGINT`, как перечислено в табл. 4.1:

Таблица 4.1. Целые типы данных в MySQL

Тип	Байт	Минимум (знаковое)	Максимум (знаковое)
<code>TINYINT</code>	1	-128	127
<code>SMALLINT</code>	2	-32768	32767
<code>MEDIUMINT</code>	3	-8388608	8388607
<code>INT</code>	4	-2147483648	2147483647
<code>BIGINT</code>	8	-9223372036854775808	9223372036854775807

Другое расширение поддерживается MySQL для необязательного указания ширины отображения целого значения в скобках, которое следует за базовым ключевым словом типа (например, `INT(4)`). Эта спецификация ширины отображения применяется для дополнения пробелами слева тех чисел, у которых разрядов меньше, чем в ней указано.

Ширина отображения никак не ограничивает ни диапазон допустимых значений столбца, ни количество разрядов для чисел, превышающих указанную ширину.

При использовании совместно с необязательным атрибутом `ZEROFILL` дополнение пробелами слева заменяется дополнением нулями. Например, для столбца, объявленного как `INT(5) ZEROFILL`, значение 4 извлекается как 00004. Имейте в виду, что если вы сохраняете значения, количество разрядов в которых больше заданной ширины отображения, то можете столкнуться с проблемами, когда MySQL будет генерировать временные таблицы для некоторых сложных соединений, поскольку в этом случае MySQL предполагает, что данные должны поместиться в отведенную ширину.

Все целые типы могут иметь необязательный (нестандартный) атрибут `UNSIGNED` (беззнаковое). Беззнаковые значения могут использоваться, когда вы хотите разрешить только неотрицательные значения в столбцах и вам требуется больший верхний предел для них.

Начиная с MySQL 4.0.2, типы с плавающей и фиксированной точкой тоже могут иметь атрибут `UNSIGNED`. Как и для целых типов, он предотвращает сохранения отрица-

тельных значений в таких столбцах. Однако, в отличие от целых, при этом верхний допустимый предел остается неизменным.

Если вы указываете атрибут `ZEROFILL` для числового столбца, MySQL автоматически добавляет атрибут `UNSIGNED`.

`DECIMAL` и `NUMERIC` в MySQL реализованы как один и тот же тип. Они применяются для хранения значений, которые должны быть точными, например, для денежных величин. Когда объявляется столбец одного из этих типов, ей можно указать (и обычно так и делается) точность (*precision*) и масштаб (*scale*). Например:

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

В этом примере 5 – это точность, а 2 – масштаб. Точность представляет количество значащих десятичных разрядов, а масштаб – количество разрядов после десятичной точки.

MySQL сохраняет `DECIMAL` и `NUMERIC` в виде строк вместо того, чтобы хранить в виде двоичных чисел с плавающей точкой, для обеспечения исключительной точности этих значений. При этом используется по одному символу для каждого разряда, один символ – для десятичной точки (если масштаб больше 0), и один – для знака '-' (для отрицательных значений). Если масштаб равен 0, `DECIMAL` и `NUMERIC` не содержат десятичной точки и дробной части.

Стандарт SQL требует, чтобы столбец `salary` мог хранить любое значение с пятью разрядами и двумя знаками после точки. Поэтому в данном случае диапазон допустимых значений `salary` составит от -999.99 до 999.99. MySQL при этом идет двумя путями:

- В положительной области диапазона столбец может хранить значения до 999.99. Для положительных чисел MySQL использует зарезервированный для знака байт, чтобы увеличить верхний предел диапазона.
- До MySQL 3.23 столбцы `DECIMAL` сохранялись иначе и не могли представлять все значения, которые требовал стандарт SQL. Это было связано с тем, что для типа `DECIMAL(M,D)` значение *M* включало байты для знака и десятичной точки. Таким образом, диапазон значений `salary` до MySQL 3.23 составлял от -9.99 до 9.99.

В стандартном SQL синтаксис `DECIMAL(M)` эквивалентен `DECIMAL(M,0)`. Соответственно, синтаксис `DECIMAL` эквивалентен `DECIMAL(M)`, где *M* по умолчанию определяется реализацией. Начиная с MySQL 3.23.6, поддерживаются обе формы типов данных `DECIMAL` и `NUMERIC`. Значение *M* по умолчанию равно 10. До версии MySQL 3.23.6 значения *M* и *D* должны были указываться явно.

Максимальный диапазон значений `DECIMAL` и `NUMERIC` такой же как `DOUBLE`, но фактический диапазон значений конкретного столбца мог быть ограничен точностью и масштабом этой столбца. Когда такому столбцу присваивается значение с большим количеством десятичных разрядов после точки, чем указано в его спецификации масштаба, значение преобразуется в указанный масштаб. (Приведение точности зависит от операционной системы, но обычно это делается просто отбрасыванием лишних разрядов.) Когда в столбец `DECIMAL` и `NUMERIC` записывается значение, выходящее за допустимый диапазон, он принимает граничное значение допустимого диапазона.

Для типов столбцов с плавающей точкой MySQL использует четыре байта для значений одинарной точности и восемь для значений двойной точности.

Тип `FLOAT` применяется для представления приближенных числовых типов данных. Стандарт SQL позволяет указывать необязательную спецификацию точности (но не диапазон экспоненты) в битах в скобках, следом за ключевым словом `FLOAT`. Реализация MySQL

также поддерживает эту спецификацию точности, но значение точности используется только для указания хранимого размера. Точность от 0 до 23 обеспечивает тип столбца `FLOAT` одинарной точности. Точность от 24 до 53 – тип столбца `DOUBLE` двойной точности.

Когда ключевое слово `FLOAT` используется для типа столбца без спецификации точности, MySQL использует четыре байта для хранения значений. MySQL также поддерживает синтаксис с двумя числами, указанными в скобках после слова `FLOAT`. При этом первое число указывает отображаемую ширину, а второе – количество разрядов после десятичной точки, которое должно сохраняться и отображаться (так же, как у `DECIMAL` и `NUMERIC`). Когда MySQL требуется в данной столбцу сохранить число с большим количеством разрядов после точки, чем указано в спецификации этой столбца, при сохранении значение округляется для исключения лишних разрядов.

В стандарте SQL типы `REAL` и `DOUBLE PRECISION` не допускают спецификаций точности. MySQL поддерживает вариацию синтаксиса с двумя числами в скобках следом за именем типа. Первое число представляет отображаемую ширину, а второе – количество разрядов после точки, которые должны сохраняться и отображаться. В качестве расширения стандарта SQL, MySQL распознает `DOUBLE` как синоним типа `DOUBLE PRECISION`. По контрасту с требованиями стандарта, что точность `REAL` должна быть меньше, чем `DOUBLE PRECISION`, MySQL реализует оба эти типа, как восьмибайтное число с плавающей точкой двойной точности (если только SQL-режим сервера не включает опцию `REAL_AS_FLOAT`).

Для достижения максимальной переносимости код, требующий сохранения приближенных числовых значений, должен использовать `FLOAT` и `DOUBLE PRECISION` без указания точности или количества разрядов после точки.

Когда в столбец записывается значение, выходящее за диапазон допустимых для данного типа, он принимает и сохраняет граничное значение допустимого диапазона.

Например, диапазон значений для столбцов типа `INT` составляет от -2147483648 до 2147483647. Если вы попытаетесь вставить значение -9999999999 в столбец типа `INT`, MySQL усечет его до нижней границы диапазона и сохранит -2147483648. Точно так же, если попытаться записать в этот столбец значение 9999999999, сохранено будет 2147483647.

Если столбец `INT` имеет атрибут `UNSIGNED`, размер диапазона значений такой же, но его границы смещаются к 0 и 4294967295. Если попытаться записать -9999999999 и 9999999999 в этот столбец, будут сохранены значения 0 и 4294967295 соответственно.

О преобразовании, выполняемом при таком усечении значений, сообщается в виде предупреждений для операторов `ALTER TABLE`, `LOAD DATA INFILE` и многострочных операторов `INSERT`.

## 4.3. Типы даты и времени

Типами даты и времени представления значений времени и даты являются `DATETIME`, `DATE`, `TIMESTAMP`, `TIME` и `YEAR`. Каждый их временных типов имеет диапазон допустимых значений, а также “нулевого” значения, используемого, когда вы указываете неверное значение, которое MySQL не может представить. Тип `TIMESTAMP` имеет специальное автоматическое поведение при обновлении, которое будет описано далее.

MySQL позволяет сохранять некоторые не “строго правильные” даты, такие как '1999-05-31'. Причина состоит в том, что мы считаем, что проверка корректности даты – обязанность приложения, а не SQL-сервера. Чтобы обеспечить более высокую скорость проверки дат, MySQL проверяет только, чтобы месяц был от 0 до 12, а день – от 0

до 31. Эти диапазоны включают нулевые значения, так как MySQL позволяет сохранять в столбцах типа DATE и DATETIME даты, у которых месяц и день равны нулю. Это очень удобно для приложений, которым необходимо хранить дни рождения, для которых неизвестна точная дата. В этом случае вы просто сохраняете дату вроде '1999-00-00' или '1999-01-00'. Если вы сохраняете дату в таком виде, то не должны ожидать корректных результатов от таких функций, как DATE\_SUB() и DATE\_ADD(), которые требуют в аргументах полной корректной даты.

Ниже приведены некоторые общие соглашения, которые следует иметь в виду при работе с типами времени и дат.

- MySQL извлекает значения типа времени и даты в стандартном формате вывода, но пытается интерпретировать множество форматов для вводимых значений (например, когда вы указываете значение времени и даты для присвоения или сравнения). Поддерживаются только форматы, описанные ниже. Ожидается, что вы будете вводить корректные значения, а при вводе значений в других форматах возможен непредсказуемый результат.
- Даты, содержащие двузначный год, неоднозначны, так как для них неизвестен век. MySQL интерпретирует двузначные годы по следующим правилам:
  - Год в диапазоне 00–69 преобразуется в 2000–2069.
  - Год в диапазоне 70–99 преобразуется в 1970–1999.
- Несмотря на то что MySQL старается интерпретировать значения в разных форматах, даты должны вводиться в порядке 'год-месяц-день' (например '98-09-04') вместо порядка 'месяц-день-год' или 'день-месяц-год', используемого где-либо (как например, '09-04-98' или '04-09-98').
- MySQL автоматически преобразует значения даты или времени, которые выходят за пределы диапазона, либо иным образом не соответствующие допустимым для данного типа (как описано в начале данного раздела), обращая их в “нулевые” значения (табл. 4.2). Исключением является тип TIME, для которого величины, входящие за пределы диапазона, смещаются в соответствующее допустимое граничное значение.

Таблица 4.2. “Нулевые” значения для разных типов столбцов

Тип столбца	“Нулевое” значение
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	0000000000000000
TIME	'00:00:00'
YEAR	0000

- “Нулевые” значения являются специальными, но вы можете сохранять их и ссылаться на них, используя константы, приведенные в таблице. Это можно делать также, используя '0' или 0, что проще.
- “Нулевые” значения даты и времени, используемые через Connector/ODBC (от версии 2.50.12 и выше), автоматически преобразуются в NULL, поскольку ODBC не может работать с такими значениями.

### 4.3.1. Типы DATETIME, DATE и TIMESTAMP

Типы DATETIME, DATE и TIMESTAMP связаны друг с другом. Этот раздел описывает их характеристики, в чем они сходны и в чем отличаются.

Тип DATETIME применяется, когда необходимо иметь значения, включающие и дату и время. MySQL извлекает и отображает значения типа DATETIME в формате 'ГГГГ-ММ-ДД ЧЧ:ММ:СС'. Поддерживаемый диапазон значений для них — от '1000-01-01 00:00:00' до '9999-12-31 23:59:59' ("Поддерживаемый" означает, что более ранние значения могут работать, но это не гарантируется.)

Тип DATE применяется, когда необходимо иметь значения, включающие только дату, без времени. MySQL извлекает и отображает значения типа DATE в формате 'ГГГГ-ММ-ДД'. Поддерживаемый диапазон — от '1000-01-01' до '9999-12-31'.

Тип столбца TIMESTAMP имеет ряд свойств, зависящих от версии MySQL и SQL-режима, в котором работает сервер. Эти свойства описаны далее в настоящем разделе.

Вы можете специфицировать значения типов DATETIME, DATE и TIMESTAMP, используя любой из общепринятых наборов форматов:

- Как строку в формате 'ГГГГ-ММ-ДД ЧЧ:ММ:СС' или 'ГГ-ММ-ДД ЧЧ:ММ:СС'. Допускается "ослабленный" синтаксис: любой символ пунктуации может быть использован в качестве разделителя между датой и временем. Например, '98-12-31 11:30:45', '98.12.31 11+30+45', '98/12/31 11\*30\*45' и '98@12@31 11^30^45' — эквивалентны.
- Как строку в формате 'YYYY-MM-DD' или 'YY-MM-DD'. "Ослабленный" синтаксис также допускается. Например, эквивалентны следующие значения: '98-12-31', '98.12.31', '98/12/31' и '98@12@31'.
- Как строку без разделителей в формате 'ГГГГММДДЧЧММСС' или 'ГГММДДЧЧММСС', предполагая, что строка имеет смысл в качестве даты. Например, '19970523091528' и '970523091528' интерпретируются как '1997-05-23 09:15:28', но '971122129015' неверно (потому что имеет бессмысленное значение минут) и становится '0000-00-00 00:00:00'.
- Как строку без разделителей в формате 'ГГГГММДД' или 'ГГММДД', предполагая, что строка имеет смысл в качестве даты. Например, '19970523' и '980523' интерпретируются как '1997-05-23', но '971332' неверно (неправильное значение месяца и дня) и превращается в '0000-00-00'.
- Как число в формате ГГГГММДДЧЧММСС или ГГММДДЧЧММСС, предполагая, что число имеет смысл в качестве даты. Например, 19830905132800 и 830905132800 интерпретируются как '1983-09-05 13:28:00'.
- Как число в формате ГГГГММДД или ГГММДД, предполагая, что число имеет смысл в качестве даты. Например, 19830905 и 830905 интерпретируются как '1983-09-05'.
- Как результат функции, которая возвращает приемлемое в контексте DATETIME, DATE или TIMESTAMP значение, такое как NOW() или CURRENT\_DATE.

Неверные величины DATETIME, DATE или TIMESTAMP преобразуются в "нулевые" значения соответствующего типа ('0000-00-00 00:00:00', '0000-00-00' или 000000000000000).

Для значений, указанных в виде строки, включающей разделитель даты, нет необходимости задавать два разряда для месяца или дня, которые меньше 10. '1976-6-9' — это то

же самое, что и '1976-06-09'. Аналогично, для значений, заданных в виде строки, включающей разделитель времени, не нужно указывать два разряда для часов, минут и секунд, которые меньше 10. '1979-10-30 1:2:3' – это то же самое, что '1979-10-30 01:02:03'.

Значения, заданные в виде числа, должны иметь длину 6, 8, 12 или 14 разрядов. Если число имеет длину 8 или 14 разрядов, предполагается, что оно задает значение в формате ГГГГММДД или ГГГГММДДЧЧММСС и что год задан четырьмя разрядами. Если число имеет длину 6 или 12, то предполагается, что оно задает значение в формате ГГММДД или ГГММДДЧЧММСС и год задан двумя разрядами. Числа, длина которых отличается от 6, 8, 12 и 14, дополняются ведущими нулями до ближайшего количества разрядов из указанного ряда.

Значения, заданные в виде строки без разделителей, интерпретируются с использованием их длины, как описано выше. Если длина строки 8 или 14 символов, предполагается, что год задан в 4-значном формате. В противном случае предполагается, что год задан первыми двумя знаками. Строка интерпретируется слева направо, чтобы извлечь значения года, месяца, дня, часов, минут и секунд. Это означает, что вы не должны использовать строки длиной менее 6 символов. Например, если вы укажете '9903', имея в виду март 1999 года, то обнаружите, что MySQL вставит “нулевую” дату в таблицу. Так получается из-за того, что значения года и месяца равны 99 и 03, но часть, указывающая день, полностью отсутствует, то есть это значение не задает корректную дату. Однако, начиная с MySQL 3.23, вы можете явно указать нулевое значение месяца или дня. Например, можно указать '990300', чтобы вставить в таблицу значение '1999-03-00'.

В определенных пределах вы можете присваивать значения одного типа объектам другого типа. Однако, при этом возможно некоторое искажение с потерей информации:

- Если вы присваиваете значение типа DATE объекту типа DATETIME или TIMESTAMP, временная часть значения принимается равной '00:00:00', поскольку значения типа DATE не содержат информации о времени.
- Если вы присваиваете значение типа DATETIME или TIMESTAMP объекту типа DATE, временная часть значения теряется, поскольку DATE не может ее включить в себя.
- Помните, несмотря на то, что значения DATETIME, DATE и TIMESTAMP могут быть указаны с использованием одного и того же набора форматов, диапазоны их допустимых значений отличаются. Например, значения TIMESTAMP не могут быть ранее 1970 или позднее 2037 года. Это означает, что дата вроде '1968-01-10', которая вполне корректна в качестве значения типа DATETIME или DATE, неверна для типа TIMESTAMP и будет преобразована в 0 при присвоении такому объекту.

Не следует также забывать о некоторых ловушках при указании значений дат:

- “Ослабленный” формат значений, заданных в виде строк, может вводит в заблуждение. Например, значение вроде '10:11:12' может выглядеть как время, потому что используется разделитель ':', но если оно применяется в контексте даты, то будет интерпретировано как '2010-11-12'. В то же время значение '10:45:15' будет преобразовано в '0000-00-00', поскольку '45' не является допустимым месяцем.
- Сервер MySQL выполняет только базовую проверку правильности дат: диапазоны значений года, месяца и дня составляют соответственно от 1000 до 9999, от 00 до 12 и от 00 до 31. Любые даты, содержащие части, выходящие за пределы этих диапазонов, становятся субъектами преобразования в '0000-00-00'. Помните, что

это позволяет вам сохранять неверные даты, вроде '2002-04-31'. Чтобы гарантировать правильность даты, выполняйте проверку внутри приложения.

- Даты, содержащие двузначный год, неоднозначны, потому что неизвестен век. MySQL интерпретирует двузначные годы следующим образом:
- Год в диапазоне 00–69 преобразуется в 2000–2069.
- Год в диапазоне 70–99 преобразуется в 1970–1999.

#### 4.3.1.1. Свойства **TIMESTAMP** в версиях MySQL, предшествующих 4.1

**TIMESTAMP** представляет собой тип столбца, который можно использовать для автоматической отметки текущей даты и времени при выполнении операций **UPDATE** или **INSERT**. Если в таблице несколько столбцов типа **TIMESTAMP**, только первый из них обновляется автоматически.

Автоматическое обновление первого столбца **TIMESTAMP** в таблице выполняется при наступлении одного из следующих условий:

- При явном присвоении ему значения **NULL**.
- Столбец не указан явно в операторе **INSERT** или **LOAD DATA INFILE**.
- Столбец не указан явно в операторе **UPDATE**, а значение какого-то другого столбца при этом изменяется. Оператор **UPDATE**, устанавливающий столбцу такое же значение, как он имел ранее, не приводит к обновлению столбца **TIMESTAMP**. Если вы присваиваете старое значение, MySQL игнорирует это в целях эффективности.

Столбцы типа **TIMESTAMP** также могут обновляться текущей датой и временем, если устанавливать их явно в нужное значение. Это верно даже для первого столбца **TIMESTAMP**. Вы можете использовать это свойство, например, если хотите установить значение столбца равным текущей дате и времени при создании строки, но не менять его при последующих обновлениях строки:

- Позвольте MySQL установить значение столбца при создании строки. Это инициализирует ее текущим значением даты и времени.
- При выполнении последующих обновлений других столбцов строки устанавливайте значение столбца **TIMESTAMP** равным его текущему значению:

```
UPDATE имя_таблицы
SET столбец_timestamp = столбец_timestamp,
    другой_столбец1 = новое_значение1,
    другой_столбец2 = новое_значение2, ...
```

Другой способ поддерживать столбец, который записывает время создания строки, предполагает использование столбца **DATETIME**, инициализируемого значением **NOW()** при создании строки и не изменяемого в дальнейшем.

Значения **TIMESTAMP** могут изменяться от начала 1970 года до части 2037 года с разрешением в одну секунду. Значения отображаются в виде чисел.

Формат, в котором MySQL извлекает и отображает значения **TIMESTAMP**, зависит от ширины отображения, как иллюстрирует табл. 4.3. Полный формат **TIMESTAMP** состоит из 14 разрядов, однако столбцы **TIMESTAMP** можно определить и в более коротком формате отображения.



Таблица 4.3. Зависимость формата отображения от ширины

Тип столбца	Формат отображения
TIMESTAMP (14)	ГГГГММДДЧЧММСС
TIMESTAMP (12)	ГГММДДЧЧММСС
TIMESTAMP (10)	ГГММДДЧЧММ
TIMESTAMP (8)	ГГГГММДД
TIMESTAMP (6)	ГГММДД
TIMESTAMP (4)	ГГММ
TIMESTAMP (2)	ГГ

Все столбцы `TIMESTAMP` имеют один и тот же размер хранения, независимо от формата отображения. Наиболее часто используемые форматы – в 6, 8, 12 и 14 символов. Вы можете задать произвольный размер отображения при создании таблицы, но значения 0 и больше 14 приводятся к 14. Нечетные значения от 1 до 13 приводятся к ближайшему большему четному.

Столбца `TIMESTAMP` хранят корректные значения, используя полную точность, с которой они были указаны, независимо от ширины отображения. Однако с этим связаны и некоторые ограничения:

- Следует всегда указывать год, месяц и день, даже если столбец объявлен как `TIMESTAMP (4)` или `TIMESTAMP (2)`. Иначе значение считается некорректным и сохраняется 0.
- Если вы используете `ALTER TABLE`, чтобы расширить столбец `TIMESTAMP`, то будет высвечиваться информация, которая ранее была “скрытой”.
- Аналогично, при сужении столбца `TIMESTAMP` информация не теряется, кроме как в том смысле, что выводиться будет меньше информации, чем ранее.
- Несмотря на то что столбцы `TIMESTAMP` хранятся с полной точностью, единственной функцией, которая работает с полным объемом хранимой в них информации, является `UNIX_TIMESTAMP()`. Все остальные функции работают с форматированным извлеченным значением. Это значит, что вы не можете использовать функцию типа `hour()` или `second()`, если только соответствующая часть не включена в форматированное значение столбца. Например, часть `чч` столбца `TIMESTAMP` не будет отображаться, если только ее отображаемая ширина не равна, по меньшей мере, 10, поэтому применение `hour()` для более коротких значений `TIMESTAMP` приведет к бессмысленному результату.

#### 4.3.1.2. Свойства `TIMESTAMP` в MySQL версии 4.1 и выше

Начиная с MySQL 4.1, свойства `TIMESTAMP` отличаются от тех, что были в предшествующих выпусках:

- Столбцы `TIMESTAMP` отображаются в том же формате, что и столбцы `DATETIME`.
- Ширина отображения больше не поддерживается, как описано ранее. Другими словами, теперь нельзя использовать `TIMESTAMP (4)` или `TIMESTAMP (2)`.

В дополнение, если сервер MySQL запущен в режиме `MAXDB`, тип `TIMESTAMP` идентичен `DATETIME`. То есть, если сервер запущен в режиме `MAXDB` в момент создания таблицы,

любые столбцы `TIMESTAMP` создаются как `DATETIME`. В результате эти столбцы используют формат отображения `DATETIME`, имеют тот же диапазон допустимых значений и никакого автоматического обновления не происходит.

В режиме `MAXDB` сервер `MySQL` можно запускать, начиная с версии 4.1.1. Чтобы включить этот режим, укажите при запуске сервера опцию `--sql-mode=MAXDB`, либо во время выполнения установите значение глобальной переменной `sql_mode`:

```
mysql> SET GLOBAL sql_mode=MAXDB;
```

Клиент может принудить сервер работать в режиме `MAXDB` для его собственного сеанса с помощью команды:

```
mysql> SET SESSION sql_mode=MAXDB;
```

### 4.3.2. Тип `TIME`

`MySQL` извлекает и отображает значения `TIME` в формате `'ЧЧ:ММ:СС'` (или `'чч:мм:сс'` для больших значений). Значения типа `TIME` могут изменяться в пределах от `'-838:59:59'` до `'838:59:59'`. Причина того, что значение часов может быть несколько большим, состоит в том, что тип `TIME` может использоваться не только для представления текущего времени (которое не может быть больше 24), но также для того, чтоб хранить значение времени, прошедшего с какого-то момента или временного интервала между двумя событиями (которые могут быть больше 24 часов и даже могут быть отрицательными).

Значение `TIME` можно специфицировать в различных форматах:

- Как строку в формате `'д ЧЧ:ММ:СС.дробная_часть'`. Вы можете также пользоваться “ослабленным” синтаксисом: `'ЧЧ:ММ:СС.дробная_часть'`, `'ЧЧ:ММ:СС'`, `'чч:мм'`, `'д ЧЧ:ММ:СС'`, `'д ЧЧ:ММ'`, `'д ЧЧ'` и `'СС'`. Здесь `д` представляет дни, значения которых могут быть от 0 до 34. Помните, что `MySQL` не хранит дробную часть.
- Как строку без разделителей в формате `'ННММСС'`, в предположении, что она содержит корректное время. Например, `'101112'` понимается как `'10:11:12'`, но `'109712'` — неверное значение (бессмысленная величина минут) и обращается в `'00:00:00'`.
- Как число формата `ННММСС`, в предположении, что оно содержит корректное время. Например, `101112` понимается как `'10:11:12'`. Следующие альтернативные форматы также допускаются: `СС`, `ММСС`, `ЧЧММСС`, `ЧЧММСС.дробная_часть`. Но помните, что `MySQL` не хранит дробную часть.
- Как результат функции, которая возвращает значение, приемлемое в контексте `TIME`, например, `CURRENT_TIME`.

Для значений `TIME`, заданных в виде строки, включающей разделитель частей времени, нет необходимости указывать два разряда для часов, минут и секунд, чье значение меньше 10. `'8:3:2'` — это то же самое, что и `'08:03:02'`.

Будьте осторожны, присваивая “короткие” значения `TIME` столбцам типа `TIME`. Без двоеточий `MySQL` интерпретирует значения в предположении, что крайние правые разряды представляют секунды. (`MySQL` интерпретирует значения `TIME` как периоды времени, а не время дня.) Например, вы можете подразумевать под `'1112'` и `1112` значение `'11:12:00'` (двенадцать минут двенадцатого), но `MySQL` интерпретирует это как `'00:11:12'` (11 минут 12 секунд). Подобным образом `'12'` и `12` интерпретируются как

'00:00:12'. В отличие от этого, значения TIME с двоеточиями всегда рассматриваются как время дня. То есть '11:12' означает '11:12:00', а не '00:11:12'.

Значения, которые лежат за пределами допустимых для типа TIME, но в остальном корректные, приводятся к ближайшему допустимому пределу. Например, '-850:00:00' и '850:00:00' преобразуются в '-838:59:59' и '838:59:59'.

Неверные значения TIME преобразуются в '00:00:00'. Нужно отметить, что поскольку '00:00:00' – вполне корректное значение TIME, нет никакой возможности узнать для хранимого в таблице '00:00:00' – было ли оно явно указано таким при сохранении, или же получилось в результате попытки записать некорректное значение.

### 4.3.3. Тип YEAR

Тип YEAR – это однобайтовый тип, применяемый для представления года.

MySQL извлекает и отображает значения типа YEAR в формате GTGT. Допустимый диапазон – от 1901 до 2155.

Вы можете специфицировать значения YEAR в различных форматах:

- Как четырехзначную строку в диапазоне от '1901' до '2155'.
- Как четырехзначное число в диапазоне от 1901 до 2155.
- Как двузначную строку в диапазоне от '00' до '99'. Значения в диапазоне от '00' до '69', и от '70' до '99' преобразуются в значения YEAR, соответственно, в диапазонах от 2000 до 2069 и от 1970 до 1999.
- Как двузначное число в диапазоне от 1 до 99. Значения от 1 до 69 и от 70 до 99 преобразуются в значения YEAR, соответственно, в диапазонах от 2001 до 2069 и от 1970 до 1999. Обратите внимание, что диапазон, задаваемый двузначным числом, слегка отличается от того, что задается двузначной строкой, так как вы не можете указать ноль непосредственно как число и интерпретировать его как 2000. Вы должны специфицировать его как строку '0' или '00', иначе он воспринимается как 0000.
- Как результат функции, которая возвращает значение, приемлемое в контексте YEAR, такой как NOW().

Неверные значения YEAR преобразуются в 0000.

### 4.3.4. Проблема двухтысячного года (Y2K) и типы данных

MySQL сам по себе защищен от влияния проблемы 2000-го года (см. раздел 1.2.5), но входные значения, представляемые MySQL, могут быть не защищены. Любой ввод, содержащий двузначное представление года, является неоднозначным, поскольку не указан век. Такие значения должны быть преобразованы в четырехзначную форму, потому что MySQL хранит значения лет во внутреннем представлении длиной в четыре цифры.

Для типов DATETIME, DATE, TIMESTAMP и YEAR MySQL интерпретирует даты с неоднозначным представлением года по следующим правилам:

- Год в диапазоне 00–69 преобразуется в 2000–2069.
- Год в диапазоне 70–99 преобразуется в 1970–1999.

Помните, что эти правила представляют только обоснованные предположения об истинном значении ваших данных. Если они не верны, вы должны однозначно указывать их, используя четырехзначный формат.

Конструкция `ORDER BY` правильно сортирует значения `TIMESTAMP` и `YEAR` с двузначным годом.

Некоторые функции, такие как `MIN()` и `MAX()`, преобразуют `TIMESTAMP` или `YEAR` в число. Это означает, что значения с двузначным годом не работают правильно с этими функциями. Чтобы исправить ситуацию, нужно преобразовать столбцы `TIMESTAMP` или `YEAR` в четырехзначный формат года или использовать что-то вроде:

```
MIN (DATE_ADD(timestamp, INTERVAL 0 DAYS))
```

## 4.4. Строковые типы

К строковым типам данных MySQL относятся `CHAR`, `VARCHAR`, `BLOB`, `TEXT`, `ENUM` и `SET`. В этом разделе описано, как работают эти типы и как ими пользоваться в запросах.

### 4.4.1. Типы `CHAR` и `VARCHAR`

Типы `CHAR` и `VARCHAR` похожи, но отличаются способом хранения и извлечения.

Длина столбца `CHAR` фиксирована и равна длине, которая указывается при создании таблицы. Она может иметь любое значение от 0 до 255 (до версии MySQL 3.23 длина `CHAR` могла быть от 1 до 255). При сохранении значение `CHAR` выравнивается за счет добавления пробелов справа до указанной длины столбца. Когда значение `CHAR` извлекается, завершающие пробелы удаляются.

Значения, хранимые в столбцах `VARCHAR`, являются строками переменной длины. Вы можете объявить столбец `VARCHAR` с длиной от 0 до 255, так же, как и `CHAR` (до версии MySQL 3.23 длина `VARCHAR` могла быть от 1 до 255). Однако, в отличие от `CHAR`, в столбце `VARCHAR` сохраняется столько знаков, сколько необходимо, плюс один байт для записи длины. Значение не дополняется пробелами, наоборот, завершающие пробелы удаляются при сохранении. Это поведение отличается от стандартных спецификаций SQL.

При сохранении и извлечении никаких преобразований регистра не выполняется.

Если вы присваиваете столбцу `CHAR` или `VARCHAR` значение, превышающее его длину, оно усекается.

Если вам нужен столбец, для которого не выполняется удаление завершающих пробелов, рассмотрите возможность применения типов `BLOB` или `TEXT`. Если вы хотите сохранять бинарные значения, такие как результаты, возвращаемые функциями сжатия или шифрования, которые могут содержать произвольные байты, используйте столбцы типа `BLOB` вместо `CHAR` или `VARCHAR`, дабы избежать потенциальных проблем с удалением пробелов, изменяющим данные.

В приведенной ниже табл. 4.4 представлены различия между двумя типами столбцов, на примере результата сохранения строкового значения в столбцы `CHAR(4)` и `VARCHAR(4)`.

Значения, извлеченные из столбцов `CHAR(4)` и `VARCHAR(4)`, будут одинаковы в любом случае, поскольку завершающие пробелы удаляются при чтении.

Начиная с версии MySQL 4.1.0, значения столбцов `CHAR` и `VARCHAR` сортируются и сравниваются в соответствии с порядком сопоставления серверного набора символов. Вы можете объявить столбец с атрибутом `BINARY`, чтобы сделать сортировку и сравнения нечувствительными к регистру, используя коды хранящихся символов вместо лексикографического порядка. Атрибут `BINARY` не влияет на то, как столбец сохраняется или извлекается.

Таблица 4.4. Сохранение значений в столбцах CHAR(4) и VARCHAR(4)

Значение	CHAR(4)	Требует	VARCHAR(4)	Требует
' '	' '	4 байта	' '	1 байт
'ab'	'ab '	4 байта	'ab'	3 байта
'abcd'	'abcd'	4 байта	'abcd'	5 байт
'abcdefgh'	'abcd'	4 байта	'abcd'	5 байт

Начиная с MySQL 4.1.0, тип столбца CHAR BYTE – это псевдоним для CHAR BINARY. Это сделано для совместимости.

Атрибут BINARY является жестким. Это означает, что если столбец, помеченный как BINARY, участвует в выражении, то все выражение становится BINARY.

Начиная с версии MySQL 4.1.0, для столбцов CHAR может быть указан атрибут ASCII. Это назначает столбцу набор символов latin1.

Начиная с MySQL 4.1.0, для столбцов CHAR может быть указан также атрибут UNICODE. Это назначает ему набор символов ucs2.

MySQL может молча изменить тип столбцов CHAR или VARCHAR при создании таблицы (см. раздел 6.2.5.2).

#### 4.4.2. Типы BLOB и TEXT

BLOB – это большой двоичный объект, который может содержать данные переменного объема. Существуют четыре типа BLOB: TINYBLOB, BLOB, MEDIUMBLOB и LONGBLOB; они отличаются только максимальной длиной хранимых значений. См. раздел 4.5.

Четыре типа TEXT – TINYTEXT, TEXT, MEDIUMTEXT и LONGTEXT – соответствуют четырем типам BLOB и имеют ту же максимальную длину и требования по хранению.

Столбцы BLOB рассматриваются как бинарные строки, в то время как столбцы TEXT рассматриваются в соответствии с назначенными им символьными наборами. До MySQL 4.1 столбцы TEXT сортировались и сравнивались на основе порядка сопоставления серверного набора символов.

При сохранении и извлечении никаких преобразований регистра не выполняется.

Если вы присваиваете столбцу BLOB или TEXT значение, превышающее его длину, оно усекается.

Во многих отношениях столбец типа TEXT можно рассматривать как столбец VARCHAR, который может быть настолько большим, насколько это необходимо. И точно так же можно рассматривать столбец типа BLOB, как “безразмерный” VARCHAR BINARY. Отличаются же типы BLOB и TEXT от CHAR и VARCHAR следующим:

- Индексы по столбцам BLOB и TEXT введены только в версии MySQL 3.23.2. Более старые версии не поддерживали индексацию по этим столбцам.
- При создании индексов по столбцам BLOB и TEXT необходимо указывать длину индексируемого префикса. Для CHAR и VARCHAR длина префикса не обязательна.
- При сохранении и извлечении никакого удаления завершающих пробелов из столбцов BLOB и TEXT не выполняется. Это отличает их от столбцов CHAR (завершающие пробелы удаляются при извлечении) и от столбцов VARCHAR (завершающие пробелы удаляются при сохранении).
- Столбцы BLOB и TEXT не могут иметь значений DEFAULT.

Начиная с MySQL 4.1.0, типы LONG и LONG VARCHAR отображаются на тип MEDIUMTEXT. Это сделано для совместимости.

Connector/ODBC определяет значения BLOB как LONGVARBINARY, а TEXT — как LONGVARCHAR.

Поскольку значения BLOB и TEXT могут быть чрезвычайно большими, вы можете столкнуться с некоторыми ограничениями при их использовании:

- Если вы хотите применять конструкции GROUP BY или ORDER BY на столбцах BLOB или TEXT, то должны преобразовать их значения в объекты с фиксированной длиной. Стандартный способ сделать это — воспользоваться функцией SUBSTRING, например:

```
mysql> SELECT comment FROM имя_таблицы, SUBSTRING(comment, 20) AS substr  
-> ORDER BY substr;
```

Если этого не сделать, для сортировки будет использовано только max\_sort\_length байт. Значение по умолчанию max\_sort\_length равно 1024; его можно изменить с помощью опции --max-sort-length при запуске сервера mysqld.

Группировать выражения, включающие значения BLOB и TEXT, можно, используя псевдонимы или номера позиций столбцов:

```
mysql> SELECT id, SUBSTRING(столбец_blob, 1, 100) AS b  
-> FROM имя_таблицы GROUP BY b;  
mysql> SELECT id, SUBSTRING(столбец_blob, 1, 100)  
-> FROM имя_таблицы GROUP BY 2;
```

- Максимальный размер объекта BLOB или TEXT определяется его типом, но фактическое наибольшее значение, которое можно передавать между клиентом и сервером, определяется объемом доступной памяти и размером коммуникационных буферов. Вы можете изменить размер буфера сообщений, изменяя значение переменной max\_allowed\_packet, но это нужно сделать как на стороне клиента, так и на стороне сервера. Так, например, и mysql, и mysqldump позволяют изменять значение переменной max\_allowed\_packet.

Каждое значение BLOB или TEXT имеет внутреннее представление в виде отдельно распределенного объекта. Это отличает их от других типов столбцов, для которых ресурсы хранения выделяются при открытии таблицы по одному разу на столбец.

### 4.4.3. Тип ENUM

ENUM (перечисление) — это строковый объект, имеющий значение, выбранное из списка допустимых, которые явно перечислены в спецификации столбца во время создания таблицы.

При некоторых условиях значение может быть также пустой строкой (' ') или NULL:

- Если вы вставляете недопустимое значение в столбец типа ENUM (то есть, строку, которой нет в списке допустимых значений), то вставляется пустая строка в качестве специального ошибочного значения. Эту строку можно отличить от нормальной пустой строки по тому признаку, что она имеет числовое значение 0. Подробнее об этом будет сказано далее.
- Если столбец типа ENUM объявлен как допускающий значение NULL, то NULL — это корректное значение для него, и это значение по умолчанию. Если же столбец объявлен как NOT NULL, то значением по умолчанию будет первое из ее списка допустимых значений.

Каждое перечислимое значение имеет индекс:

- Значения из списка допустимых, указанных в спецификации столбца, пронумерованы, начиная с 1.
- Индексное значение пустой строки равно 0. Это означает, что вы можете использовать следующий оператор `SELECT`, чтобы найти строки, в которые вставлялось неверное значение `ENUM`:

```
mysql> SELECT * FROM имя_таблицы WHERE столбец_enum = 0;
```

- Индекс значения `NULL` равен `NULL`.

Например, столбец, определенный как `ENUM('one', 'two', 'three')`, может иметь любое из значений, показанных ниже. Соответствующий каждому значению числовой индекс также показан:

Значение	Индекс
<code>NULL</code>	<code>NULL</code>
<code>' '</code>	0
<code>'one'</code>	1
<code>'two'</code>	2
<code>'three'</code>	3

Перечисление может иметь максимум 65535 элементов.

Начиная с версии MySQL 3.23.51, завершающие пробелы автоматически удаляются из значений-членов `ENUM` при создании таблицы.

Когда столбцу `ENUM` присваивается значение, регистр символов роли не играет. Однако значения, извлеченные из столбца позже, отображаются в нижнем регистре, как было при определении столбца.

Если вы извлекаете значение `ENUM` в числовом контексте, возвращается индекс значения столбца. Например, вы можете извлечь числовые значения столбца `ENUM` следующим образом:

```
mysql> SELECT столбец_enum + 0 FROM имя_таблицы;
```

Если вы сохраняете число в столбце `ENUM`, число рассматривается как индекс и сохраняемое значение — это член перечисления с этим индексом. (Однако это не работает с `LOAD DATA INFILE`, который трактует весь ввод как строки.) Мы не советуем определять столбцы `ENUM` со значениями, выглядящими как числа, потому что это легко может привести к путанице. Например, следующий столбец имеет перечислимые строковые значения `'0'`, `'1'` и `'2'`, но числовыми значениями индекса будут 1, 2 и 3:

```
numbers ENUM('0', '1', '2')
```

Значения `ENUM` сортируются в соответствии с порядком, в котором члены перечислений приведены в спецификации столбца (Другими словами, значения `ENUM` сортируются в соответствии с их числовыми индексами.) Например `'a'` предшествует `'b'` для `ENUM('a', 'b')`, но `'b'` предшествует `'a'` для `ENUM('b', 'a')`. Пустые строки предшествуют непустым строкам, а значения `NULL` предшествуют всем остальным. Чтобы предотвратить неожиданные результаты, определяйте списки значений `ENUM` в алфавитном порядке. Вы также можете применить `GROUP BY CAST(столбец AS VARCHAR)` или `GROUP BY CONCAT(столбец)`, чтобы обеспечить сортировку столбца в лексическом порядке, а не по порядку номеров индексов.

Если вы хотите просмотреть все возможные значения столбца ENUM, воспользуйтесь `SHOW COLUMNS FROM имя_таблицы LIKE столбец_enum` и проанализируйте определение ENUM во втором столбце вывода.

#### 4.4.4. Тип SET

SET (набор) – это строковый объект, которым может иметь от нуля и более значений, каждое из которых должно быть выбрано из списка допустимых значений, указанного при создании таблицы. Значения столбцов SET, которые состоят из множества членов, специфицируются списком членов, разделенных запятой. Следствием этого является то, что значения членов SET не могут содержать в себе запятых.

Например, столбец, объявленный как SET ('one', 'two') NOT NULL, может содержать только такие значения:

```
''
'one'
'two'
'one, two'
```

SET может иметь максимум 64 различных члена.

Начиная с MySQL 3.23.51, завершающие пробелы автоматически удаляются из значений членов SET при создании таблицы.

MySQL хранит значения SET в числовом виде, при этом первый справа бит соответствует первому члену набора. Если вы извлекаете значение SET в числовом контексте, то оно будет содержать набор битов, соответствующий членам, которые образуют значение столбца. Например, вы можете извлечь числовое значение столбца SET следующим образом:

```
mysql> SELECT столбец_set + 0 FROM имя_таблицы;
```

Если в столбце типа SET сохраняется число, биты, установленные в его двоичном представлении, определяют состав членов, входящих в значение столбца. Для столбца, определенного как SET('a', 'b', 'c', 'd'), его члены имеют следующие десятичные и двоичные значения:

Член SET	Десятичное	Двоичное
'a'	1	0001
'b'	2	0010
'c'	4	0100
'd'	8	1000

Если вы присвоите этому столбцу значение 9, что в двоичном виде выглядит как 1001, то будут выбраны первый и четвертый члены – 'a' и 'd', и результирующим значением будет 'a,d'.

Для значений, содержащих более одного элемента SET, неважно, в каком порядке были указаны элементы при вставке. Неважно также, сколько раз отдельный элемент встречался в списке. Когда значение позже будет извлечено, каждый элемент в наборе появится один раз, а их последовательность будет соответствовать порядку, в котором были перечислены допустимые значения при создании таблицы. Если столбец определен как SET('a', 'b', 'c', 'd'), то присвоенные значения 'a,d', 'd,a' и 'd,a,a,d,d' при извлечении дадут 'a,d'. Если вы присвоите столбцу SET неподдерживаемое значение, оно будет проигнорировано.



Значения SET сортируются в соответствии с числовыми представлениями. Значения NULL предшествуют всем значениям, отличным от NULL. Обычно поиск значения в наборе осуществляется с помощью функции `FIND_IN_SET()` или операции `LIKE`:

```
mysql> SELECT * FROM имя_таблицы
      WHERE FIND_IN_SET('значение', столбец_set) > 0;
mysql> SELECT * FROM имя_таблицы WHERE столбец_set LIKE '%значение%';
```

Первый оператор найдет строки, в которых `столбец_set` содержит член набора `значение`. Второй оператор похож, однако означает не то же самое. Он выбирает строки, в которых `столбец_set` содержит значение даже в виде подстроки в другом члене набора.

Следующие операторы также допустимы:

```
mysql> SELECT * FROM имя_таблицы WHERE столбец_set & 1;
mysql> SELECT * FROM имя_таблицы WHERE столбец_set = 'значение1, значение2';
```

Первый из этих операторов ищет значения, содержащие первый член набора. Второй – ищет полное совпадение. Будьте осторожны со сравнениями второго типа. Сравнение значений набора с `'значение1, значение2'` вернет другой результат, чем сравнение с `'значение2, значение1'`. Вы должны указывать элементы в том же порядке, в котором они перечислены в определении столбца.

Если вы хотите просмотреть все возможные значения столбца SET, воспользуйтесь `SHOW COLUMNS FROM имя_таблицы LIKE столбец_set` и проанализируйте определение SET во втором столбце вывода.

## 4.5. Требования по хранению типов столбцов

Требования к хранению каждого из типов столбцов, поддерживаемых MySQL, перечислены по категориям (табл. 4.5).

Максимальный размер строки таблицы MyISAM составляет 65534 байта. Каждый столбец BLOB и TEXT занимает только от пяти до девяти байт, в зависимости от размера.

Если таблицы MyISAM или ISAM содержат столбцы любых типов с переменной длиной, формат строки таблицы также будет переменной длины. При создании таблицы при некоторых условиях MySQL может изменять тип столбца с типа с фиксированной длиной на тип с переменной длиной и наоборот. См. раздел 6.2.5.2.

Типы VARCHAR, BLOB и TEXT являются типами с переменной длиной. Для каждого из них требования по хранению зависят от фактического размера значений столбцов (представленных в предыдущей таблице как *L*), а не от максимально возможного размера типа. Например, столбец типа VARCHAR(10) может содержать строки длиной максимум в 10 символов. Фактический размер хранения складывается из длины строки (*L*) плюс один байт, хранящий фактическую длину. Для строки 'abcd' *L* равно 4, и для хранения значения нужно 5 байт.

Типы BLOB и TEXT требуют 1, 2, 3 или 4 байта для записи длины значения столбца, в зависимости от максимально возможной длины конкретного типа. См. раздел 4.4.2.

Размер объекта типа ENUM определяется количеством различных перечислимых значений. Один байт используется для перечислений, имеющих до 255 возможных значений. Два байта используются для перечислений, имеющих до 65535 возможных значений. См. раздел 4.4.3.

Таблица 4.5. Требования по хранению для различных типов столбцов

Тип столбца	Требования по хранению
<i>Требования по хранению числовых типов</i>	
TINYINT	1 байт
SMALLINT	2 байта
MEDIUMINT	3 байта
INT, INTEGER	4 байта
BIGINT	8 байт
FLOAT (p)	4 байта, если $0 \leq p \leq 24$ , 8 байт, если $25 \leq p \leq 53$
FLOAT	4 байта
DOUBLE [PRECISION], REAL	8 байт
DECIMAL (M, D), NUMERIC (M, D)	$M+2$ байт, если $D > 0$ , $M+1$ байт, если $D = 0$ ( $D+2$ , если $M < D$ )
<i>Требования по хранению типов даты и времени</i>	
DATE	3 байта
DATETIME	8 байт
TIMESTAMP	4 байта
TIME	3 байта
YEAR	1 байт
<i>Требования по хранению строковых типов</i>	
CHAR (M)	$M$ байт, $0 \leq M \leq 255$
VARCHAR (M)	$L+1$ байт, где $L \leq M$ и $0 \leq M \leq 255$
TINYBLOB, TINYTEXT	$L+1$ байт, где $L < 28$
BLOB, TEXT	$L+2$ байт, где $L < 2^{16}$
MEDIUMBLOB, MEDIUMTEXT	$L+3$ байт, где $L < 2^{24}$
LONGBLOB, LONGTEXT	$L+4$ байт, где $L < 2^{32}$
ENUM ('значение1', 'значение2', ...)	1 или 2 байта, в зависимости от числа допустимых значений (максимум 65535)
SET ('значение1', 'значение2', ...)	1, 2, 3, 4 или 8 байт, в зависимости от числа членов набора (максимум 64)

Размер объекта типа SET определяется количеством различных членов набора. Если размер набора равен  $N$ , то объект занимает  $(N+7)/8$ , округленное до ближайшего большего значения из ряда 1, 2, 3, 4 или 8, байт. SET может иметь максимум 64 члена. См. раздел 4.4.4.

## 4.6. Выбор правильного типа столбца

Для наиболее эффективного использования хранилища во всех случаях старайтесь использовать наиболее точно подходящий тип. Например, если столбец будет хранить значения в диапазоне от 1 до 99999, то самым подходящим типом для него будет

MEDIUMINT UNSIGNED. Из всех типов, которые в состоянии представлять все нужные значения, этот является самым компактным.

Аккуратное представление денежных величин – это общеизвестная проблема. В MySQL вы можете использовать для этого тип DECIMAL. Он хранит значения в виде строк, поэтому не происходит никаких потерь точности. (Однако вычисления над столбцами DECIMAL могут производиться с использованием операций двойной точности.) Если точность не очень важна, тип DOUBLE может подойти достаточно хорошо.

Для высокой точности вы всегда можете преобразовать значения в тип с фиксированной точкой, хранимый как BIGINT. Это позволит выполнять все вычисления над целыми и при необходимости преобразовывать результат в формат числа с плавающей точкой.

## 4.7. Использование типов столбцов и других систем управления базами данных

Чтобы облегчить использование кода, написанного в других реализациях SQL, от других поставщиков, MySQL отображает типы столбцов друг на друга, как показано в табл. 4.6. Это отображение упрощает импорт определений таблиц из других СУБД в MySQL:

Таблица 4.6. Отображение типов столбцов

Тип в других СУБД	Тип MySQL
BINARY (M)	CHAR (M) BINARY
CHAR VARYING (M)	VARCHAR (M)
FLOAT4	FLOAT
FLOAT8	DOUBLE
INT1	TINYINT
INT2	SMALLINT
INT3	MEDIUMINT
INT4	INT
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
LONG	MEDIUMTEXT (MySQL 4.1.0)
MIDDLEINT	MEDIUMINT
VARBINARY (M)	VARCHAR (M) BINARY

Отображение типов столбцов происходит при создании таблиц, после чего исходная спецификация типа отбрасывается. Если вы создаете таблицу с типами, используемыми в продуктах других поставщиков, а затем выполняете оператор DESCRIBE имя\_таблицы, MySQL показывает структуру в терминах эквивалентных типов MySQL.

## Функции и операции

**В** операторах SQL выражения могут встречаться в нескольких местах, таких как конструкции ORDER BY или HAVING оператора SELECT, в конструкции WHERE операторов SELECT, DELETE или UPDATE, либо в операторах SET. Выражения могут быть записаны с использованием литеральных значений, значений столбцов, NULL, функций и операций. Эта глава описывает функции и операции, которые допускаются в выражениях MySQL.

Выражение, которое содержит NULL, всегда возвращает значение NULL, если только другое не указано в документации для данной функции или операции.

### На заметку!

По умолчанию не должно быть пробелов между именем функции и следующими за ним скобками. Это позволяет анализатору выражений MySQL делать различие между вызовами функций и ссылками на таблицы или столбцы, которые могут иметь то же имя, что функция. Однако пробелы вокруг аргументов функций допускаются.

Вы можете указать серверу MySQL, что нужно разрешить пробелы после имен функций, запустив его с опцией `--sql-mode=IGNORE_SPACE`. Отдельные клиентские программы могут включить такое поведение, используя опцию `CLIENT_IGNORE_CASE` для `mysql_real_connect()`. В обоих случаях имена функций становятся зарезервированными словами.

Для обеспечения краткости большинство примеров в этой главе отображают вывод программы `mysql` в сжатой форме. Вместо такого формата вывода:

```
mysql> SELECT MOD(29,9) ;
+-----+
| mod(29,9) |
+-----+
|          2 |
+-----+
1 rows in set (0.00 sec)
```

применяется такой:

```
mysql> SELECT MOD(29,9) ;
-> 2
```

## 5.1. Операции

### 5.1.1. Скобки

- ( ... ). Скобки используются для изменения приоритета операций. Например:

```
mysql> SELECT 1+2*3;  
-> 7  
mysql> SELECT (1+2)*3;  
-> 9
```

### 5.1.2. Операции сравнения

Результатом выполнения операций сравнения являются 1 (TRUE – истина), 0 (FALSE – ложь) или NULL. Эти операции действительны и для чисел, и для строк. При необходимости строки автоматически конвертируются в числа, а числа в строки.

MySQL выполняет сравнения, используя следующие правила:

- Если один или оба аргумента равны NULL, результатом сравнения будет NULL, за исключением NULL-безопасной операции <=>.
- Если оба аргумента операции сравнения – строки, они сравниваются как строки.
- Если оба аргумента – целые числа, они сравниваются как целые числа.
- Шестнадцатеричные значения интерпретируются как бинарные строки, если они не сравниваются с числами.
- Если один аргумент – столбец типа TIMESTAMP или DATETIME, а другой – константа, то константа перед сравнением конвертируется во временную метку. Это сделано для достижения большей совместимости с ODBC.
- Во всех других случаях аргументы сравниваются как числа с плавающей точкой (действительные числа).

По умолчанию, сравнения строк нечувствительны к регистру и используют текущий набор символов (ISO-8859-1 Latin, что отлично работает с английским языком).

Следующие примеры иллюстрируют преобразования строк в числа в операциях сравнения:

```
mysql> SELECT 1 > '6x';  
-> 0  
mysql> SELECT 7 > '6x';  
-> 1  
mysql> SELECT 0 > 'x6';  
-> 0  
mysql> SELECT 0 = 'x6';  
-> 1
```

Отметим, что когда вы сравниваете строковый столбец с числом, MySQL не может использовать индекс по столбцу для быстрого поиска значения. Если *строковый\_столбец* – индексированный строковый столбец, в следующем запросе индекс не может использоваться для поиска:

```
SELECT * FROM имя_таблицы WHERE строковый_столбец=1;
```

Причина состоит в том, что существует множество вариантов строки, которые могут быть конвертированы в число 1: '1', ' 1', '1a' и так далее.

■ **=**. Равенство:

```
mysql> SELECT 1 = 0;
-> 0
mysql> SELECT '0' = 0;
-> 1
mysql> SELECT '0.0' = 0;
-> 1
mysql> SELECT '0.01' = 0;
-> 0
mysql> SELECT '.01' = 0.01;
-> 1
```

■ **<=>**. NULL-безопасное равенство. Эта операция проверяет операнды на предмет равенства, как и **=**, но возвращает 1 вместо NULL, если оба операнда равны NULL, и 0 вместо NULL, если только один из операндов равен NULL.

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
-> 1, 1, 0
mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
-> 1, NULL, NULL
```

■ **<>**, **!=**. Неравенство:

```
mysql> SELECT '.01' <> '0.01';
-> 1
mysql> SELECT .01 <> '0.01';
-> 0
mysql> SELECT 'zapp' <> 'zappp';
-> 1
```

■ **<=**. Меньше или равно:

```
mysql> SELECT 0.1 <= 2;
-> 1
```

■ **<**. Меньше:

```
mysql> SELECT 2 < 2;
-> 0
```

■ **>=**. Больше или равно:

```
mysql> SELECT 2 >= 2;
-> 1
```

■ **>**. Больше:

```
mysql> SELECT 2 > 2;
-> 0
```

■ **IS NULL**, **IS NOT NULL**. Проверяет, равно ли значение NULL:

```
mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
-> 0, 0, 1
mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
-> 1, 1, 0
```

Чтобы обеспечить возможность работы с программами ODBC, MySQL поддерживает дополнительные средства для значений NULL:

- Вы можете искать строку, которая содержит самое последнее значение `AUTO_INCREMENT`, выполнив следующий запрос немедленно после генерирования этого значения:

```
SELECT * FROM имя_таблицы WHERE столбец_auto IS NULL
```

Такое поведение может быть отключено установкой `SQL_AUTO_IS_NULL=0`. См. раздел 6.5.3.1.

- Для столбцов `DATE` и `DATETIME`, которые объявлены как `NOT NULL`, вы можете найти специальную дату `'0000-00-00'`, применив запрос вроде следующего:

```
SELECT * FROM имя_таблицы WHERE столбец_date IS NULL
```

Это необходимо для правильной работы некоторых приложений ODBC, потому что ODBC не допускает значения даты `'0000-00-00'`.

■ выражение `BETWEEN` минимум AND максимум

Если выражение больше или равно минимум и меньше или равно максимум, то `BETWEEN` возвратит 1, иначе – 0. Если все аргументы одного типа, то это эквивалентно выражению `(минимум <= выражение AND выражение <= максимум)`. В противном случае преобразование типов выполняется в соответствии с правилами, описанными в начале раздела, но применяется ко всем трем аргументам. Следует отметить, что до версии MySQL 4.0.5 вместо этого аргументы конвертировались к типу `выражение`.

```
mysql> SELECT 1 BETWEEN 2 AND 3;
-> 0
mysql> SELECT 'b' BETWEEN 'a' AND 'c';
-> 1
mysql> SELECT 2 BETWEEN 2 AND '3';
-> 1
mysql> SELECT 2 BETWEEN 2 AND 'x-3';
-> 0
```

■ выражение `NOT BETWEEN` минимум AND максимум

Это то же самое, что `NOT (выражение BETWEEN минимум AND максимум)`.

■ `COALESCE(значение, ...)`

Возвращает первое значение в списке, не равное `NULL`.

```
mysql> SELECT COALESCE(NULL,1);
-> 1
mysql> SELECT COALESCE(NULL,NULL,NULL);
-> NULL
```

`COALESCE()` появилась в MySQL 3.23.3.

■ `GREATEST(значение1, значение2, ...)`

С двумя или более аргументами возвращает наибольший (с наибольшим значением) аргумент. Аргументы сравниваются по тем же правилам, что и в `LEAST()`.

```
mysql> SELECT GREATEST(2,0);
-> 2
mysql> SELECT GREATEST(34.0,3.0,5.0,767.0);
-> 767.0
mysql> SELECT GREATEST('B','A','C');
-> 'C'
```

До версии MySQL 3.22.5 вместо GREATEST() нужно было использовать MAX().

■ **выражение IN (значение, ...)**

Возвращает 1, если *выражение* равно любому из значений списка IN, в противном случае возвращает 0. Если все значения – константы, они обрабатываются в соответствии с типом *выражение* и сортируются. Извлечение значения затем выполняется с использованием бинарного поиска. Это значит, что IN работает очень быстро, если список состоит только из констант. Если же *выражение* является чувствительным к регистру и строковым, сравнения строк будут зависеть от регистра.

```
mysql> SELECT 2 IN (0,3,5,'wefwf');
-> 0
mysql> SELECT 'wefwf' IN (0,3,5,'wefwf');
-> 1
```

Количество значений в списке IN ограничивается только значением переменной max\_allowed\_packet.

Для соответствия стандарту SQL, начиная с MySQL 4.1, IN возвращает NULL не только когда *выражение* в левой части равно NULL, но также если соответствие не найдено и один из компонентов списка IN равен NULL.

Начиная с MySQL 4.1, синтаксис IN() также используется для записи некоторых типов подзапросов. См. раздел 6.1.8.3.

■ **выражение NOT IN (значение, ...)**

То же самое, что NOT (выражение IN (значение, ...)).

■ **ISNULL(выражение)**

Если *выражение* равно NULL, возвращает 1, иначе – 0.

```
mysql> SELECT ISNULL(1+1);
-> 0
mysql> SELECT ISNULL(1/0);
-> 1
```

Помните, что сравнение значений NULL с применением операции = всегда возвращает “ложь”!

■ **INTERVAL (N, N1, N2, N3, ...)**

Возвращает 0, если  $N < N1$ , 1 – если  $N < N2$ , и так далее, либо NULL, если  $N$  равно NULL. Все аргументы рассматриваются как целые. Для правильной работы этой функции требуется соблюдение условия  $N1 < N2 < N3 < \dots < Nn$ .

Это связано с применением бинарного поиска (очень быстрого).

```
mysql> SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
-> 3
mysql> SELECT INTERVAL(10, 1, 10, 100, 1000);
-> 2
mysql> SELECT INTERVAL(22, 23, 30, 44, 200);
-> 0
```

■ **LEAST(значение1, значение2, ...)**

С двумя или более аргументами возвращает минимальный (с минимальным значением) аргумент. Аргументы сравниваются по следующим правилам:



- Если возвращаемое значение используется в контексте INTEGER, или все аргументы целые, они сравниваются как целые.
- Если возвращаемое значение используется в контексте REAL, или же все аргументы – действительные числа, они сравниваются как действительные числа.
- Если любой из аргументов – чувствительная к регистру строка, все они сравниваются как чувствительные к регистру строки.
- В других случаях аргументы сравниваются как нечувствительные к регистру строки.

```
mysql> SELECT LEAST(2,0);
-> 0
mysql> SELECT LEAST(34.0,3.0,5.0,767.0);
-> 3.0
mysql> SELECT LEAST('B','A','C');
-> 'A'
```

До версии MySQL 3.22.5 вместо LEAST() нужно было использовать MIN().

Следует отметить, что предыдущие правила преобразования в некоторых граничных случаях могут приводить к получению странных результатов:

```
mysql> SELECT CAST(LEAST(3600, 9223372036854775808.0) as SIGNED);
-> -9223372036854775808
```

Это происходит потому, что MySQL читает 9223372036854775808.0 в целом контексте. Целое представление не подходит для этого значения, поэтому оно обращается в целое со знаком.

### 5.1.3. Логические операции

В SQL все логические операции возвращают TRUE, FALSE или NULL (UNKNOWN).

В MySQL это реализовано, как 1 (TRUE), 0 (FALSE) и NULL. В основном это совместимо с тем, что возвращают другие SQL-серверы, хотя некоторые в качестве TRUE могут возвращать любое ненулевое значение.

- NOT, !. Логическое НЕ. Возвращает 1, если операнд равен 0, и 0 – если операнд ненулевой. NOT NULL возвращает NULL.

```
mysql> SELECT NOT 10;
-> 0
mysql> SELECT NOT 0;
-> 1
mysql> SELECT NOT NULL;
-> NULL
mysql> SELECT ! (1+1);
-> 0
mysql> SELECT ! 1+1;
-> 1
```

В последнем примере возвращается 1, поскольку выражение вычисляется как (!1)+1.

- AND, &&. Логическое И. Возвращает 1, если оба операнда ненулевые и не NULL, если один или более операндов равны 0, то возвращает 0, в противном случае – NULL.

```
mysql> SELECT 1 && 1;
-> 1
mysql> SELECT 1 && 0;
-> 0
mysql> SELECT 1 && NULL;
-> NULL
mysql> SELECT 0 && NULL;
-> 0
mysql> SELECT NULL && 0;
-> 0
```

Следует отметить, что в версиях MySQL, предшествующих 4.0.5, вычисление прекращалось, если встречался NULL, вместо того, чтобы продолжать проверять на возможное наличие 0. Это означает, что в этих версиях `SELECT (NULL AND 0)` вернет NULL вместо 0. Начиная с MySQL 4.0.5, код был перестроен таким образом, чтобы результат выражения всегда соответствовал стандарту SQL, при этом используя оптимизацию, когда это возможно.

- **OR, ||. Логическое ИЛИ.** Возвращает 1, если любой операнд ненулевой, NULL – если любой операнд NULL, и 0 в остальных случаях.

```
mysql> SELECT 1 || 1;
-> 1
mysql> SELECT 1 || 0;
-> 1
mysql> SELECT 0 || 0;
-> 0
mysql> SELECT 0 || NULL;
-> NULL
mysql> SELECT 1 || NULL;
-> 1
```

- **XOR. Логическое ИСКЛЮЧАЮЩЕЕ ИЛИ.** Возвращает NULL, если оба операнда NULL. Для операндов, отличных от NULL, возвращает 1, если нечетное число операндов ненулевые, иначе возвращает 0.

```
mysql> SELECT 1 XOR 1;
-> 0
mysql> SELECT 1 XOR 0;
-> 1
mysql> SELECT 1 XOR NULL;
-> NULL
mysql> SELECT 1 XOR 1 XOR 1;
-> 1
```

$a \text{ XOR } b$  математически эквивалентно  $(a \text{ AND } (\text{NOT } b)) \text{ OR } ((\text{NOT } a) \text{ AND } b)$ .

Операция XOR появилась в MySQL 4.0.2.

### 5.1.4. Операции, чувствительные к регистру

- **BINARY.** Операция BINARY приводит строку-аргумент к типу бинарной строки. Это простой способ сделать сравнение столбцов чувствительным к регистру, даже если тип столбцов не BINARY или BLOB.

```
mysql> SELECT 'a' = 'A';
-> 1
mysql> SELECT BINARY 'a' = 'A';
-> 0
```

Операция BINARY появилась в MySQL 3.23.0. Начиная с версии MySQL 4.0.2, BINARY строка является краткой формой CAST(строка AS BINARY). См. раздел 5.7.

Следует отметить, что в некоторых контекстах, если вы приводите индексированный столбец к BINARY, MySQL не сможет использовать индекс.

Если вы хотите сравнивать значения BLOB в нечувствительной к регистру манере, то можно поступить так:

- В версиях MySQL, предшествующих 4.1.1, используйте функцию UPPER() для перевода символов значения BLOB в верхний регистр перед выполнением сравнения:

```
SELECT 'A' LIKE UPPER(столбец_blob) FROM имя_таблицы;
```

Если сравниваемое значение представлено в нижнем регистре, преобразуйте значение BLOB с помощью LOWER().

- Для MySQL 4.1.1 и выше столбцы BLOB имеют набор символов binary, который не поддерживает концепцию регистра. Для выполнения сравнения, нечувствительного к регистру, пользуйтесь функцией CONVERT(), чтобы преобразовать значения BLOB в набор символов, не зависящий от регистра. Результатом будет небинарная строка, к которой можно применить чувствительную к регистру операцию LIKE:

```
SELECT 'A' LIKE CONVERT(столбец_blob USING latin1) FROM имя_таблицы;
```

Чтобы использовать другой набор символов, подставьте его имя в предыдущий запрос вместо latin1.

CONVERT() может использоваться в более общем виде для сравнения строк, представленных в разных наборах символов.

## 5.2. Функции управления потоком выполнения

- CASE значение WHEN [значение-сравнения] THEN результат [WHEN [значение-сравнения] THEN результат ...] [ELSE результат] END,  
CASE WHEN [условие] THEN результат [WHEN [условие] THEN результат ...] [ELSE результат] END

Первая версия возвращает результат, когда значение=значение-сравнения. Вторая версия возвращает результат для первого истинного условия. Если нет соответствующего результирующего значения, возвращается результат, следующий за словом ELSE, либо NULL, если ELSE отсутствует.

```
mysql> SELECT CASE 1 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'more' END;
-> 'one'
mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
-> 'true'
mysql> SELECT CASE BINARY 'B' WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
-> NULL
```

Тип возвращаемого значения (INTEGER, DOUBLE или STRING) определяется типом первого возвращаемого значения (выражения после первого THEN).

Функция CASE появилась в MySQL 3.23.3.

■ **IF(выражение1, выражение2, выражение3)**

Если выражение *выражение1* истинно (*выражение1* <> 0 и *выражение1* <> NULL), то IF() возвращает *выражение2*, иначе — *выражение3*. IF() возвращает числовое или строковое значение, в зависимости от контекста, в котором применяется.

```
mysql> SELECT IF(1>2,2,3);
-> 3
mysql> SELECT IF(1<2,'yes','no');
-> 'yes'
mysql> SELECT IF(STRCMP('test','test1'),'no','yes');
-> 'no'
```

Если только одно из выражений *выражение2* или *выражение3* равно NULL, тип результата функции IF() будет типом выражения, отличного от NULL. (Такое поведение принято, начиная с MySQL 4.0.3.)

*выражение1* вычисляется как целое значение, что означает, что если вы проверяете число с плавающей точкой или строку, то должны это делать с помощью операции сравнения.

```
mysql> SELECT IF(0.1,1,0);
-> 0
mysql> SELECT IF(0.1<>0,1,0);
-> 1
```

В первом случае IF(0.1) вернет 0, потому что 0.1 преобразуется в целое, что приводит к проверке IF(0). Это может быть не тем, чего вы ожидаете. Во втором случае сравнение проверяет исходное значение с плавающей точкой на равенство нулю. Результат сравнения трактуется как целое.

Тип возврата по умолчанию для IF() (что может быть существенно при сохранении его во временной таблице) вычисляется MySQL следующим образом:

Выражение	Возвращаемое значение
<i>выражение2</i> или <i>выражение3</i>	Строка
<i>выражение2</i> или <i>выражение3</i>	Число с плавающей точкой
<i>выражение2</i> или <i>выражение3</i>	Целое число

Если *выражение2* и *выражение3* являются строками, результат чувствителен к регистру, если любая из строк чувствительна к регистру (начиная с MySQL 3.23.51).

■ **IFNULL(выражение1, выражение2)**

Если *выражение1* не равно NULL, IFNULL() возвратит *выражение1*, иначе *выражение2*. IFNULL() возвращает число или строку, в зависимости от контекста, в котором вызывается.

```
mysql> SELECT IFNULL(1,0);
-> 1
mysql> SELECT IFNULL(NULL,10);
-> 10
mysql> SELECT IFNULL(1/0,10);
-> 10
mysql> SELECT IFNULL(1/0,'yes');
-> 'yes'
```

В MySQL 4.0.6 и выше возвращаемое IFNULL(*выражение1*, *выражение2*) значение по умолчанию определяется, как более “общее” из двух, в таком порядке: STRING, REAL или INTEGER. Отличие от предшествующих версий MySQL наиболее ярко проявляется, когда вы создаете таблицу на базе выражений, либо когда MySQL должен скрыто сохранять IFNULL() во временной таблице.

```
CREATE TABLE tmp SELECT IFNULL(1, 'test') AS test;
```

Начиная с MySQL 4.0.6, типом столбца test будет CHAR(4), в то время как в более ранних версиях типом будет BIGINT.

■ **NULLIF(*выражение1*, *выражение2*)**

Возвращает NULL, если *выражение1*=*выражение2*, иначе возвращает *выражение1*. Это то же самое, что CASE WHEN *выражение1*=*выражение2* THEN NULL ELSE *выражение1* END.

```
mysql> SELECT NULLIF(1,1);  
-> NULL  
mysql> SELECT NULLIF(1,2);  
-> 1
```

Следует отметить, что MySQL дважды вычисляет выражение *выражение1*, если аргументы не равны.

Функция NULLIF() появилась в MySQL 3.23.15.

## 5.3. Строковые функции

Строковые функции возвращают NULL, если длина результата превысит значение системной переменной `max_allowed_packet`. Для функций, которые оперируют позициями в строке, нумерация позиций начинается с 1.

■ **ASCII(*строка*)**. Возвращает числовое значение первого символа строки *строка*. Возвращает 0, если *строка* является пустой. Возвращает NULL, если *строка* равна NULL. ASCII() работает с символами в диапазоне кодов от 0 до 255.

```
mysql> SELECT ASCII('2');  
-> 50  
mysql> SELECT ASCII(2);  
-> 50  
mysql> SELECT ASCII('dx');  
-> 100
```

См. также функцию ORD().

■ **BIN(*N*)**. Возвращает строковое представление двоичного значения *N*, где *N* – длинное целое (BIGINT). Это эквивалентно CONV(*N*, 10, 2). Возвращает NULL, если *N* равно NULL.

```
mysql> SELECT BIN(12);  
-> '1100'
```

■ **BIT\_LENGTH(*строка*)**. Возвращает длину строки *строка* в битах.

```
mysql> SELECT BIT_LENGTH('text');  
-> 32
```

Функция BIT\_LENGTH() была добавлена в MySQL 4.0.2.

- **CHAR(N, ...)**. Интерпретирует аргументы как целые и возвращает строку, состоящую из символов с кодами, заданными этими целыми. Значение NULL пропускаются.

```
mysql> SELECT CHAR(77,121,83,81,'76');  
-> 'MySQL'  
mysql> SELECT CHAR(77,77.3,'77.3');  
-> 'MMM'
```

- **CHAR\_LENGTH(строка)**. Возвращает длину строки *строка*, измеренную в символах. Многобайтные символы считаются как один. Это значит, что для строки, состоящей из пяти двухбайтных символов, **LENGTH()** вернет 10, в то время как **CHAR\_LENGTH()** – 5.

- **CHARACTER\_LENGTH(строка)**

**CHARACTER\_LENGTH()** – это синоним **CHAR\_LENGTH()**.

- **COMPRESS(строка\_для\_сжатия)**. Сжимает строку. Эта функция требует, чтобы MySQL был скомпилирован с библиотекой поддержки сжатия, такой как **zlib**. В противном случае возвращаемым значением всегда будет NULL.

```
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',1000)));  
-> 21  
mysql> SELECT LENGTH(COMPRESS(''));  
-> 0  
mysql> SELECT LENGTH(COMPRESS('a'));  
-> 13  
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',16)));  
-> 15
```

Содержимое сжатой строки сохраняется следующим образом:

- Пустая строка сохраняется как пустая строка.
- Непустая строка сохраняется как четырехбайтовая длина несжатой строки (младший байт идет первым), за которой следует сжатая строка. Если строка завершается пробелом, добавляется дополнительный символ '.' во избежание усечения завершающих пробелов, которое имеет место при сохранении в столбцах **CHAR** или **VARCHAR**. (Использовать для сохранения сжатых строк столбцы **CHAR** или **VARCHAR** не рекомендуется. Взамен лучше применять столбцы **BLOB**.)

Функция **COMPRESS()** появилась в MySQL 4.1.1.

- **CONCAT(строка1, строка2, ...)**. Возвращает строку, которая состоит из сцепленных аргументов. Возвращает NULL, если любой из аргументов равен NULL. Принимает один или более аргументов. Числовой аргумент преобразуется в эквивалентную строковую форму.

```
mysql> SELECT CONCAT('My', 'S', 'QL');  
-> 'MySQL'  
mysql> SELECT CONCAT('My', NULL, 'QL');  
-> NULL  
mysql> SELECT CONCAT(14.3);  
-> '14.3'
```

■ **CONCAT\_WS** (*разделитель*, *строка1*, *строка2*, ...)

CONCAT\_WS означает "Concat With Separator" ("CONCAT с разделителем") и представляет собой особую форму CONCAT(). Первый аргумент – это разделитель для остальных аргументов. Разделитель добавляется между соединяемыми строками. Разделитель может быть строкой, как и остальные аргументы. Если разделитель равен NULL, результат тоже равен NULL. Функция пропускает любые аргументы NULL после разделителя.

```
mysql> SELECT CONCAT_WS(',', 'First name', 'Second name', 'Last Name');  
-> 'First name,Second name,Last Name'  
mysql> SELECT CONCAT_WS(',', 'First name', NULL, 'Last Name');  
-> 'First name,Last Name'
```

До версии MySQL 4.0.14 функция CONCAT\_WS() пропускала пустые строки, так же как и значения NULL.

■ **CONV** (*N*, *основание\_начальное*, *основание\_конечное*)

Конвертирует числа между разными системами счисления. Возвращает строковое представление числа *N*, преобразованное из системы счисления с основанием *основание\_начальное* в систему счисления с основанием *основание\_конечное*. Возвращает NULL, если любой из аргументов равен NULL. Аргумент *N* интерпретируется как целое, но может указываться и как целое, и как строка. Минимальное основание системы счисления – 2, максимальное – 36. Если значение *основание\_конечное* отрицательное, *N* рассматривается как целое со знаком. В противном случае *N* считается беззнаковым целым. CONV() работает с 64-разрядной точностью.

```
mysql> SELECT CONV('a', 16, 2);  
-> '1010'  
mysql> SELECT CONV('6E', 18, 8);  
-> '172'  
mysql> SELECT CONV(-17, 10, -18);  
-> '-H'  
mysql> SELECT CONV(10+'10'+ '10'+0xa, 10, 10);  
-> '40'
```

■ **ELT** (*N*, *строка1*, *строка2*, *строка3*, ...)

Возвращает *строка1*, если *N* = 1, *строка2*, если *N* = 2, и так далее. Возвращает NULL, если *N* меньше 1 или больше количества аргументов. ELT() – это дополнение FIELD().

```
mysql> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');  
-> 'ej'  
mysql> SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');  
-> 'foo'
```

■ **EXPORT\_SET** (*биты*, *вкл*, *выкл*, [*разделитель*], [*количество\_бит*])

Возвращает строку, в которой для каждого установленного в 1 бита в аргументе *биты* возвращается строка *вкл*, а для каждого бита, установленного в 0, – строка *выкл*. Каждая строка отделяется разделителем *разделитель* (по умолчанию – ','), и используются только *количество\_бит* бит (по умолчанию 64).

```
mysql> SELECT EXPORT_SET(5, 'Y', 'N', ',', ',4')  
-> Y,N,Y,N
```

- **FIELD(строка, строка1, строка2, строка3, ...)**. Возвращает позицию вхождения аргумента *строка* в список *строка1, строка2, строка3, ...*. Возвращает 0, если вхождение не найдено. **FIELD()** – это дополнение **ELT()**.

```
mysql> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
-> 2
```

```
mysql> SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
-> 0
```

- **FIND\_IN\_SET(строка, список\_строк)**. Возвращает значение от 1 до *N*, если строка находится в списке строк *список\_строк*, состоящего из *N* подстрок. Список строк – это строка, состоящая из подстрок, разделенных символом ','. Если первый аргумент – константная строка, а второй – столбец типа SET, функция **FIND\_IN\_SET()** оптимизирована для использования битовой арифметики. Возвращает 0, если строка не входит в список строк, или если *список\_строк* – пустая строка. Возвращает NULL, если любой из аргументов равен NULL. Эта функция не работает правильно, если первый аргумент содержит запятую.

```
mysql> SELECT FIND_IN_SET('b', 'a,b,c,d');
-> 2
```

- **HEX(Ч или С)**. Если *Ч или С* – число, возвращает строковое представление шестнадцатеричного значения *N*, где *N* – длинное целое (BIGINT). Это эквивалентно **CONV(N, 10, 16)**.

Начиная с версии MySQL 4.0.1 и выше, если *Ч или С* – строка, то возвращается шестнадцатеричная строка *Ч или С*, в которой каждый символ преобразован в два шестнадцатеричных разряда.

```
mysql> SELECT HEX(255);
-> 'FF'
```

```
mysql> SELECT HEX(0x616263);
-> 'abc'
```

```
mysql> SELECT HEX('abc');
-> 616263
```

- **INSERT(строка, позиция, длина, новая\_строка)**. Возвращает строку *строка*, в которой подстрока длиной *длина*, начинающаяся с позиции *позиция*, заменяется строкой *новая\_строка*.

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
-> 'QuWhattic'
```

Эта функция безопасна в отношении многобайтных наборов символов.

- **INSTR(строка, подстрока)**. Возвращает позицию первого вхождения подстроки подстрока в строку строка. Это то же самое, что двухаргументная форма **LOCATE()**, только аргументы переставлены местами.

```
mysql> SELECT INSTR('foobarbar', 'bar');
-> 4
```

```
mysql> SELECT INSTR('xbar', 'foobar');
-> 0
```

Эта функция безопасна в отношении многобайтных наборов символов. В MySQL 3.23 эта функция чувствительна к регистру. В 4.0 она чувствительна к регистру, только если любой аргументов является бинарной строкой.



- **LCASE(строка)**. Функция **LCASE()** – синоним для **LOWER()**.
- **LEFT(строка, длина)**. Возвращает левые длина символов строки строка.  

```
mysql> SELECT LEFT('foobarbar', 5);  
-> 'fooba'
```
- **LENGTH(строка)**. Возвращает длину строки строка в байтах. Многобайтные символы считаются по количеству байт. Это значит, что для строки, содержащей пять двухбайтных символов, **LENGTH()** вернет 10, в то время как **CHAR\_LENGTH()** – 5.  

```
mysql> SELECT LENGTH('text');  
-> 4
```

- **LOAD\_FILE(имя\_файла)**. Читает файл и возвращает его содержимое в виде строки. Файл должен находиться на сервере и к нему должен указываться полный путь. Кроме того, необходимо иметь привилегию **FILE**. Файл должен быть доступен по чтению всем, и иметь размер менее `max_allowed_packet` байт.

Если файл не существует или не может быть прочитан, функция возвращает **NULL**.

```
mysql> UPDATE имя_файла  
      SET столбец_blob=LOAD_FILE('/tmp/picture')  
      WHERE id=1;
```

До версии MySQL 3.23 вы должны были читать файл внутри приложения и создавать оператор **INSERT** для обновления базы данных содержимым файла. Если вы используете библиотеку MySQL++, единственный способ сделать это описан в руководстве по MySQL++, которое доступно по адресу <http://dev.mysql.com/doc>.

- **LOCATE(подстрока, строка)**  
**LOCATE(подстрока, строка, позиция)**

Первый синтаксис возвращает позицию первого вхождения подстроки подстрока в строку строка. Второй синтаксис возвращает позицию первого вхождения подстроки подстрока в строку строка, начиная с позиции позиция. Если подстрока не входит в строку строка, возвращается 0.

```
mysql> SELECT LOCATE('bar', 'foobarbar');  
-> 4  
mysql> SELECT LOCATE('xbar', 'foobar');  
-> 0  
mysql> SELECT LOCATE('bar', 'foobarbar', 5);  
-> 7
```

Эта функция безопасна в отношении многобайтных наборов символов. В MySQL 3.23 эта функция чувствительна к регистру. В 4.0 она чувствительна к регистру, только если любой из аргументов является бинарной строкой.

- **LOWER(строка)**. Возвращает строку строка, в которой все символы приведены к нижнему регистру в соответствии с текущим набором символов (по умолчанию ISO-8859-1 Latin1).  

```
mysql> SELECT LOWER('QUADRATICALLY');  
-> 'quadratically'
```

Функция безопасна в отношении многобайтных наборов символов.

- **LPAD(строка, длина, строка-заполнитель)**. Возвращает строку *строка*, добавив слева строкой *строка-заполнитель* до длины *длина*. Если *строка* длиннее, чем указано в аргументе *длина*, возвращается значение, усеченное до *длина* символов.

```
mysql> SELECT LPAD('hi',4,'??');
-> '??hi'
mysql> SELECT LPAD('hi',1,'??');
-> 'h'
```

- **LTRIM(строка)**. Возвращает строку *строка* с удаленными ведущими пробелами.

```
mysql> SELECT LTRIM(' barbar');
-> 'barbar'
```

Функция безопасна в отношении многобайтных наборов символов.

- **MAKE\_SET(биты, строка1, строка2, ...)**. Возвращает набор (строку, содержащую подстроки, разделенные запятой), состоящий из строк, которые имеют соответствующую установку битов в *биты*. При этом *строка1* соответствует нулевому (по порядку) биту, *строка2* – первому и так далее. Значения NULL в списке *строка1, строка2, ...* к результату не добавляются.

```
mysql> SELECT MAKE_SET(1,'a','b','c');
-> 'a'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
-> 'hello,world'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world');
-> 'hello'
mysql> SELECT MAKE_SET(0,'a','b','c');
-> ''
```

- **MID(строка, позиция, длина)**

**MID(строка, позиция, длина)** – это синоним для **SUBSTR(строка, позиция, длина)**.

- **OCT(N)**. Возвращает строковое представление восьмеричного значения *N*, где *N* – длинное целое.

Это эквивалент **CONV(N, 10, 8)**. Возвращает NULL, если *N* равно NULL.

```
mysql> SELECT OCT(12);
-> '14'
```

- **OCTET\_LENGTH(строка)**

**OCTET\_LENGTH()** – это синоним для **LENGTH()**.

- **ORD(строка)**. Если самый левый символ строки *строка* многобайтный, возвращается код этого символа, вычисленный из числовых значений байтов, из которых он состоит, с использованием формулы:

$$\begin{aligned} & (\text{код 1-го байта} * 256) \\ & + (\text{код 2-го байта} * 256^2) \\ & + (\text{код 3-го байта} * 256^3) \dots \end{aligned}$$

Если же самый левый символ строки *строка* не многобайтный, то **ORD()** возвращает то же значение, что и функция **ASCII()**.

```
mysql> SELECT ORD('2');
-> 50
```

- **POSITION**(*подстрока* IN *строка*)

**POSITION**(*подстрока* IN *строка*) — это синоним для **LOCATE**(*подстрока*, *строка*).

- **QUOTE**(*строка*). Заключает строку в кавычки, чтобы результат можно было использовать как допустимое значение в SQL-операторах. Строка окружается одинарными кавычками, а все вхождения в нее одинарной кавычки ("'"), обратной косой черты ("\"), ASCII NUL и Control-Z предваряются обратной косой чертой. Если аргумент равен NULL, возвращается слово "NULL" без кавычек. Функция **QUOTE**() появилась в версии MySQL 4.0.3.

```
mysql> SELECT QUOTE('Don\'t!');
-> 'Don\'t!'
```

```
mysql> SELECT QUOTE(NULL);
-> NULL
```

- **REPEAT**(*строка*, *количество*). Возвращает строку, состоящую из аргумента *строка*, повторенного *количество* раз. Если *количество* < 0, возвращается пустая строка. Возвращает NULL, если *строка* или *количество* равно NULL.

```
mysql> SELECT REPEAT('MySQL', 3);
-> 'MySQLMySQLMySQL'
```

- **REPLACE**(*строка*, *строка\_с*, *строка\_в*). Возвращает строку *строка*, в которой все вхождения *строка\_с* заменены на *строка\_в*.

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
-> 'WwWwWw.mysql.com'
```

Функция безопасна в отношении многобайтных наборов символов.

- **REVERSE**(*строка*). Возвращает строку *строка* с обратным порядком символов.

```
mysql> SELECT REVERSE('abc');
-> 'cba'
```

Функция безопасна в отношении многобайтных наборов символов.

- **RIGHT**(*строка*, *длина*). Возвращает *длина* правых символов строки *строка*.

```
mysql> SELECT RIGHT('foobarbar', 4);
-> 'rbar'
```

Функция безопасна в отношении многобайтных наборов символов.

- **RPAD**(*строка*, *длина*, *строка-заполнитель*). Возвращает строку *строка*, дополненную справа строкой *строка-заполнитель* до длины *длина*. Если *строка* длиннее, чем *длина*, возвращается значение, усеченное до *длина* символов.

```
mysql> SELECT RPAD('hi', 5, '???');
-> 'hi???'
```

```
mysql> SELECT RPAD('hi', 1, '?');
-> 'h'
```

Функция безопасна в отношении многобайтных наборов символов.

- **RTRIM**(*строка*). Возвращает строку *строка* с удаленными завершающими пробелами.

```
mysql> SELECT RTRIM('barbar ');
-> 'barbar'
```

Функция безопасна в отношении многобайтных наборов символов.

- **SOUNDEX(строка)**. Возвращает строку, описывающую звучание (soundex) строки *строка*. Две строки, которые произносятся почти одинаково, должны иметь идентичные строки звучания. Стандартная строка звучания состоит из четырех символов, однако функция SOUNDEX() возвращает строку любой длины. С помощью функции SUBSTRING(), примененной к результату, можно получить стандартную строку звучания. Все небуквенные символы игнорируются. Все интернациональные символы вне диапазона A-Z, представляются гласными.

```
mysql> SELECT SOUNDEX('Hello');
-> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
-> 'Q36324'
```

### На заметку!

Эта функция реализует оригинальный Soundex-алгоритм, а не более популярную расширенную версию (также описанную Дональдом Кнутом). Разница заключается в том, что оригинальная версия сначала удаляет гласные, а затем дубли, в то время как расширенная версия сначала убирает дубли, а затем гласные.

- **выражение1 SOUNDS LIKE выражение2**. Это то же самое, что SOUNDEX(выражение1) = SOUNDEX(выражение2). Доступно только в MySQL 4.1 и выше.

- **SPACE(N)**. Возвращает строку, состоящую из N пробелов.

```
mysql> SELECT SPACE(6);
-> '      '
```

- **SUBSTRING(строка, позиция)**  
**SUBSTRING(строка FROM позиция)**  
**SUBSTRING(строка, позиция, длина)**  
**SUBSTRING(строка FROM позиция FOR длина)**

Формы без аргумента *длина* возвращают подстроку строки *строка*, начиная с позиции *позиция*. Формы с аргументом *длина* возвращают подстроку строки *строка* длиной *длина* символов, начиная с позиции *позиция*. Формы, использующие FROM, представляют стандартный синтаксис SQL.

```
mysql> SELECT SUBSTRING('Quadratically',5);
-> 'ratically'
mysql> SELECT SUBSTRING('foobarbar' FROM 4);
-> 'barbar'
mysql> SELECT SUBSTRING('Quadratically',5,6);
-> 'ratica'
```

Функция безопасна в отношении многобайтных наборов символов.

- **SUBSTRING\_INDEX(строка, разделитель, количество)**. Возвращает подстроку строки *строка* до вхождения номер *количество* разделителя *разделитель*. Если значение *количество* положительное, возвращается все, что лежит слева от финального разделителя (считая слева направо). Если значение *количество* отрицательное, возвращается все, что лежит справа от финального разделителя (считая справа налево).

```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
-> 'www.mysql'
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
-> 'mysql.com'
```

Функция безопасна в отношении многобайтных наборов символов.

- `TRIM([BOTH | LEADING | TRAILING] [удаляемая_строка] FROM строка)`

Возвращает строку *строка* с удаленными префиксами и/или суффиксами *удаляемая\_строка*. Если не указано ни `BOTH`, ни `LEADING`, ни `TRAILING`, подразумевается `BOTH`. Если не указано *удаляемая\_строка*, удаляются пробелы.

```
mysql> SELECT TRIM(' bar ');
-> 'bar'
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
-> 'barxxx'
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
-> 'bar'
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxyz');
-> 'barx'
```

Функция безопасна в отношении многобайтных наборов символов.

- `UCASE(строка)`

`UCASE()` — синоним для `UPPER()`.

- `UNCOMPRESS(строка_для_распаковки)`. Распаковывает строку, сжатую функцией `COMPRESS()`. Если аргумент не является упакованной строкой, возвращается `NULL`. Функция требует, чтобы MySQL был скомпилирован с библиотекой сжатия, такой как `zlib`. В противном случае возвращаемое значение всегда равно `NULL`.

```
mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
-> 'any string'
mysql> SELECT UNCOMPRESS('any string');
-> NULL
```

Функция `UNCOMPRESS()` появилась в версии MySQL 4.1.1.

- `UNCOMPRESSED_LENGTH(строка_для_распаковки)`. Возвращает длину строки перед сжатием.

```
mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));
-> 30
```

Функция `UNCOMPRESSED_LENGTH()` была добавлена в MySQL 4.1.1.

- `UNHEX(строка)`

Выполняет действие, противоположное функции `HEX(строка)`. То есть, интерпретирует каждую пару шестнадцатеричных цифр аргумента как число и преобразует в символ, представленный этим числом. Символы результата возвращаются в виде бинарной строки.

```
mysql> SELECT UNHEX('4D7953514C');
-> 'MySQL'
mysql> SELECT 0x4D7953514C;
-> 'MySQL'
mysql> SELECT UNHEX(HEX('string'));
-> 'string'
mysql> SELECT HEX(UNHEX('1267'));
-> '1267'
```

Функция `UNHEX()` появилась в версии MySQL 4.1.2.

■ **UPPER(строка)**

Возвращает строку *строка*, у которой все символы приведены к верхнему регистру в соответствии с текущим набором символов (по умолчанию ISO-8859-1 Latin1).

```
mysql> SELECT UPPER('Hej');
-> 'HEJ'
```

Функция безопасна в отношении многобайтных наборов символов.

### 5.3.1. Функции сравнения строк

MySQL при необходимости автоматически преобразует числа в строки и обратно.

```
mysql> SELECT 1+'1';
-> 2
mysql> SELECT CONCAT(2, ' test');
-> '2 test'
```

Если требуется преобразовать число в строку явно, воспользуйтесь для этого функцией CAST() или CONCAT():

```
mysql> SELECT 38.8, CAST(38.8 AS CHAR);
-> 38.8, '38.8'
mysql> SELECT 38.8, CONCAT(38.8);
-> 38.8, '38.8'
```

Функция CAST() предпочтительнее, однако она недоступна в версиях MySQL, предшествующих 4.0.2.

Если строковой функции в качестве аргумента передана бинарная строка, результирующая строка также будет бинарной. Это касается только сравнений.

Обычно если любое выражение в сравнении строк чувствительно к регистру, то сравнение также чувствительно к регистру.

■ **выражение LIKE шаблон [ESCAPE 'символ-отмены']**

Проверка на соответствие шаблону, заданному простыми регулярными выражениями SQL. Возвращает 1 (TRUE) или 0 (FALSE). Если выражение или шаблон равны NULL, возвращает NULL.

В шаблонах LIKE можно использовать следующие два символа:

Символ	Описание
%	Соответствие любому числу символов, включая нуль символов.
_	Соответствие любому одному символу.

```
mysql> SELECT 'David!' LIKE 'David_';
-> 1
mysql> SELECT 'David!' LIKE '%D%v%';
-> 1
```

Чтобы протестировать литеральные вхождения шаблонных символов, предваряйте их символом отмены. Если конструкция ESCAPE не указана, предполагается '\'.

Строка	Описание
\%	Соответствует одиночному символу '%'.
\_	Соответствует одиночному символу '_'.

```
mysql> SELECT 'David!' LIKE 'David\_';
-> 0
mysql> SELECT 'David_' LIKE 'David\_';
-> 1
```

Чтобы указать другой символ отмены, используйте конструкцию ESCAPE:

```
mysql> SELECT 'David_' LIKE 'David|_' ESCAPE '|';
-> 1
```

Следующие два оператора иллюстрируют, что сравнение строк нечувствительно к регистру, если только хотя бы один из операндов не является бинарной строкой:

```
mysql> SELECT 'abc' LIKE 'ABC';
-> 1
mysql> SELECT 'abc' LIKE BINARY 'ABC';
-> 0
```

В MySQL LIKE допускает числовые выражения. (Это расширение LIKE из стандартного SQL).

```
mysql> SELECT 10 LIKE '1%';
-> 1
```

### На заметку!

Поскольку MySQL применяет синтаксис отмены языка C в строках (например, '\n' представляет перевод строки), вы должны дублировать любые символы '\', которые встречаются в строках LIKE. Например, чтобы искать '\n', следует указывать '\\n'. Для поиска '\' указывайте '\\\\' (обратная косая черта отсекается первый раз анализатором выражений и второй – когда готовится шаблон, в результате чего остается только одна обратная косая черта).

- выражение NOT LIKE шаблон [ESCAPE 'символ-отмены']

Это то же самое, что NOT (выражение LIKE шаблон [ESCAPE 'символ-отмены']).

- выражение NOT REGEXP шаблон  
выражение NOT RLIKE шаблон

Это то же самое, что NOT (выражение REGEXP шаблон).

- выражение REGEXP шаблон  
выражение RLIKE шаблон

Выполняет сравнение строкового выражения *выражение* с шаблоном *шаблон*. Шаблон может быть расширенным регулярным выражением. Синтаксис регулярных выражений рассматривается в приложении А. Возвращает 1, если выражение соответствует шаблону, иначе возвращает 0. Если либо *выражение*, либо *шаблон* равны NULL, результатом также будет NULL. RLIKE – это синоним REGEXP, добавленный для достижения совместимости с mSQL. Следует отметить, что поскольку MySQL применяет синтаксис отмены языка C в строках (например, '\n' представляет перевод строки), вы должны дублировать любые символы '\', которые встречаются в строках REGEXP. Начиная с версии MySQL 3.23.4, REGEXP не чувствителен к регистру обычных (не бинарных) строк.

```
mysql> SELECT 'Monty!' REGEXP 'mty%';
-> 0
mysql> SELECT 'Monty!' REGEXP '.*';
-> 1
mysql> SELECT 'new*n*line' REGEXP 'new\\.*\\.\\*line';
-> 1
```

```
mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';
-> 1 0
mysql> SELECT 'a' REGEXP '^[a-d]';
-> 1
```

REGEXP и RLIKE используют текущий набор символов (по умолчанию ISO-8859-1 Latin1), однако эти операции не являются безопасными в отношении многобайтных наборов.

■ **STRCMP** (*выражение1*, *выражение2*)

STRCMP() возвращает 0, если строки идентичны, -1 – если первый аргумент меньше второго в соответствии с действующим порядком сопоставления, и 1 – в противном случае.

```
mysql> SELECT STRCMP('text', 'text2');
-> -1
mysql> SELECT STRCMP('text2', 'text');
-> 1
mysql> SELECT STRCMP('text', 'text');
-> 0
```

Начиная с версии MySQL 4.0, STRCMP() использует текущий набор символов при выполнении сравнений. Это делает поведение при сравнении строк нечувствительным к регистру, если только один или оба аргумента не являются бинарными строками. До версии MySQL 4.0 функция STRCMP() была чувствительной к регистру.

## 5.4. Числовые функции

### 5.4.1. Арифметические операции

Доступны обычные арифметические операции. Помните, что для операций -, + и \* результат вычисляется с точностью BIGINT (64-разрядной), если оба аргумента – целые. Если один из аргументов – беззнаковое целое, а другой – также целое, результатом будет беззнаковое целое. См. раздел 5.7.

■ **+**. Сложение:

```
mysql> SELECT 3+5;
-> 8
```

■ **-**. Вычитание:

```
mysql> SELECT 3-5;
-> -2
```

■ **-**. Унарный минус, меняет знак аргумента:

```
mysql> SELECT - 2;
-> -2
```

Помните, что если операция используется для BIGINT, возвращаемое значение будет иметь тип BIGINT. Это значит, что вы должны избегать применения операции “-” с целыми, которые могут принимать значение -263.

■ **\***. Умножение:

```
mysql> SELECT 3*5;
-> 15
```



```
mysql> SELECT 18014398509481984*18014398509481984.0;
-> 324518553658426726783156020576256.0
mysql> SELECT 18014398509481984*18014398509481984;
-> 0
```

Результат последнего выражения неправильный, потому что результат умножения превысил диапазон допустимых значений 64-разрядных BIGINT.

■ /. Деление:

```
mysql> SELECT 3/5;
-> 0.60
```

Деление на ноль дает в результате NULL:

```
mysql> SELECT 102/(1-1);
-> NULL
```

Деление выполняется с использованием арифметики BIGINT, только если это делается в контексте, где результат преобразуется с целое.

■ DIV. Целочисленное деление. Похоже на FLOOR(), но безопасно в отношении значений BIGINT.

```
mysql> SELECT 5 DIV 2;
-> 2
```

Операция DIV появилась в версии MySQL 4.1.0.

## 5.4.2. Математические функции

Все математические функции в случае ошибки возвращают NULL.

■ ABS(X). Возвращает абсолютное значение X.

```
mysql> SELECT ABS(2);
-> 2
mysql> SELECT ABS(-32);
-> 32
```

Безопасна для применения со значениями BIGINT.

■ ACOS(X). Возвращает арккосинус X, то есть значение, косинус которого равен X. Если значение X лежит вне диапазона от -1 до 1, возвращается NULL.

```
mysql> SELECT ACOS(1);
-> 0.000000
mysql> SELECT ACOS(1.0001);
-> NULL
mysql> SELECT ACOS(0);
-> 1.570796
```

■ ASIN(X). Возвращает арксинус X, то есть значение, синус которого равен X. Если значение X лежит вне диапазона от -1 до 1, возвращается NULL.

```
mysql> SELECT ASIN(0.2);
-> 0.201358
mysql> SELECT ASIN('foo');
-> 0.000000
```

- **ATAN(*X*)**. Возвращает арктангенс *X*, то есть значение, тангенс которого равен *X*.

```
mysql> SELECT ATAN(2);  
-> 1.107149  
mysql> SELECT ATAN(-2);  
-> -1.107149
```

- **ATAN(*X*, *Y*)**  
**ATAN2(*X*, *Y*)**

Возвращает арктангенс двух переменных *X* и *Y*. Это подобно вычислению арктангенса  $X/Y$ , за исключением того, что знаки обоих аргументов используются для определения квадранта результата.

```
mysql> SELECT ATAN(-2,2);  
-> -0.785398  
mysql> SELECT ATAN2(PI(),0);  
-> 1.570796
```

- **CEILING(*X*)**  
**CEIL(*X*)**

Возвращает наименьшее целое значение, которое не меньше *X*.

```
mysql> SELECT CEILING(1.23);  
-> 2  
mysql> SELECT CEIL(-1.23);  
-> -1
```

Следует отметить, что возвращаемое значение преобразуется к типу **BIGINT**.

Псевдоним **CEIL()** был добавлен в MySQL 4.0.6.

- **COS(*X*)**. Возвращает косинус *X*, где *X* задан в радианах.

```
mysql> SELECT COS(PI());  
-> -1.000000
```

- **COT(*X*)**. Возвращает котангенс *X*.

```
mysql> SELECT COT(12);  
-> -1.57267341  
mysql> SELECT COT(0);  
-> NULL
```

- **CRC32(*выражение*)**. Вычисляет проверочное значение в циклическом избыточном коде и возвращает 32-разрядное целое. Результат равен **NULL**, если передается аргумент **NULL**. Ожидается, что аргумент будет строкой, и будет рассматриваться в качестве таковой в противном случае.

```
mysql> SELECT CRC32('MySQL');  
-> 3259397556
```

**CRC32()** была добавлена в MySQL 4.1.0.

- **DEGREES(*X*)**

Возвращает аргумент *X*, преобразованный из радианов в градусы.

```
mysql> SELECT DEGREES(PI());  
-> 180.000000
```

- **EXP(X)**. Возвращает значение числа  $e$  (основания натурального логарифма), возведенное в степень  $X$ .

```
mysql> SELECT EXP(2);  
-> 7.389056  
mysql> SELECT EXP(-2);  
-> 0.135335
```

- **FLOOR(X)**. Возвращает максимальное целое число, не большее  $X$ .

```
mysql> SELECT FLOOR(1.23);  
-> 1  
mysql> SELECT FLOOR(-1.23);  
-> -2
```

Следует отметить, что возвращаемое значение преобразуется к типу **BIGINT**.

- **LN(X)**. Возвращает натуральный логарифм  $X$ .

```
mysql> SELECT LN(2);  
-> 0.693147  
mysql> SELECT LN(-2);  
-> NULL
```

Функция была добавлена в MySQL 4.0.3. Является синонимом **LOG(X)**.

- **LOG(X)**

**LOG(B, X)**

При вызове с одним параметром возвращает натуральный логарифм  $X$ .

```
mysql> SELECT LOG(2);  
-> 0.693147  
mysql> SELECT LOG(-2);  
-> NULL
```

При вызове с двумя параметрами возвращает логарифм  $X$  по основанию  $B$ .

```
mysql> SELECT LOG(2, 65536);  
-> 16.000000  
mysql> SELECT LOG(1, 100);  
-> NULL
```

Вариант функции с аргументом основания появился в MySQL 4.0.3.

**LOG(B, X)** эквивалентна **LOG(B) / LOG(X)**.

- **LOG2(X)**. Возвращает логарифм  $X$  по основанию 2.

```
mysql> SELECT LOG2(65536);  
-> 16.000000  
mysql> SELECT LOG2(-100);  
-> NULL
```

Функция **LOG2()** удобна для того, чтобы определить, сколько бит потребуется для сохранения числа. Эта функция была добавлена в MySQL 4.0.3. В более ранних версиях вместо нее можно использовать **LOG(X) / LOG(2)**.

- **LOG10(X)**. Возвращает логарифм  $X$  по основанию 10.

```
mysql> SELECT LOG10(2);  
-> 0.301030  
mysql> SELECT LOG10(100);  
-> 2.000000
```

```
mysql> SELECT LOG10(-100);  
-> NULL
```

■ MOD(N, M)

$N \% M$

$N \text{ MOD } M$

Модуль (подобен операции  $\%$  в языке C). Возвращает остаток от деления  $N$  на  $M$ .

```
mysql> SELECT MOD(234, 10);  
-> 4
```

```
mysql> SELECT 253 \% 7;  
-> 1
```

```
mysql> SELECT MOD(29, 9);  
-> 2
```

```
mysql> SELECT 29 MOD 9;  
-> 2
```

Эта функция безопасна для применения с типом BIGINT. Синтаксис  $N \text{ MOD } M$  работает только в версии MySQL 4.1.

- PI(). Возвращает значение числа  $\pi$ . По умолчанию отображается пять знаков после десятичной запятой, но внутренне MySQL использует полное представление действительного числа двойной точности.

```
mysql> SELECT PI();  
-> 3.141593
```

```
mysql> SELECT PI()+0.000000000000000000;  
-> 3.141592653589793116
```

■ POW(X, Y)

POWER(X, Y)

Возвращает значение  $X$ , возведенное в степень  $Y$ .

```
mysql> SELECT POW(2, 2);  
-> 4.000000
```

```
mysql> SELECT POW(2, -2);  
-> 0.250000
```

■ RADIANS(X)

Возвращает аргумент  $X$ , преобразованный из градусов в радианы.

```
mysql> SELECT RADIANS(90);  
-> 1.570796
```

■ RAND()

RAND(N)

Возвращает случайное число двойной точности в диапазоне от 0 до 1.0. Если указан целочисленный аргумент  $N$ , он служит начальным числом для генератора случайных чисел (генерируя повторяющуюся последовательность).

```
mysql> SELECT RAND();  
-> 0.9233482386203
```

```
mysql> SELECT RAND(20);  
-> 0.15888261251047
```

```
mysql> SELECT RAND(20);  
-> 0.15888261251047
```

```
mysql> SELECT RAND();  
-> 0.63553050033332  
mysql> SELECT RAND();  
-> 0.70100469486881
```

Вы не можете указывать столбец со значениями RAND() в конструкции ORDER BY, поскольку ORDER BY оценивает столбец множество раз. Начиная с версии MySQL 3.23, появилась возможность извлекать строки в случайном порядке следующим образом:

```
mysql> SELECT * FROM имя_таблицы ORDER BY RAND();
```

ORDER BY RAND() в комбинации с LIMIT удобно для выбора случайного примера из набора строк:

```
mysql> SELECT * FROM table1, table2 WHERE a=b AND c<d  
-> ORDER BY RAND() LIMIT 1000;
```

Следует отметить, что RAND() в конструкции WHERE вычисляется заново при каждом выполнении WHERE.

Функция RAND() не является безупречным генератором случайных чисел, тем не менее, это быстрое и переносимое между платформами решение для одной и той же версии MySQL.

■ **ROUND(X)**  
**ROUND(X, D)**

Возвращает аргумент X, округленный до ближайшего целого. Если вызывается с двумя аргументами, округляется до D разрядов. Если D отрицательное, обнуляется целая часть числа.

```
mysql> SELECT ROUND(-1.23);  
-> -1  
mysql> SELECT ROUND(-1.58);  
-> -2  
mysql> SELECT ROUND(1.58);  
-> 2  
mysql> SELECT ROUND(1.298, 1);  
-> 1.3  
mysql> SELECT ROUND(1.298, 0);  
-> 1  
mysql> SELECT ROUND(23.298, -1);  
-> 20
```

Следует отметить, что поведение ROUND(), когда аргумент точно на середине отрезка между двумя целыми зависит от реализации библиотеки C. Различные реализации округляют до ближайшего четного, либо всегда в большую сторону, либо всегда в меньшую сторону, либо в сторону ближайшего нуля. Если вам нужно иметь предсказуемое поведение в этом случае, применяйте вместо этой функции TRUNCATE() или FLOOR().

■ **SIGN(X)**. Возвращает знак аргумента как -1, 0 или 1, в зависимости от того, X отрицательное, нуль или положительное.

```
mysql> SELECT SIGN(-32);  
-> -1
```

```
mysql> SELECT SIGN(0);  
-> 0  
mysql> SELECT SIGN(234);  
-> 1
```

- **SIN(X)**. Возвращает синус  $X$ , где  $X$  задан в радианах.

```
mysql> SELECT SIN(PI());  
-> 0.000000
```

- **SQRT(X)**. Возвращает неотрицательный квадратный корень из  $X$ .

```
mysql> SELECT SQRT(4);  
-> 2.000000  
mysql> SELECT SQRT(20);  
-> 4.472136
```

- **TAN(X)**. Возвращает тангенс  $X$ , где  $X$  задан в радианах.

```
mysql> SELECT TAN(PI()+1);  
-> 1.557408
```

- **TRUNCATE(X, D)**. Возвращает число  $X$  с дробной частью, усеченной до  $D$  десятичных разрядов. Если  $D$  равно 0, результат не имеет точки и дробной части. Если  $D$  отрицательное, целая часть числа длиной  $D$  обнуляется.

```
mysql> SELECT TRUNCATE(1.223,1);  
-> 1.2  
mysql> SELECT TRUNCATE(1.999,1);  
-> 1.9  
mysql> SELECT TRUNCATE(1.999,0);  
-> 1  
mysql> SELECT TRUNCATE(-1.999,1);  
-> -1.9  
mysql> SELECT TRUNCATE(122,-2);  
-> 100
```

Начиная с MySQL 3.23.51, все числа округляются в сторону нуля.

Следует отметить, что десятичные числа обычно не хранятся в компьютерах именно в виде чисел, а в виде двоичных значений двойной точности, поэтому иногда результат может вызвать удивление:

```
mysql> SELECT TRUNCATE(10.28*100,0);  
-> 1027
```

Это происходит потому, что 10.28 на самом деле сохраняется как что-нибудь вроде 10.2799999999999999.

## 5.5. Функции даты и времени

В этом разделе описаны функции, которые могут использоваться для манипуляции значениями времени. В разделе 4.3 представлены диапазоны допустимых значений каждого типа даты и времени, а также правильные форматы, в которых они могут отображаться.

Ниже приведен пример использования функций даты. Запрос выбирает все записи, у которых значение `столбец_date` находится в пределах последних 30 дней:

```
mysql> SELECT something FROM имя_таблицы
-> WHERE DATE_SUB(CURDATE(), INTERVAL 30 DAY) <= столбец_date;
```

Следует отметить, что этот запрос также выберет записи с датами, относящимися к будущему.

Функции, которые ожидают в аргументе значения даты, обычно принимают значения типа даты и времени, при этом игнорируя время. Функции, принимающие аргумент типа времени, принимают значения типа даты и времени, отбрасывая дату.

Функции, которые возвращают текущую дату или время, вычисляются только один раз на запрос, в начале его выполнения. Это значит, что множественные обращения к такой функции, как NOW(), внутри отдельного запроса всегда генерируют один и тот же результат. Этот принцип также справедлив и в отношении CURDATE(), CURTIME(), UTC\_DATE(), UTC\_TIME(), UTC\_TIMESTAMP() и многих других синонимов.

Диапазоны возвращаемых значений последующих описаний функций касаются полных дат. Если дата имеет “нулевое” либо неполное значение, вроде '2001-11-00', функции, извлекающие часть даты, могут вернуть 0. Например, DAYOFMONTH('2001-11-00') возвращает 0.

- ADDDATE(дата, INTERVAL выражение тип)  
ADDDATE(выражение, дни)

При вызове со вторым аргументом в форме INTERVAL функция ADDDATE() является синонимом DATE\_ADD(). Родственная функция SUBDATE() — это синоним DATE\_SUB(). Информацию об аргументах INTERVAL можно найти в описании DATE\_ADD().

```
mysql> SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY);
-> '1998-02-02'
mysql> SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY);
-> '1998-02-02'
```

Начиная с MySQL 4.1.1, доступен второй синтаксис, где выражение — выражение типа дата или дата-время, а дни — количество дней, которые необходимо добавить к выражению.

```
mysql> SELECT ADDDATE('1998-01-02', 31);
-> '1998-02-02'
```

- ADDTIME(выражение, выражение2)

ADDTIME() добавляет выражение2 к выражение и возвращает результат. выражение представляет собой выражение типа даты или даты-времени, а выражение2 — типа времени.

```
mysql> SELECT 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');
-> '03:00:01.999997'
```

ADDTIME() появилась в MySQL 4.1.1.

- CURDATE()

Возвращает текущую дату в формате 'ГГГГ-ММ-ДД' или ГГГГММДД, в зависимости от контекста, в котором вызывается функция — строковый или числовой.

```
mysql> SELECT CURDATE();
-> '1997-12-15'
```

```
mysql> SELECT CURDATE() + 0;
-> 19971215
```

■ CURRENT\_DATE, CURRENT\_DATE()

CURRENT\_DATE и CURRENT\_DATE() — это синонимы CURDATE().

■ CURTIME(). Возвращает текущее время в формате 'ЧЧ:ММ:СС' или ЧЧММСС, в зависимости от контекста, в котором вызывается функция — строковый или числовой.

```
mysql> SELECT CURTIME();
-> '23:50:26'
```

```
mysql> SELECT CURTIME() + 0;
-> 235026
```

■ CURRENT\_TIME, CURRENT\_TIME()

CURRENT\_TIME и CURRENT\_TIME() — это синонимы CURTIME().

■ CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP()

CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP() — это синонимы NOW().

■ DATE(*выражение*). Извлекает дату из выражения *выражение* типа даты или даты-времени.

■ DATEDIFF(*выражение*, *выражение2*)

DATEDIFF() возвращает количество дней между начальной датой *выражение* и конечной датой *выражение2*. *выражение* и *выражение2* — это выражения типа даты или даты-времени. В вычислениях участвует только дата.

```
mysql> SELECT DATEDIFF('1997-12-31 23:59:59', '1997-12-30');
-> 1
```

```
mysql> SELECT DATEDIFF('1997-11-30 23:59:59', '1997-12-31');
-> -31
```

DATEDIFF() появилась в MySQL 4.1.1.

■ DATE\_ADD(*дата*, INTERVAL *выражение* *тип*)

DATE\_SUB(*дата*, INTERVAL *выражение* *тип*)

Эти функции реализуют арифметику дат. *дата* — это значение типа DATE или DATETIME, задающее начальную дату. *выражение* — выражение, задающее величину интервала, который нужно добавить или вычесть из начальной даты. *выражение* — это строка, которая может начинаться с '-', чтобы задавать отрицательные интервалы. *тип* — ключевое слово, задающее, как должно интерпретироваться выражение.

Ключевое слово INTERVAL и спецификатор *тип* нечувствительны к регистру.

В табл. 5.1 показано, как связаны аргументы *тип* и *выражение*.

Значения аргумента *тип* DAY\_MICROSECOND, HOUR\_MICROSECOND, MINUTE\_MICROSECOND, SECOND\_MICROSECOND и MICROSECOND доступны, начиная с версии MySQL 4.1.1. Значения QUARTER и WEEK появились в MySQL 5.0.0.

MySQL допускает любую пунктуацию разделителей в формате *выражение*. Та, что приведена в таблице, служит исключительно для примера. Если аргумент *дата* имеет тип DATE и вычисления затрагивают только год, месяц и день (то есть, не касаются времени), результат имеет тип DATE. В противном случае возвращается значение типа DATETIME.



Таблица 5.1. Связь между аргументами тип и выражение

Значение аргумента тип	Ожидаемый формат значения выражение
MICROSECOND	Микросекунды
SECOND	Секунды
MINUTE	Минуты
hour	Часы
DAY	Дни
WEEK	Недели
MONTH	Месяцы
QUARTER	Кварталы
YEAR	Годы
SECOND_MICROSECOND	'Секунды.Микросекунды'
MINUTE_MICROSECOND	'Минуты.Микросекунды'
MINUTE_SECOND	'Минуты:Секунды'
hour_MICROSECOND	'Часы.Микросекунды'
hour_SECOND	'Часы:Минуты:Секунды'
hour_MINUTE	'Часы:Минуты'
DAY_MICROSECOND	'Часы.Микросекунды'
DAY_SECOND	'Дни Часы:Минуты:Секунды'
DAY_MINUTE	'Дни Часы:Минуты'
DAY_HOUR	'Дни Часы'
YEAR_MONTH	'Годы-Месяцы'

Начиная с MySQL 3.23, INTERVAL выражение тип допускается с обеих сторон операции "+", если выражение на другой стороне имеет тип даты или даты-времени. Для операции "-" конструкция INTERVAL выражение тип разрешена только справа, поскольку нет смысла отнимать дату или время от интервала. Ниже представлены соответствующие примеры.

```
mysql> SELECT '1997-12-31 23:59:59' + INTERVAL 1 SECOND;
-> '1998-01-01 00:00:00'
mysql> SELECT INTERVAL 1 DAY + '1997-12-31';
-> '1998-01-01'
mysql> SELECT '1998-01-01' - INTERVAL 1 SECOND;
-> '1997-12-31 23:59:59'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59', INTERVAL 1 SECOND);
-> '1998-01-01 00:00:00'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59', INTERVAL 1 DAY);
-> '1998-01-01 23:59:59'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59', INTERVAL '1:1' MINUTE_SECOND);
-> '1998-01-01 00:01:00'
mysql> SELECT DATE_SUB('1998-01-01 00:00:00', INTERVAL '1 1:1:1' DAY_SECOND);
-> '1997-12-30 22:58:59'
mysql> SELECT DATE_ADD('1998-01-01 00:00:00', INTERVAL '-1 10' DAY_HOUR);
-> '1997-12-30 14:00:00'
```

```
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
-> INTERVAL '1.999999' SECOND MICROSECOND);
-> '1993-01-01 00:00:01.000001'
```

Если вы указываете слишком малое значение интервала (не включающее все части, заданные ключевым словом *тип*), MySQL предполагает, что в неполном значении интервала исключается левая часть. Например, если вы специфицируете тип `DAY_SECOND`, то ожидается, что выражение содержит дни, часы, минуты и секунды. Если вы укажете значение вроде `'1:10'`, MySQL предположит, что дни и часы пропущены, и значение представляет минуты и секунды. Другими словами, `'1:10' DAY_SECOND` интерпретируется как `'1:10' MINUTE_SECOND`. Это аналогично тому, как MySQL интерпретирует значения `TIME` в виде диапазона времени, а не времени дня.

Если вы добавляете или вычитаете из значения даты нечто, что содержит в себе время, результат автоматически преобразуется в значение типа даты-времени:

```
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 DAY);
-> '1999-01-02'
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 HOUR);
-> '1999-01-01 01:00:00'
```

Если вы используете неверные значения даты, результатом будет `NULL`. Если вы добавляете `MONTH`, `YEAR_MONTH` или `YEAR`, и результирующая дата имеет день, который превышает максимальное число в месяце, день исправляется на последний день месяца:

```
mysql> SELECT DATE_ADD('1998-01-30', INTERVAL 1 MONTH);
-> '1998-02-28'
```

#### ■ `DATE_FORMAT(дата, формат)`

Форматирует значение *дата* в соответствии со строкой *формат*. Спецификаторы, которые могут использоваться в строке *формат*, представлены в табл. 5.2.

Все остальные символы копируются в результат без изменений.

Спецификаторы формата `%V`, `%v`, `%x` и `%X` доступны, начиная с версии MySQL 3.23.8, `%f` — с версии MySQL 4.1.1.

Начиная с MySQL 3.23, символ `'%'` требуется перед символом спецификатора формата. В более ранних версиях он был не обязателен.

Причина того, что диапазоны месяцев и дней в спецификаторах начинаются с нуля, состоит в том, что MySQL, начиная с версии 3.23, допускает сохранение неполных дат, наподобие `'2004-00-00'`.

```
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');
-> 'Saturday October 1997'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H:%i:%s');
-> '22:23:00'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%D %y %a %d %m %b %j');
-> '4th 97 Sat 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H %k %I %r %T %S %w');
-> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
-> '1998 52'
```

Таблица 5.2. Спецификаторы формата

Спецификатор	Описание
%a	Сокращенное название дня недели (Sun..Sat – вск..суб).
%b	Сокращенное название месяца (Jan..Dec – янв..дек).
%c	Номер месяца (0..12).
%D	День месяца с суффиксом на английском языке (0th, 1st, 2nd, 3rd, ...).
%d	День месяца, число (00..31).
%e	День месяца, число (0..31).
%f	Микросекунды (000000..999999).
%H	Часы (00..23).
%h	Часы (01..12).
%I	Часы (01..12).
%i	Минуты, число (00..59).
%j	День года (001..366).
%k	Часы (0..23).
%l	Часы (0..12).
%M	Наименование месяца (January..December – январь..декабрь).
%m	Номер месяца (00..12).
%p	АМ (до полудня) или РМ (после полудня).
%r	Время в 12-часовом формате (чч:мм:сс с АМ или РМ).
%S	Секунды (00..59).
%s	Секунды (00..59).
%T	Время в 24-часовом формате (чч:мм:сс).
%U	Неделя (00..53), где воскресенье – первый день недели.
%u	Неделя (00..53), где понедельник – первый день недели.
%V	Неделя (01..53), где воскресенье – первый день недели, используется с %X.
%v	Неделя (01..53), где понедельник – первый день недели, используется с %x.
%W	Наименование дня недели (Sunday..Saturday – воскресенье..суббота).
%w	День недели (0=Sunday (воскресенье)..6=Saturday (суббота)).
%X	Год для недели, в которой воскресенье – первый день, число, 4 цифры, используется с %V.
%x	Год для недели, в которой понедельник – первый день, число, 4 цифры, используется с %v.
%Y	Год, в виде числа из 4-х цифр.
%y	Год, в виде числа из 2-х цифр.
%%	Литеральный символ '% '.

- DAY(*дата*)

DAY() – синоним DAYOFMONTH(). Добавлена в MySQL 4.1.1.

- DAYNAME(*дата*). Возвращает название дня недели для *дата*.

```
mysql> SELECT DAYNAME('1998-02-05');
-> 'Thursday'
```

- DAYOFMONTH(*дата*). Возвращает день месяца, соответствующий *дата*, в диапазоне от 1 до 31.

```
mysql> SELECT DAYOFMONTH('1998-02-03');
-> 3
```

- DAYOFWEEK(*дата*). Возвращает индекс дня недели для *дата* (1=воскресенье, 2=понедельник..., 7=суббота). Эти значения индексов соответствуют стандарту ODBC.

```
mysql> SELECT DAYOFWEEK('1998-02-03');
-> 3
```

- DAYOFYEAR(*дата*)

Возвращает день года для *дата*, в диапазоне от 1 до 366.

```
mysql> SELECT DAYOFYEAR('1998-02-03');
-> 34
```

- EXTRACT(*тип* FROM *дата*)

Функция EXTRACT() использует те же типы интервалов, что и DATE\_ADD() или DATE\_SUB(), но извлекает из дат части вместо того, чтобы выполнять арифметические действия над датами. См. информацию о значениях *тип* в описании DATE\_ADD().

```
mysql> SELECT EXTRACT(YEAR FROM '1999-07-02');
-> 1999
mysql> SELECT EXTRACT(YEAR_MONTH FROM '1999-07-02 01:02:03');
-> 199907
mysql> SELECT EXTRACT(DAY_MINUTE FROM '1999-07-02 01:02:03');
-> 20102
mysql> SELECT EXTRACT(MICROSECOND FROM '2003-01-02 10:30:00.00123');
-> 123
```

Функция EXTRACT() была добавлена в MySQL 3.23.0.

- FROM\_DAYS(*N*). Принимает номер дня *N* и возвращает дату.

```
mysql> SELECT FROM_DAYS(729669);
-> '1997-10-07'
```

FROM\_DAYS(*N*) не предназначена для использования со значениями, предшествующими принятию Григорианского календаря (1582), потому что она не принимает во внимание дни, потерянные при смене календаря.

- FROM\_UNIXTIME(*метка\_времени\_unix*)

FROM\_UNIXTIME(*метка\_времени\_unix*, *формат*)

Возвращает представление аргумента *метка\_времени\_unix* в формате 'ГГГГ-ММ-ДД ЧЧ:ММ:СС' или ГГГГММДДЧЧММСС, в зависимости от того, применяется функция в строковом или числовом контексте.

```
mysql> SELECT FROM_UNIXTIME(875996580);
-> '1997-10-04 22:23:00'
mysql> SELECT FROM_UNIXTIME(875996580) + 0;
-> 19971004222300
```

Если задан аргумент *формат*, результат форматируется в соответствии со строкой *формат*. Эта строка может содержать те же спецификаторы, что и для функции `DATE_FORMAT()`.

```
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y %D %M %h:%i:%s %x');
-> '2003 6th August 06:22:58 2003'
```

■ `GET_FORMAT(DATE|TIME|TIMESTAMP, 'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL')`

Возвращает строку формата. Эта функция применима в комбинации с функциями `DATE_FORMAT()` и `STR_TO_DATE()`.

Три возможных значения первого аргумента и пять возможных значений второго дают в результате 15 комбинаций строки формата (значения спецификаторов можно найти в описании функции `DATE_FORMAT()`).

Функция	Результат вызова
<code>GET_FORMAT(DATE, 'USA')</code>	<code>'%m.%d.%Y'</code>
<code>GET_FORMAT(DATE, 'JIS')</code>	<code>'%Y-%m-%d'</code>
<code>GET_FORMAT(DATE, 'ISO')</code>	<code>'%Y-%m-%d'</code>
<code>GET_FORMAT(DATE, 'EUR')</code>	<code>'%d.%m.%Y'</code>
<code>GET_FORMAT(DATE, 'INTERNAL')</code>	<code>'%Y%m%d'</code>
<code>GET_FORMAT(TIMESTAMP, 'USA')</code>	<code>'%Y-%m-%d-%H.%i.%s'</code>
<code>GET_FORMAT(TIMESTAMP, 'JIS')</code>	<code>'%Y-%m-%d %H:%i:%s'</code>
<code>GET_FORMAT(TIMESTAMP, 'ISO')</code>	<code>'%Y-%m-%d %H:%i:%s'</code>
<code>GET_FORMAT(TIMESTAMP, 'EUR')</code>	<code>'%Y-%m-%d-%H.%i.%s'</code>
<code>GET_FORMAT(TIMESTAMP, 'INTERNAL')</code>	<code>'%Y%m%d%H%i%s'</code>
<code>GET_FORMAT(TIME, 'USA')</code>	<code>'%h:%i:%s %p'</code>
<code>GET_FORMAT(TIME, 'JIS')</code>	<code>'%H:%i:%s'</code>
<code>GET_FORMAT(TIME, 'ISO')</code>	<code>'%H:%i:%s'</code>
<code>GET_FORMAT(TIME, 'EUR')</code>	<code>'%H.%i.%S'</code>
<code>GET_FORMAT(TIME, 'INTERNAL')</code>	<code>'%H%i%s'</code>

Форматом ISO является ISO 9075, а не ISO 8601.

```
mysql> SELECT DATE_FORMAT('2003-10-03', GET_FORMAT(DATE, 'EUR'));
-> '03.10.2003'
mysql> SELECT STR_TO_DATE('10.31.2003', GET_FORMAT(DATE, 'USA'));
-> 2003-10-31
```

`GET_FORMAT()` доступна, начиная с MySQL 4.1.1. См. раздел 6.5.3.1.

■ `HOUR(время)`. Возвращает часы для *время*. Возвращаемый диапазон — от 0 до 23 для значений времени дня.

```
mysql> SELECT HOUR('10:05:03');
-> 10
```

Однако диапазон возможных значений типа TIME в настоящее время гораздо шире, поэтому HOUR может вернуть значения больше 23.

```
mysql> SELECT HOUR('272:59:59');  
-> 272
```

- **LAST\_DAY(дата)**. Принимает значение даты или даты со временем и возвращает соответствующее значение для последнего дня месяца. Возвращает NULL, если аргумент неверный.

```
mysql> SELECT LAST_DAY('2003-02-05');  
-> '2003-02-28'  
mysql> SELECT LAST_DAY('2004-02-05');  
-> '2004-02-29'  
mysql> SELECT LAST_DAY('2004-01-01 01:01:01');  
-> '2004-01-31'  
mysql> SELECT LAST_DAY('2003-03-32');  
-> NULL
```

Функция LAST\_DAY() доступна, начиная с MySQL 4.1.1.

- **LOCALTIME, LOCALTIME()**

LOCALTIME и LOCALTIME() – это синонимы для NOW(). Они добавлены в MySQL 4.0.6.

- **LOCALTIMESTAMP, LOCALTIMESTAMP()**

LOCALTIMESTAMP и LOCALTIMESTAMP() – это синонимы для NOW(). Они добавлены в MySQL 4.0.6.

- **MAKEDATE(год, день\_года)**. Возвращает дату для заданного года и дня. Значение день\_года должно быть больше 0, в противном случае результатом будет NULL.

```
mysql> SELECT MAKEDATE(2001,31), MAKEDATE(2001,32);  
-> '2001-01-31', '2001-02-01'  
mysql> SELECT MAKEDATE(2001,365), MAKEDATE(2004,365);  
-> '2001-12-31', '2004-12-30'  
mysql> SELECT MAKEDATE(2001,0);  
-> NULL
```

MAKEDATE() доступна, начиная с MySQL 4.1.1.

- **MAKETIME(часы, минуты, секунды)**. Возвращает значение времени, вычисленное на основе аргументов часы, минуты и секунды.

```
mysql> SELECT MAKETIME(12,15,30);  
-> '12:15:30'
```

MAKETIME() доступна, начиная с MySQL 4.1.1.

- **MICROSECOND(выражение)**

Возвращает микросекунды, извлеченные из значения выражение времени или даты-времени, как число в диапазоне от 0 до 999999.

```
mysql> SELECT MICROSECOND('12:00:00.123456');  
-> 123456  
mysql> SELECT MICROSECOND('1997-12-31 23:59:59.000010');  
-> 10
```

MICROSECOND() доступна, начиная с MySQL 4.1.1.

- **MINUTE(время)**. Возвращает минуты для значения *время*, в диапазоне от 0 до 59.  

```
mysql> SELECT MINUTE('98-02-03 10:05:03');  
-> 2
```
- **MONTH(дата)**. Возвращает минуты для значения *дата*, в диапазоне от 1 до 12.  

```
mysql> SELECT MONTH('1998-02-03');  
-> 2
```
- **MONTHNAME(дата)**. Возвращает полное наименование месяца для значения *дата*.  

```
mysql> SELECT MONTHNAME('1998-02-05');  
-> 'February'
```
- **NOW()**. Возвращает текущую дату и время в формате 'ГГГГ-ММ-ДД ЧЧ:ММ:СС' или ГГГГММДДЧЧММСС, в зависимости от того, в каком контексте вызывается функция – строковом или числовом.  

```
mysql> SELECT NOW();  
-> '1997-12-15 23:50:26'  
mysql> SELECT NOW() + 0;  
-> 19971215235026
```
- **PERIOD\_ADD(P, N)**. Добавляет *N* месяцев к периоду *P* (в формате ГГММ или ГГГГММ). Возвращает значение в формате ГГГГММ. Следует отметить, что аргумент периода *P* – это не дата.  

```
mysql> SELECT PERIOD_ADD(9801,2);  
-> 199803
```
- **PERIOD\_DIFF(P1, P2)**. Возвращает количество месяцев между периодами *P1* и *P2*. *P1* и *P2* должны быть в формате ГГММ или ГГГГММ. Следует отметить, что аргументы периодов *P1* и *P2* – это не даты.  

```
mysql> SELECT PERIOD_DIFF(9802,199703);  
-> 11
```
- **QUARTER(дата)**. Возвращает квартал года для аргумента *дата*, в диапазоне от 1 до 4.  

```
mysql> SELECT QUARTER('98-04-01');  
-> 2
```
- **SECOND(время)**. Возвращает секунды для значения *время*, в диапазоне от 0 до 59.  

```
mysql> SELECT SECOND('10:05:03');  
-> 3
```
- **SEC\_TO\_TIME(секунды)**. Возвращает аргумент *секунды*, преобразованный в часы, минуты и секунды, как значение в формате 'ЧЧ:ММ:СС' или ЧЧММСС, в зависимости от того, в каком контексте вызывается функция – строковом или числовом.  

```
mysql> SELECT SEC_TO_TIME(2378);  
-> '00:39:38'  
mysql> SELECT SEC_TO_TIME(2378) + 0;  
-> 3938
```
- **STR\_TO\_DATE(строка, формат)**. Это функция, обратная **DATE\_FORMAT()**. Принимает аргумент *строка*, строку формата *формат* и возвращает значение типа DATETIME. Значения даты, времени или даты-времени, содержащиеся в аргументе *строка*, должны быть представлены в формате, указанном в аргументе *формат*. Спецификаторы

формата можно найти в описании функции `DATE_FORMAT()`. Все остальные символы принимаются буквально, поэтому не интерпретируются. Если строка содержит некорректную дату, время или дату и время, `STR_TO_DATE()` возвращает `NULL`.

```
mysql> SELECT STR_TO_DATE('03.10.2003 09.20', '%d.%m.%Y %H.%i');
-> '2003-10-03 09:20:00'
mysql> SELECT STR_TO_DATE('10arp', '%carp');
-> '0000-10-00 00:00:00'
mysql> SELECT STR_TO_DATE('2003-15-10 00:00:00', '%Y-%m-%d %H.%i.%s');
-> NULL
```

`STR_TO_DATE()` доступна, начиная с MySQL 4.1.1.

- `SUBDATE(дата, INTERVAL выражение тип)`

`SUBDATE(выражение, дни)`

Когда вызывается с формой второго аргумента `INTERVAL`, `SUBDATE()` представляет собой синоним `DATE_SUB()`. Информация об аргументе `INTERVAL` представлена в описании `DATE_ADD()`.

```
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
mysql> SELECT SUBDATE('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
```

Начиная с MySQL 4.1.1, допускается второй синтаксис, в котором *выражение* — это выражение даты или даты-времени, а *дни* — количество дней, которые нужно отнять от *выражение*.

```
mysql> SELECT SUBDATE('1998-01-02 12:00:00', 31);
-> '1997-12-02 12:00:00'
```

- `SUBTIME(выражение, выражение2)`

`SUBTIME()` отнимает *выражение2* от *выражение* и возвращает результат. *выражение* — это выражение типа даты или даты и времени, а *выражение2* — типа времени.

```
mysql> SELECT SUBTIME('1997-12-31 23:59:59.999999', '1 1:1:1.000002');
-> '1997-12-30 22:58:58.999997'
mysql> SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
-> '-00:59:59.999999'
```

`SUBTIME()` появилась в версии MySQL 4.1.1.

- `SYSDATE()`

`SYSDATE()` — это синоним `NOW()`.

- `TIME(выражение)`. Извлекает часть, относящуюся ко времени, из выражения *выражение* типа времени или даты-времени.

```
mysql> SELECT TIME('2003-12-31 01:02:03');
-> '01:02:03'
mysql> SELECT TIME('2003-12-31 01:02:03.000123');
-> '01:02:03.000123'
```

`TIME()` добавлена в MySQL 4.1.1.

- `TIMEDIFF(выражение, выражение2)`. Возвращает время между начальным моментом *выражение* и конечным моментом *выражение2*. *выражение* и *выражение2* — выражения типа времени или даты-времени, причем оба они должны иметь один и тот же тип.



```
mysql> SELECT TIMEDIFF('2000:01:01 00:00:00', '2000:01:01 00:00:00.000001');
-> '-00:00:00.000001'
mysql> SELECT TIMEDIFF('1997-12-31 23:59:59.000001',
-> '1997-12-30 01:01:01.000002');
-> '46:58:57.999999'
```

TIMEDIFF() появилась в MySQL 4.1.1.

#### ■ TIMESTAMP(выражение)

TIMESTAMP(выражение, выражение2)

В варианте с одним аргументом функция возвращает выражение *выражение* в виде даты со временем. В варианте с двумя аргументами функция добавляет выражение типа времени *выражение2* к дате или дате со временем *выражение*, и возвращает значение типа даты со временем.

```
mysql> SELECT TIMESTAMP('2003-12-31');
-> '2003-12-31 00:00:00'
mysql> SELECT TIMESTAMP('2003-12-31 12:00:00', '12:00:00');
-> '2004-01-01 00:00:00'
```

TIMESTAMP() добавлена в MySQL 4.1.1.

#### ■ TIMESTAMPADD(интервал, целочисленное\_выражение, выражение\_datetime)

Добавляет целое выражение *целочисленное\_выражение* к выражению типа даты или даты со временем *выражение\_datetime*. Единица измерения *целочисленное\_выражение* задается аргументом *интервал*, которое может быть одним из следующих: *FRAC\_SECOND*, *SECOND*, *MINUTE*, *HOURL*, *DAY*, *WEEK*, *MONTH*, *QUARTER* или *YEAR*.

Значение *интервал* может указываться с использованием одного из приведенных выше ключевых слов или с префиксом *SQL\_TST\_*. Например, правильно и *DAY*, и *SQL\_TST\_DAY*.

```
mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
-> '2003-01-02 00:01:00'
mysql> SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
-> '2003-01-09'
```

TIMESTAMPADD() появилась в MySQL 5.0.0.

#### ■ TIMESTAMPDIFF(интервал, выражение\_datetime1, выражение\_datetime2)

Возвращает целочисленную разницу между выражениями типа даты или даты со временем *выражение\_datetime1* и *выражение\_datetime2*. Единица измерения задается параметром *интервал*. Допустимыми значениями *интервал* являются те же, что и перечисленные в описании функции *TIMESTAMPADD()*.

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
-> 3
mysql> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
-> -1
```

TIMESTAMPDIFF() добавлена в MySQL 5.0.0.

#### ■ TIME\_FORMAT(время, формат). Эта функция используется подобно DATE\_FORMAT(), но строка *формат* может содержать только такие спецификаторы формата, которые управляют представлением часов, минут и секунд. Все остальные спецификаторы генерируют NULL или 0.

Если значение *время* содержит часы, которые больше 23, спецификаторы формата %H и %h генерируют значение, не входящее в обычный диапазон от 0 до 23. Другие спецификаторы часового формата генерируют значение по модулю 12.

```
mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');  
-> '100 100 04 04 4'
```

- **TIME\_TO\_SEC(*время*)**. Возвращает значение аргумента *время*, преобразованное в секунды.

```
mysql> SELECT TIME_TO_SEC('22:23:00');  
-> 80580  
mysql> SELECT TIME_TO_SEC('00:39:38');  
-> 2378
```

- **TO\_DAYS(*дата*)**. Для заданной в аргументе *дата* значения даты возвращает номер дня (количество дней, прошедших с начала года 0).

```
mysql> SELECT TO_DAYS(950501);  
-> 728779  
mysql> SELECT TO_DAYS('1997-10-07');  
-> 729669
```

Функция **TO\_DAYS()** не предназначена для использования со значениями, которые предшествуют дате принятия Григорианского календаря (1582), потому что не принимает во внимание дни, потерянные при смене календаря.

Помните, что MySQL преобразует годы, представленные двумя цифрами, в форму с четырьмя цифрами по правилам, описанным в разделе 4.3. Например, '1997-10-07' и '97-10-07' рассматриваются как одинаковые даты:

```
mysql> SELECT TO_DAYS('1997-10-07'), TO_DAYS('97-10-07');  
-> 729669, 729669
```

Для дат ранее 1582 года результат функции не определен.

- **UNIX\_TIMESTAMP()**

**UNIX\_TIMESTAMP(*дата*)**

Если вызывается без аргументов, возвращает метку времени Unix (секунды, прошедшие с момента '1970-01-01 00:00:00' GMT) в виде беззнакового целого. Если **UNIX\_TIMESTAMP()** вызывается с аргументом *дата*, она возвращает значение аргумента в виде секунд, прошедших с момента '1970-01-01 00:00:00' GMT. Аргумент *дата* может быть строкой DATE, строкой DATETIME, TIMESTAMP или числом в формате ГГММДД или ГГГГММДД в локальном времени.

```
mysql> SELECT UNIX_TIMESTAMP();  
-> 882226357  
mysql> SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00');  
-> 875996580
```

Когда **UNIX\_TIMESTAMP** применяется со столбцом типа **TIMESTAMP**, функция возвращает значение внутренней метки времени непосредственно, без неявного преобразования из строки в метку времени Unix. Если вы передаете неверную дату **UNIX\_TIMESTAMP**, она вернет 0, однако помните, что выполняется только базовая проверка диапазона (год – от 1970 до 2037, месяц – от 01 до 12, день – от 01 до 31). Если вам нужно извлекать столбцы с помощью **UNIX\_TIMESTAMP()**, возможно, потребуется приводить результат к целому со знаком. См. раздел 5.7.

- **UTC\_DATE, UTC\_DATE()**. Возвращает текущую дату UTC (Universal Coordinated Time – Универсальное скоординированное время) как значение в формате 'ГГГГ-ММ-ДД' или ГГГГММДД, в зависимости от контекста, в котором функция вызывается.

```
mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
-> '2003-08-14', 20030814
```

UTC\_DATE() появилась в MySQL 4.1.1

- **UTC\_TIME, UTC\_TIME()**. Возвращает текущее время UTC (Universal Coordinated Time – Универсальное скоординированное время) как значение в формате 'ЧЧ:ММ:СС' или ЧЧММСС, в зависимости от контекста, в котором функция вызывается.

```
mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
-> '18:07:53', 180753
```

UTC\_TIME() добавлена в MySQL 4.1.1.

- **UTC\_TIMESTAMP, UTC\_TIMESTAMP()**. Возвращает текущую дату и время UTC (Universal Coordinated Time – Универсальное скоординированное время) как значение в формате 'ГГГГ-ММ-ДД ЧЧ:ММ:СС' или ГГГГММДДЧЧММСС, в зависимости от контекста, в котором функция вызывается.

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
-> '2003-08-14 18:08:04', 20030814180804
```

UTC\_TIMESTAMP() появилась в MySQL 4.1.1.

- **WEEK(дата[, режим])**. Функция возвращает номер недели для даты дата. При вызове с двумя аргументами, WEEK() позволяет указать, считать началом недели воскресенье или понедельник, и в каком диапазоне должно быть возвращаемое значение: 0–53 или 1–52. Если аргумент режим пропущен, используется значение системной переменной default\_week\_format (или 0 – до версии MySQL 4.0.14). В табл. 5.3 представлены допустимые значения аргумента режим вместе с описаниями их эффекта.

Таблица 5.3. Допустимые значения аргумента режим

Значение	Описание
0	Неделя начинается в воскресенье; возвращаемое значение лежит в диапазоне от 0 до 53; неделя 1 – первая неделя в году.
1	Неделя начинается в понедельник; возвращаемое значение лежит в диапазоне от 0 до 53; неделя 1 – первая неделя в году.
2	Неделя начинается в воскресенье; возвращаемое значение лежит в диапазоне от 1 до 53; неделя 1 – первая неделя в году.
3	Неделя начинается в понедельник; возвращаемое значение лежит в диапазоне от 1 до 53; неделя 1 – первая неделя в году.
4	Неделя начинается в воскресенье; возвращаемое значение лежит в диапазоне от 0 до 53; неделя 1 – первая неделя, на которую в данном году приходится более 3 дней.
5	Неделя начинается в воскресенье; возвращаемое значение лежит в диапазоне от 0 до 53; неделя 1 – первая неделя, которая начинается в данном году.
6	Неделя начинается в воскресенье; возвращаемое значение лежит в диапазоне от 1 до 53; неделя 1 – первая неделя, на которую в данном году приходится более 3 дней.
7	Неделя начинается в воскресенье; возвращаемое значение лежит в диапазоне от 1 до 53; неделя 1 – первая неделя, которая начинается в данном году.

Значение *режим*, равное 3, может использоваться, начиная с MySQL 4.0.5. Значения от 4 и выше могут использоваться, начиная с MySQL 4.0.17.

```
mysql> SELECT WEEK('1998-02-20');
-> 7
mysql> SELECT WEEK('1998-02-20',0);
-> 7
mysql> SELECT WEEK('1998-02-20',1);
-> 8
mysql> SELECT WEEK('1998-12-31',1);
-> 53
```

#### На заметку!

В MySQL 4.0 функция WEEK(*дата*, 0) была изменена для соответствия календарю, принятому в США. До этого функция WEEK() вычислялась неправильно для дат США. (В результате и WEEK(*дата*), и WEEK(*дата*, 0) давали некорректные результаты во всех случаях.)

Следует отметить, что если дата приходится на последнюю неделю предыдущего года, MySQL возвращает 0, если не указано значение 2, 3, 6 или 7 для аргумента *режим*.

```
mysql> SELECT WEEK('2000-01-01',2);
-> 52
```

В качестве альтернативы можно пользоваться функцией YEARWEEK():

```
mysql> SELECT YEARWEEK('2000-01-01');
-> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2);
-> '52'
```

- WEEKOFDAY(*дата*). Возвращает индекс дня недели для даты *дата* (0=понедельник, 1=вторник, ... 6=воскресенье).

```
mysql> SELECT WEEKDAY('1998-02-03 22:23:00');
-> 1
mysql> SELECT WEEKDAY('1997-11-05');
-> 2
```

- WEEKOFYEAR(*дата*). Возвращает календарную неделю года для заданной даты *дата* в диапазоне от 1 до 53.

```
mysql> SELECT WEEKOFYEAR('1998-02-20');
-> 8
```

WEEKOFYEAR() доступна, начиная с MySQL 4.1.1.

- YEAR(*дата*). Возвращает год заданной даты, в диапазоне от 1000 до 9999.

```
mysql> SELECT YEAR('98-02-03');
-> 1998
```

- YEARWEEK(*дата*)

YEARWEEK(*дата*, *начало*)

Возвращает год и неделю для заданной даты. Аргумент *начало* имеет тот же смысл, что и в функции WEEK(). Год в результате может отличаться от года в аргументе *дата* для первой и последней недели в году.

```
mysql> SELECT YEARWEEK('1987-01-01');
-> 198653
```

Следует отметить, что номер недели отличается от того, который возвращает функция `WEEK()` (то есть 0) при значениях необязательного аргумента *режим*, равных 0 или 1, поскольку `WEEK()` возвращает неделю в контексте заданного года.

`YEARWEEK()` была добавлена в MySQL 3.23.8.

## 5.6. Функции полнотекстового поиска

- `MATCH (столбец1, столбец2, ...) AGAINST (выражение [IN BOOLEAN MODE | WITH QUERY EXPANSION])`

Начиная с версии 3.23.23, MySQL поддерживает полнотекстовую индексацию и поиск. Полнотекстовый индекс в MySQL – это индекс типа `FULLTEXT`. Такие индексы применяются только в таблицах `MyISAM`, и создаваться могут только на столбцах типа `CHAR`, `VARCHAR` или `TEXT` во время выполнения оператора `CREATE TABLE`, либо добавляться позже с помощью операторов `ALTER TABLE` или `CREATE INDEX`. Для больших наборов данных гораздо быстрее будет сначала загрузить данные в таблицу, которая не имеет индекса `FULLTEXT`, а затем создать его с помощью `ALTER TABLE` или `CREATE INDEX`. Загрузка данных в таблицу, у которой уже есть такой индекс, выполняется ощутимо медленнее.

Ограничения на полнотекстовый поиск перечислены в разделе 5.6.3.

Полнотекстовый поиск выполняется с применением функции `MATCH()`.

```
mysql> CREATE TABLE articles (
  -> id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
  -> title VARCHAR(200),
  -> body TEXT,
  -> FULLTEXT (title,body)
  -> );

Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO articles (title,body) VALUES
  -> ('MySQL Tutorial','DBMS stands for DataBase ...'),
  -> ('How To Use MySQL Well','After you went through a ...'),
  -> ('Optimizing MySQL','In this tutorial we will show ...'),
  -> ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
  -> ('MySQL vs. YourSQL','In the following database comparison ...'),
  -> ('MySQL Security','When configured properly, MySQL ...');
```

Query OK, 6 rows affected (0.00 sec)

Records: 6 Duplicates: 0 Warnings: 0

```
mysql> SELECT * FROM articles
  -> WHERE MATCH (title,body) AGAINST ('database');
```

```
+-----+-----+-----+
| id | title                | body                                |
+-----+-----+-----+
| 5 | MySQL vs. YourSQL    | In the following database comparison ... |
| 1 | MySQL Tutorial       | DBMS stands for DataBase ...           |
+-----+-----+-----+

2 rows in set (0.00 sec)
```

Функция `MATCH()` осуществляет естественный языковый поиск строки в текстовой коллекции. Коллекция – это набор из одного или более столбцов, включенных в индекс

**FULLTEXT.** Искомая строка задается аргументом `AGAINST()`. Поиск осуществляется в режиме, нечувствительном к регистру. Для каждой строки в таблице `MATCH()` возвращает релевантное значение, то есть измеренную степень сходства между искомой строкой и текстом в строке таблицы, состоящим из столбцов, перечисленных в списке `MATCH()`.

Когда `MATCH()` применяется в конструкции `WHERE`, как в предыдущем примере, возвращаемые строки автоматически сортируются в порядке убывания релевантного значения (степени сходства). Релевантные значения – это неотрицательные числа с плавающей точкой. Нулевая релевантность означает отсутствие сходства. Оно вычисляется на базе количества слов в строке, количества уникальных слов в строке, общего числа слов в коллекции и количества документов (строк таблицы), содержащих конкретное слово.

Для естественного языкового полнотекстового поиска существует требование, что столбцы, перечисленные в аргументах функции `MATCH()`, были теми же, по которым построен индекс `FULLTEXT` вашей таблицы. Заметьте, что в предшествующем запросе столбцы, перечисленные в аргументах функции `MATCH()` (`title` и `body`), – те же самые, что и в определении `FULLTEXT`-индекса таблицы `articles`. Если вы хотите искать раздельно по столбцам `title` и `body`, то должны создать отдельный `FULLTEXT`-индекс для каждого столбца.

Существует также возможность булевского поиска или поиска с расширением запроса. Эти типы поиска описаны в разделах 5.6.1 и 5.6.2.

Предыдущий пример – это базовая иллюстрация, показывающая как использовать функцию `MATCH()`, в которой строки возвращаются в порядке убывания релевантности. Следующий пример демонстрирует, как явно извлекать значения релевантности. Возвращаемые строки не упорядочены, потому что оператор `SELECT` не содержит ни конструкции `ORDER BY`, ни конструкции `WHERE`:

```
mysql> SELECT id, MATCH (title,body) AGAINST ('Tutorial')
-> FROM articles;
```

id	MATCH (title,body) AGAINST ('Tutorial')
1	0.65545833110809
2	0
3	0.66266459226608
4	0
5	0
6	0

```
6 rows in set (0.00 sec)
```

Представленный ниже пример более сложный. Запрос возвращает значения релевантности, а также строки в порядке его убывания. Чтобы достичь этого результата, вы должны указать `MATCH()` дважды: один раз в списке столбцов `SELECT` и второй – в конструкции `WHERE`. Это не приводит к дополнительной нагрузке, поскольку оптимизатор MySQL определяет, что два вызова `MATCH()` идентичны и выполняет код полнотекстового поиска только один раз.

```
mysql> SELECT id, body, MATCH (title,body) AGAINST
-> ('Security implications of running MySQL as root') AS score
-> FROM articles WHERE MATCH (title,body) AGAINST
-> ('Security implications of running MySQL as root');
```

```

+-----+-----+-----+
| id | body | score |
+-----+-----+-----+
| 4 | 1. Never run mysqld as root. 2. ... | 1.5219271183014 |
| 6 | When configured properly, MySQL ... | 1.3114095926285 |
+-----+-----+-----+
2 rows in set (0.00 sec)

```

MySQL применяет очень простой анализатор для разделения текста на слова. “Слово” – это любая последовательность букв, цифр, “\_” или “-”. Некоторые слова при полнотекстовом поиске игнорируются:

- Любые слишком короткие слова игнорируются. Минимальная длина по умолчанию слова, которое может быть найдено при полнотекстовом поиске, – четыре символа.
- Слова из списка “стоп-слов” игнорируются. Стоп-слова – это слова вроде “the”, “some”, которые настолько часто встречаются, что рассматриваются как имеющие нулевое семантическое значение. Существует встроенный список стоп-слов.

Минимальная длина слова по умолчанию и список стоп-слов могут быть изменены, как описано в разделе 5.6.4.

Каждое правильное слово в коллекции и в запросе “взвешивается” в соответствии с его важностью в коллекции или запросе. Таким образом, слово, представленное во многих документах, имеет меньший вес (может даже иметь нулевой вес), поскольку имеет меньшее семантическое значение в данной коллекции. И наоборот, если слово редкое, то оно получает больший вес. Показатели веса слов затем комбинируются для вычисления релевантности строки.

Такая техника работает наилучшим образом с большими коллекциями (фактически, она специально настроена на это). Для очень маленьких таблиц статистическое распределение слов не отражает адекватно их семантические значения, что иногда может приводить к причудливым результатам. Например, несмотря на то, что слово “MySQL” представлено в каждой строке таблицы `articles`, поиск по этому слову не дает результата:

```

mysql> SELECT * FROM articles
-> WHERE MATCH (title,body) AGAINST ('MySQL');
Empty set (0.00 sec)

```

Результат поиска пуст, потому что слово “MySQL” содержится, по меньшей мере, в 50% всех строк. В связи с этим, оно рассматривается как стоп-слово. Для больших наборов данных это наиболее желательное поведение – запрос на естественном языке не должен ежесекундно возвращать строку из таблицы размером, скажем, в 1 Гбайт. Для малых наборов, это, конечно же, менее желательно.

Слово, которому соответствует половина строк таблицы, менее полезно для поиска релевантных документов. Фактически, вероятно, что оно найдет большое число нерелевантных документов. Мы все знаем, что такое случается слишком часто, когда пытаемся найти что-то в Internet с помощью поисковых систем. Это происходит по той причине, что строки содержат слово, имеющее низкое семантическое значение *в конкретном наборе данных, в котором оно встречается*. Определенное слово может превысить порог “попадания” в 50% в одном наборе данных, а в другом – нет.

Этот 50%-ный порог, скорее всего, будет достигнут, когда вы впервые пытаетесь выполнить полнотекстовый поиск, чтобы посмотреть, как он работает. Если вы создаете таблицу и вставляете в нее только одну или две строки текста, то получается, что каждое

слово текста встречается как минимум в 50% строк. В результате поиск не вернет никакого результата. Для уверенности введите минимум три строки, а лучше – намного больше.

### 5.6.1. Булевский полнотекстовый поиск

Начиная с версии 4.0.1, MySQL может также выполнять булевский полнотекстовый поиск, используя модификатор `IN BOOLEAN MODE`.

```
mysql> SELECT * FROM articles WHERE MATCH (title,body)
-> AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);
```

id	title	body
1	MySQL Tutorial	DBMS stands for DataBase ...
2	How To Use MySQL Well	After you went through a ...
3	Optimizing MySQL	In this tutorial we will show ...
4	1001 MySQL Tricks	1. Never run mysqld as root. 2. ...
6	MySQL Security	When configured properly, MySQL ...

Представленный выше запрос извлекает все строки, содержащие слово “MySQL”, но не содержащие слова “YourSQL”.

Булевский полнотекстовый поиск имеет следующие характеристики:

- Он не использует 50%-ный порог.
- Он не сортирует автоматически строки в порядке убывания релевантности. Вы можете это видеть в предыдущем примере: строка с наибольшей релевантностью – та, что содержит “MySQL” дважды, – в списке выведена последней, а не первой.
- Он может работать даже без FULLTEXT-индекса, несмотря на то, что это будет медленнее.

Булевский полнотекстовый поиск поддерживает следующие операции:

- **+**. Ведущий знак “плюс” означает, что слово *должно* присутствовать во всех возвращаемых строках.
- **-**. Ведущий знак минус означает, что слово *не должно* присутствовать ни в одной из возвращаемых строк.
- (операция не указана). По умолчанию (когда не указан ни “плюс”, ни “минус”), слово является необязательным, однако строки, содержащие его, будут оцениваться выше. Это подобно поведению `MATCH() ... AGAINST()` без модификатора `IN BOOLEAN MODE`.
- **> <**. Эти две операции применяются для изменения влияния слова на значение релевантности, присваиваемого строке. Операция **>** увеличивает влияние слова, а операция **<** – уменьшает. См. примеры ниже.
- **()**. Скобки используются для того, чтобы группировать слова в подвыражения. Подгруппы в скобках могут быть вложенными.
- **~**. Ведущий символ “тильда” действует как операция отрицания, делая влияние слова на релевантность строки отрицательным. Это применимо для отметки излишних слов. Строка, включающая такие слова, будет оцениваться ниже



лишних слов. Строка, включающая такие слова, будет оцениваться ниже других, но не будет исключаться вообще, как это делается с помощью операции `-`.

- `*`. “Звездочка” – это операция усечения. В отличие от других операций, она должна быть *добавлена* к слову.
- `"`. Фраза, заключенная в двойные кавычки, соответствует только тем строкам, которые содержат ее *литерально – в точности так, как она приведена*.

Следующие примеры демонстрируют некоторые поисковые строки, которые используют булевские операции полнотекстового поиска:

- `'apple banana'`. Искать строки, которые содержат, как минимум, одно из двух слов.
- `'+apple +juice'`. Искать строки, которые содержат оба слова.
- `'+apple macintosh'`. Искать строки, содержащие слово “apple”, но ранг строки повышается, если в ней есть также слово “macintosh”.
- `'apple -macintosh'`. Искать строки, в которых есть слово “apple”, но нет “macintosh”.
- `'apple +( >turnover <strudel )'`. Искать строки, содержащие слова “apple” и “turnover”, либо “apple” и “strudel” (в любом порядке), но ранг “apple turnover” выше, чем “apple strudel”.
- `'apple*'`. Искать строки, содержащие такие слова, как “apple”, “apples”, “applesauce” или “apple”.
- `'"some words"'`. Искать строки, содержащие буквальную фразу “some words” (например, строки, которые содержат “some words of wisdom”, но не “some noise words”). Следует отметить, что символы `'`, которые обрамляют фразу, являются ограничителями фразы, а не кавычками, которые окружают собственно поисковую строку.

### 5.6.2. Полнотекстовый поиск с расширением запроса

Начиная с MySQL 4.1.1, полнотекстовый поиск поддерживает расширение запроса (в частности, вариант “слепого расширения запроса” (blind query expansion)). Это применимо, как правило, когда поисковая фраза слишком короткая, что часто означает, что пользователь полагается на некие подразумеваемые знания, которых механизм полнотекстового поиска обычно не имеет. Например, пользователь, выполняя поиск по слову “database”, может иметь в виду, что фразы с “MySQL”, “Oracle”, “DB2” и “RDBMS” должны соответствовать поисковому слову и быть также возвращены. Это подразумеваемое знание.

Слепое расширение запроса (также известное, как автоматическая релевантная обратная связь (automatic relevance feedback)) осуществляется путем добавления `WITH QUERY EXPANSION` следом за искомой фразой. Это работает за счет выполнения двойного поиска – когда к искомой фразе при втором проходе добавляются несколько документов из вершины списка, сгенерированного на первом проходе. Таким образом, если один из этих документов содержит слово “database” и слово “MySQL”, второй проход поиска обнаружит документы, содержащие “MySQL”, даже если в них не будет “database”. Следующий пример демонстрирует эту разницу:

```
mysql> SELECT * FROM articles
-> WHERE MATCH (title,body) AGAINST ('database');
```

```

+-----+-----+-----+
| id | title                | body                |
+-----+-----+-----+
| 5 | MySQL vs. YourSQL    | In the following database comparison ... |
| 1 | MySQL Tutorial       | DBMS stands for DataBase ... |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM articles
      -> WHERE MATCH (title,body)
      -> AGAINST ('database' WITH QUERY EXPANSION);
+-----+-----+-----+
| id | title                | body                |
+-----+-----+-----+
| 1 | MySQL Tutorial       | DBMS stands for DataBase ... |
| 5 | MySQL vs. YourSQL    | In the following database comparison ... |
| 3 | Optimizing MySQL     | In this tutorial we will show ... |
+-----+-----+-----+
3 rows in set (0.00 sec)

```

Технику, представленную во втором примере, можно использовать для поиска книг, скажем, Жоржа Сименона о комиссаре Мегрэ, когда пользователь не знает точно, как пишется “Мегрэ”. Поиск по “Megre and the reluctant witnesses” найдет только “Maigret and the Reluctant Witnesses” без расширений запроса. Поиск с расширением запроса найдет все книги со словом “Maigret” на втором проходе.

#### На заметку!

Поскольку слепое расширение запроса имеет тенденцию ощутимо увеличивать количество “шума”, возвращая нерелевантные документы, это имеет смысл применять, только когда поисковая фраза достаточно короткая.

### 5.6.3. Ограничения полнотекстового поиска

- Полнотекстовый поиск поддерживается только для таблиц MyISAM.
- Начиная с MySQL 4.1.1, полнотекстовый поиск может использоваться с большинством многобайтных символьных наборов. Исключением является Unicode. Набор символов utf8 может использоваться, тогда как ucs2 — нет.
- Начиная с MySQL 4.1, поддерживается применение нескольких наборов символов в пределах одной таблицы. Однако все столбцы FULLTEXT-индекса должны использовать один набор символов и порядок сопоставления.
- Список столбцов MATCH() должен точно соответствовать списку столбцов FULLTEXT-индекса таблицы, если только не указано MATCH() IN BOOLEAN MODE.
- Аргументом AGAINST() должна быть константная строка.

### 5.6.4. Тонкая настройка полнотекстового поиска MySQL

Средство полнотекстового поиска MySQL пока имеет только несколько настраиваемых пользователем параметров, несмотря на то, что задача добавления новых обладает высоким приоритетом в планах того, что должно быть нами сделано. Вы можете получить больший контроль над поведением полнотекстового поиска, если у вас есть исходный дистрибутив MySQL, потому что некоторые изменения требуют модификации исходного кода.

Отметим, что полнотекстовый поиск был тщательно настроен для обеспечения наилучшей эффективности. Модификация поведения по умолчанию может в большинстве случаев только ухудшить результат. Не изменяйте исходных текстов MySQL, если только вы точно не знаете, что делаете!

Большинство переменных, имеющих отношение к полнотекстовому поиску, и описанных ниже, могут быть установлены во время запуска сервера. Для того чтобы изменить эти переменные, требуется перезапуск сервера; динамическая модификация во время работы сервера не предусмотрена.

Некоторые изменения переменных требуют перестройки FULLTEXT-индексов ваших таблиц. Инструкции приведены в конце настоящего раздела.

- Минимальная и максимальная длина слов, подлежащих индексации, определяется системными переменными `ft_min_word_len` и `ft_max_word_len` (доступными, начиная с версии MySQL 4.0.0). Минимальное значение по умолчанию – четыре символа. Максимум по умолчанию зависит от вашей версии MySQL. Если вы изменяете любое из этих значений, то должны перестроить свои FULLTEXT-индексы. Например, если вы хотите, чтобы можно было искать трехсимвольные слова, то можете изменить значение переменной `ft_min_word_len`, поместив следующие строки в файл опций:

```
[mysqld]  
ft_min_word_len=3
```

Затем перезапустите сервер и перестройте существующие FULLTEXT-индексы. Обратите особое внимание на примечания относительно `mysamchk` в инструкциях, следующих за настоящим списком.

- Чтобы переопределить список стоп-слов по умолчанию, установите системную переменную `ft_stopword_file` (применяется, начиная с MySQL 4.0.10). Значением переменной должен быть полный путь к файлу, содержащему список стоп-слов, либо пустая строка, если надо отключить фильтрацию по стоп-словам. После изменения этой переменной перестройте существующие FULLTEXT-индексы.
- 50%-ный порог для естественного языкового поиска определяется выбором определенной схемы “весовых соотношений”. Чтобы отключить ее, найдите в файле `myisam/ftdefs.h` следующую строку:

```
#define GWS_IN_USE GWS_PROB
```

Измените ее следующим образом:

```
#define GWS_IN_USE GWS_FREQ
```

Затем перекомпилируйте MySQL. В этом случае перестраивать индексы не требуется.

#### На заметку!

Сделав это, вы значительно снизите способность MySQL присваивать строкам адекватные релевантные значения функцией `MATCH()`. Если вам действительно нужно выполнять поиск по таким часто употребляемым словам, будет лучше вместо этого применить поиск с `IN BOOLEAN MODE`, который не обращает внимания на 50%-ный порог.

- Чтобы изменить операции, применяемые для булевого полнотекстового поиска, установите системную переменную `ft_boolean_syntax` (доступна, начиная с MySQL 4.0.1). Эта переменная также может быть изменена на работающем сервере, но вы должны иметь привилегию `SUPER` для того, чтобы сделать это. Перестраивать индексы не нужно.

Если вы модифицируете полнотекстовые переменные, которые затрагивают индексацию (`ft_min_word_len`, `ft_max_word_len` или `ft_stopword_file`), то после внесения изменений должны перестроить все свои FULLTEXT-индексы и перезапустить сервер. Чтобы перестроить индексы в этом случае, достаточно выполнить операцию быстрого восстановления таблицы:

```
mysql> REPAIR TABLE имя_таблицы QUICK;
```

Специально в связи с использованием средства IN BOOLEAN MODE, если вы обновляете сервер MySQL 3.23 до версии 4.0 или выше, также необходимо заменить заголовок индекса. Чтобы сделать это, выполните следующее:

```
mysql> REPAIR TABLE имя_таблицы USE_FRM;
```

Это необходимо потому, что булевский полнотекстовый поиск требует наличия флага в заголовке индекса, которого не было в MySQL 3.23, и который не добавляется, если вы осуществляете только восстановление QUICK. При попытке выполнить булевский полнотекстовый поиск без такой перестройки индексов, он вернет некорректный результат.

Отметим, что если вы используете `myisamchk` для операции, которая модифицирует индексы (такой как анализ или восстановление), FULLTEXT-индексы перестраиваются с использованием значений по умолчанию для параметров минимальной и максимальной длины слова, а также файла стоп-слов на сервере, если вы не укажете другого. Это может приводить к аварийному завершению запросов.

Проблема возникает из-за того, что эти параметры известны только серверу. Они не сохраняются в индексных файлах `myISAM`. Чтобы избежать проблем, когда вы модифицируете длину минимального и максимального слова или файл стоп-слов на сервере, нужно указывать те же значения `ft_min_word_len`, `ft_max_word_len` и `ft_stopword_file` программе `myisamchk`, что используются в `mysqld`. Например, если минимальная длина слова установлена в 3 символа, вы можете восстановить таблицу с помощью `myisamchk` следующим образом:

```
shell> myisamchk --recover --ft_min_word_len=3 имя_таблицы.MYI
```

Чтобы гарантировать, что `myisamchk` и сервер используют те же значения параметров полнотекстового поиска, можно поместить каждый из них в оба раздела `[mysqld]` и `[myisamchk]` файла опций.

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

Альтернативой применению `myisamchk` являются операторы `REPAIR TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE` или `ALTER TABLE`. Эти операторы выполняются сервером, которому известны правильные значения параметров полнотекстового поиска.

### 5.6.5. Что планируется сделать для полнотекстового поиска

- Увеличить производительность операций FULLTEXT.
- Реализовать операции сходства.

- Ввести поддержку “всегда индексированных слов”. Это будут любые строки, которые пользователь желает интерпретировать как слова, подобные “C++”, “AS/400” или “TCP/IP”.
- Реализовать поддержку полнотекстового поиска в таблицах MERGE.
- Реализовать поддержку набора символов ucs2.
- Сделать список стоп-слов зависимым от языка набора символов.
- Реализовать морфологический поиск (зависимый от языка набора символов).
- Создать общий пользовательский предварительный анализатор UDF (определенных пользователем функций).
- Сделать модель более гибкой (добавив некоторые настраиваемые параметры FULLTEXT в операторы CREATE TABLE и ALTER TABLE).

## 5.7. Функции приведения

- CAST(выражение AS тип)
- CONVERT(выражение AS тип)
- CONVERT(выражение USING имя\_преобразователя)

Функции CAST() и CONVERT() могут использоваться для получения значения другого типа из значения заданного типа.

Аргумент тип может принимать одно из следующих значений:

- BINARY
- CHAR
- DATE
- DATETIME
- SIGNED [INTEGER]
- TIME
- UNSIGNED [INTEGER]

CAST() и CONVERT() доступны, начиная с MySQL 4.0.2. Преобразование типа CHAR доступно, начиная с версии 4.0.6. Форма с USING функции CONVERT() доступна, начиная с версии 4.1.0.

CAST() и CONVERT(... USING ...) имеют стандартный синтаксис SQL. Форма CONVERT() без USING – это синтаксис ODBC.

CONVERT() с USING применяется для преобразования данных между различными наборами символов. В MySQL имена преобразователей совпадают с именами соответствующих наборов символов. Например, приведенный ниже оператор преобразует строку 'abc', представленную в серверном наборе символов по умолчанию, в набор символов utf8:

```
SELECT CONVERT('abc' USING utf8);
```

Функции приведения удобны, если необходимо создать столбец специфического типа в операторе CREATE...SELECT:

```
CREATE TABLE new_table SELECT CAST('2000-01-01' AS DATE);
```

Эти функции также можно использовать для сортировки столбцов типа ENUM в лексикографическом порядке. Обычно сортировка таких столбцов происходит в соответствии с их внутренними числовыми значениями. Приведение к типу CHAR позволяет выполнить лексическую сортировку:

```
SELECT столбец_enum FROM имя_таблицы ORDER BY CAST(столбец_enum AS CHAR);
```

CAST(строка AS BINARY) — это то же самое, что BINARY строка. CAST(выражение AS CHAR) трактует выражение как строку в наборе символов по умолчанию.

### На заметку!

В MySQL 4.0 применение функции CAST() к столбцу DATE, DATETIME или TIME только помечает столбец как относящийся к определенному типу, но не меняет его значения.

Начиная с MySQL 4.1.0, значение конвертируется в правильный тип столбца, когда отправляется пользователю (это свойство нового протокола 4.1, который отправляет информацию о дате клиенту):

```
mysql> SELECT CAST(NOW() AS DATE);
-> 2003-05-26
```

Начиная с MySQL 4.1.1, CAST() также изменяет результат, если вы используете ее как часть более сложного выражения, такого как CONCAT('Date: ', CAST(NOW() AS DATE)).

Не следует использовать CAST() для извлечения данных в различных форматах, а лучше применять для этого функции LEFT() или EXTRACT(). См. раздел 5.5.

Чтобы привести строку к числовому значению, как правило, не нужно делать ничего. Просто используйте строковое значение так, как будто бы оно числовое:

```
mysql> SELECT 1+'1';
-> 2
```

Если использовать число в строковом контексте, оно автоматически преобразуется в бинарную строку:

```
mysql> SELECT CONCAT('hello you ',2);
-> 'hello you 2'
```

MySQL поддерживает арифметику 64-разрядных значений со знаком и без. Если вы используете числовые операции (такие как +), и один из операндов является беззнаковым целым, результат будет беззнаковым. Это можно изменить, если применять операции приведения SIGNED или UNSIGNED, чтобы соответственно привести операцию к знаковому или беззнаковому 64-разрядному целому.

```
mysql> SELECT CAST(1-2 AS UNSIGNED)
-> 18446744073709551615
mysql> SELECT CAST(CAST(1-2 AS UNSIGNED) AS SIGNED);
-> -1
```

Следует отметить, что если оба операнда являются значениями с плавающей точкой, результатом будет значение с плавающей точкой, и это не затрагивает предыдущего правила. (В этом контексте значения столбцов DECIMAL трактуются как значения с плавающей точкой.)

```
mysql> SELECT CAST(1 AS UNSIGNED) ~ 2.0;
-> -1.0
```

Если вы используете строку в арифметической операции, она конвертируется в число с плавающей точкой.

Работа с беззнаковыми величинами изменилась в MySQL 4.0 таким образом, чтобы правильно поддерживать значения `BIGINT`. Если у вас есть какой-то код, который нужно выполнять и в MySQL 4.0, и в MySQL 3.23, вероятно, вы не сможете использовать функцию `CAST()`. Вы можете применить следующую технику, чтобы получить результат со знаком при вычитании двух беззнаковых целых столбцов `ucol1` и `ucol2`:

```
mysql> SELECT (ucol1+0.0)-(ucol2+0.0) FROM ...;
```

Идея заключается в том, чтобы преобразовать операнды в значения с плавающей точкой перед выполнением вычитания.

Если при переносе старых приложений MySQL в MySQL 4.0 возникают проблемы, можно указать опцию `--sql-mode=NO_UNSIGNED_SUBTRACTION` при запуске `mysqld`. Однако, в таком случае вы не сможете эффективно использовать тип столбцов `BIGINT UNSIGNED`.

## 5.8. Другие функции

### 5.8.1. Поразрядные функции

MySQL использует арифметику `BIGINT` (64-разрядную) в поразрядных операциях, поэтому эти операции имеют максимальный предел в 64 бита.

- |. Поразрядное ИЛИ:

```
mysql> SELECT 29 | 15;  
-> 31
```

Результат – беззнаковое 64-разрядное целое.

- &. Поразрядное И:

```
mysql> SELECT 29 & 15;  
-> 13
```

Результат – беззнаковое 64-разрядное целое.

- ^. Поразрядное исключающее ИЛИ:

```
mysql> SELECT 1 ^ 1;  
-> 0  
mysql> SELECT 1 ^ 0;  
-> 1  
mysql> SELECT 11 ^ 3;  
-> 8
```

Результат – беззнаковое 64-разрядное целое.

- <<. Поразрядный сдвиг числа `BIGINT` влево.

```
mysql> SELECT 1 << 2;  
-> 4
```

Результат – беззнаковое 64-разрядное целое.

- >>. Поразрядный сдвиг числа `BIGINT` вправо.

```
mysql> SELECT 4 >> 2;  
-> 1
```

Результат – беззнаковое 64-разрядное целое.

- ~. Поразрядное инвертирование.

```
mysql> SELECT 5 & ~1;  
-> 4
```

Результат – беззнаковое 64-разрядное целое.

- BIT\_COUNT(N). Возвращает количество битов аргумента N, которые установлены в единицу.

```
mysql> SELECT BIT_COUNT(29);  
-> 4
```

## 5.8.2. Функции шифрования

Функции, описанные в данном разделе, шифруют и дешифруют значения данных. Если вы хотите сохранять результаты функции шифрования, которые могут иметь произвольные байтовые значения, применяйте столбцы типа BLOB вместо CHAR или VARCHAR, чтобы избежать потенциальных проблем с удалением завершающих пробелов, которые изменяют значения данных.

- AES\_ENCRYPT(*строка*, *строка\_ключа*)  
AES\_DECRYPT(*зашифрованная\_строка*, *строка\_ключа*)

Эти функции позволяют выполнять шифрование и дешифрацию данных с использованием официального алгоритма AES (Advanced Encryption Standard), ранее известного как “Rijndael”. Применяется кодирование с 128-разрядным ключом, но можно расширить его до 256 разрядов, должным образом изменив исходные тексты. Мы выбрали длину ключа 128, поскольку он работает намного быстрее и при этом обеспечивает приемлемый уровень безопасности.

Входные аргументы могут иметь любую длину. Если любой из них равен NULL, результатом функции также будет NULL.

Поскольку AES – алгоритм блочного типа, дополнение применяется для строк с нечетным количеством символов, и поэтому длина результирующей строки может быть рассчитана как  $16 * (\text{trunc}(\text{длина\_строки}/16) + 1)$ .

Если функция AES\_DECRYPT() обнаруживает неверные данные или неправильное дополнение, она возвращает NULL. Однако существует вероятность, что AES\_DECRYPT() вернет значение, не равное NULL (возможно, “мусор”), если входные данные или ключ не верны.

Вы можете использовать AES-функции для сохранения данных в зашифрованной форме, модифицировав существующие запросы:

```
INSERT INTO t VALUES (1, AES_ENCRYPT('text', 'password'));
```

Можно обеспечить даже более высокий уровень безопасности, если не передавать значение ключа для каждого запроса, а сохранять его в переменной сервера во время подключения, например:

```
SELECT @password := 'my password';  
INSERT INTO t VALUES (1, AES_ENCRYPT('text', @password));
```

Функции AES\_ENCRYPT() и AES\_DECRYPT() были добавлены в MySQL 4.0.2 и могут рассматриваться как наиболее криптографически безопасные функции, доступные в MySQL на текущий момент.



■ `DECODE(зашифрованная_строка, строка_пароля)`

Расшифровывает строку *зашифрованная\_строка*, используя значение *строка\_пароля* в качестве пароля. Аргумент *зашифрованная\_строка* должен быть строкой, ранее возвращенной функцией `ENCODE()`.

■ `ENCODE(строка, строка_пароля)`

Шифрует строку *строка*, используя значение *строка\_пароля* в качестве пароля. Для расшифровки результата применяется функция `DECODE()`. Результатом является бинарная строка той же длины, что и *строка*. Если нужно сохранить ее в столбце, применяйте тип `BLOB`.

■ `DES_DECRYPT(зашифрованная_строка[, строка_ключа])`

Расшифровывает строку *зашифрованная\_строка*, зашифрованную с помощью `DES_ENCRYPT()`. В случае ошибки возвращает `NULL`. Следует отметить, что эта функция работает, только если MySQL настроен на поддержку SSL. Если не указан аргумент *строка\_ключа*, `DES_DECRYPT()` проверяет первый байт зашифрованной строки для определения номера DES-ключа, использованного при шифровании исходной строки, а затем читает ключ из файла DES-ключей для расшифровки сообщения. Чтобы это работало, пользователь должен иметь привилегию `SUPER`. Файл ключей может быть указан с помощью опции сервера `--des-key-file`.

Если вы передаете этой функции аргумент *строка\_ключа*, он используется в качестве ключа при расшифровке сообщения.

Если аргумент *зашифрованная\_строка* не выглядит как зашифрованная строка, MySQL вернет строку *зашифрованная\_строка* без изменений.

Функция `DES_DECRYPT()` была добавлена в MySQL 4.0.1.

■ `DES_ENCRYPT(строка[, (номер_ключа|строка_ключа)])`

Шифрует строку с помощью заданного ключа, используя тройной DES-алгоритм. В случае ошибки возвращает `NULL`.

Следует отметить, что эта функция работает, только если MySQL настроен на поддержку SSL. Ключ шифрования выбирается на базе второго аргумента `DES_ENCRYPT()`, если таковой указан:

Аргумент	Описание
Не указан	Первый ключ из используемого файла DES-ключей.
<i>номер_ключа</i>	Заданный номер ключа (0–9) из используемого файла DES-ключей.
<i>строка_ключа</i>	Предоставленная строка ключа используется для шифрования строки <i>строка</i> .

Имя файла ключей указывается в опции сервера `--des-key-file`.

Возвращаемое значение является бинарной строкой, в которой первый символ – это `CHAR(128|номер_ключа)`.

128 добавлено для того, чтобы упростить распознавание ключа шифрования. Если применяется строковый ключ, *номер\_ключа* будет равен 127.

Длина строки результата рассчитывается как  $\text{новая\_длина} = \text{оригинальная\_длина} + (8 - (\text{оригинальная\_длина} \% 8)) + 1$ . Каждая строка в файле DES-ключей имеет следующий формат: *номер\_ключа строка\_ключа\_des*.

Каждый *номер\_ключа* должен быть числом в диапазоне от 0 до 9. Строки в файле могут следовать в любом порядке. *строка\_ключа\_des* – это строка, которая будет использоваться для шифрования сообщения. Между номером и ключом должен быть, по меньшей мере, один пробел. Первый ключ является ключом по умолчанию, который применяется в случае, если не указан аргумент *строка\_ключа* в функции `DES_ENCRYPT()`.

Можно указать MySQL на необходимость чтения новых значений ключа из файла ключей с помощью оператора `FLUSH DES_KEY_FILE`. Это требует наличия привилегии `RELOAD`.

Одна выгода от наличия набора ключей по умолчанию состоит в том, что это дает возможность приложениям проверить наличие зашифрованных значений столбцов, без необходимости предоставления конечному пользователю прав непосредственного доступа к ключам.

```
mysql> SELECT customer_address FROM customer_table WHERE  
-> crypted_credit_card = DES_ENCRYPT('credit_card_number');
```

`DES_ENCRYPT()` была добавлена в MySQL 4.0.1.

- `ENCRYPT(строка[, нач])`

Шифрует строку *строка*, используя системный вызов Unix `crypt()`. Аргумент *нач* должен быть строкой из двух символов. (Начиная с версии MySQL 3.22.16, *нач* может быть длиннее 2 символов.)

```
mysql> SELECT ENCRYPT('hello');  
-> 'VxuFAJXVARROc'
```

`ENCRYPT()` игнорирует все, кроме первых восьми символов аргумента *строка*, по крайней мере, в некоторых системах. Это поведение определяется реализацией лежащего в основе системного вызова `crypt()`.

Если функция `crypt()` не доступна в вашей системе, `ENCRYPT()` всегда возвращает `NULL`. По этой причине мы рекомендуем применять вместо этой функции `MD5()` или `SHA1()`, поскольку эти две функции представлены на всех платформах.

- `MD5(строка)`

Вычисляет 128-разрядную контрольную сумму MD5 для аргумента *строка*. Значение возвращается в виде 32-разрядной шестнадцатеричной строки или же `NULL`, если аргумент равен `NULL`. Возвращаемое значение может быть использовано, например, в качестве хэш-ключа.

```
mysql> SELECT MD5('testing');  
-> 'ae2b1fca515949e5d54fb22b8ed95575'
```

Дополнительные сведения можно найти в документе “RSA Data Security, Inc. MD5 Message-Digest Algorithm” (“Компания RSA Data Security, Inc. Алгоритм вычисления дайджеста сообщения”).

Функция `MD5()` появилась в MySQL 3.23.2.

- `OLD_PASSWORD(строка)`

Функция `OLD_PASSWORD()` стала доступной, начиная с MySQL 4.1, когда была изменена реализация `PASSWORD()` для повышения безопасности. `OLD_PASSWORD()` воз-

возвращает такое же значение, какое возвращала старая реализация (до версии 4.1) функции `PASSWORD()`.

■ **PASSWORD(строка)**

Вычисляет и возвращает строку пароля по значению пароля *строка*, заданному простым текстом, или же `NULL`, если аргумент равен `NULL`. Эта функция применяется для шифрования паролей, сохраняемых в столбце `Password` таблицы привилегий `user`.

```
mysql> SELECT PASSWORD('badpwd');
-> '7f84554057dd964b'
```

Шифрование функцией `PASSWORD()` является односторонним (то есть необратимым).

### На заметку!

Функция `PASSWORD()` используется системой аутентификации сервера MySQL, которая не должна быть задействованной в ваших собственных приложениях. Для этой цели вместо нее применяйте функции `MD5()` и `SHA1()`. Кроме того, в документе RFC2195 можно найти дополнительную информацию по работе с паролями и аутентификацией прикладных приложений.

■ **SHA1(строка)**

**SHA(строка)**

Вычисляет 160-разрядную контрольную сумму SHA1 для строки, как описано в документе RFC3174 ("Secure Hash Algorithm" – "Защищенный алгоритм хэширования"). Возвращаемое значение – шестнадцатеричная строка длиной в 40 цифр или `NULL`, если аргумент равен `NULL`. Одним из возможных применений этой функции является получение хэш-ключа. Вы также можете использовать ее как безопасную криптографическую функцию для сохранения паролей.

```
mysql> SELECT SHA1('abc');
-> 'a9993e364706816aba3e25717850c26c9cd0d89d'
```

Функция `SHA()` была добавлена в MySQL 4.0.2 и может рассматриваться как более безопасный эквивалент с точки зрения криптографии, нежели `MD5()`. `SHA()` является синонимом для `SHA1()`.

## 5.8.3. Информационные функции

■ **BENCHMARK(количество, выражение)**

Функция `BENCHMARK()` выполняет выражение *выражение* в точности *количество* раз. Она может использоваться для определения того, насколько быстро MySQL выполняет выражение. Возвращаемый результат всегда равен 0. Предполагаемое применение – в среде клиента `mysql`, который сообщает время выполнения запроса:

```
mysql> SELECT BENCHMARK(1000000, ENCODE('hello', 'goodbye'));
+-----+
| BENCHMARK(1000000, ENCODE('hello', 'goodbye')) |
+-----+
| 0 |
+-----+
1 row in set (4.74 sec)
```

Время, которое сообщает `mysql` – это время обслуживания клиента, а не потраченное центральным процессором время на стороне сервера. Рекомендуется выполнить `BENCHMARK()` несколько раз, и интерпретировать результат в зависимости от степени загрузки сервера.

#### ■ `CHARSET(строка)`

Возвращает набор символов аргумента строка.

```
mysql> SELECT CHARSET('abc');
-> 'latin1'
mysql> SELECT CHARSET(CONVERT('abc' USING utf8));
-> 'utf8'
mysql> SELECT CHARSET(USER());
-> 'utf8'
```

Функция `CHARSET()` была добавлена в MySQL 4.1.0.

#### ■ `COERCIBILITY(строка)`

Возвращает значение принуждения сопоставления для аргумента строка.

```
mysql> SELECT COERCIBILITY('abc' COLLATE latin1_swedish_ci);
-> 0
mysql> SELECT COERCIBILITY('abc');
-> 3
mysql> SELECT COERCIBILITY(USER());
-> 2
```

Возвращаемые значения имеют следующий смысл:

Принуждение	Значение
0	Явное сравнение
1	Нет сравнения
2	Неявное сравнение
3	Принуждаемое

Меньшие значения обладают большим приоритетом.

Функция `COERCIBILITY()` появилась в версии MySQL 4.1.0.

#### ■ `COLLATION(строка)`

Возвращает наименование порядка сопоставления символьного набора для заданного аргумента строка.

```
mysql> SELECT COLLATION('abc');
-> 'latin1_swedish_ci'
mysql> SELECT COLLATION(_utf8'abc');
-> 'utf8_general_ci'
```

Функция `COLLATION()` была добавлена в MySQL 4.1.0.

#### ■ `CONNECTION_ID()`

Возвращает идентификатор соединения (идентификатор потока) текущего сеанса. Каждое клиентское соединение получает свой собственный уникальный идентификатор.

```
mysql> SELECT CONNECTION_ID();
-> 23786
```

Функция `CONNECTION_ID()` была введена в MySQL 3.23.14.

■ **CURRENT\_USER()**

Возвращает комбинацию имени пользователя и имени хоста после аутентификации в текущем сеансе. Это значение соответствует пользовательской учетной записи MySQL, которая определяет ваши права доступа. Оно может отличаться от значения, возвращаемого функцией `USER()`.

```
mysql> SELECT USER();
-> 'davida@localhost'
mysql> SELECT * FROM mysql.user;
ERROR 1044: Access denied for user: '@localhost' to
database 'mysql'
mysql> SELECT CURRENT_USER();
-> '@localhost'
```

Приведенный выше пример иллюстрирует, что, несмотря на то, что клиент имеет имя `davida` (как показывает функция `USER()`), сервер аутентифицировал клиента, использующего анонимный доступ (что видно по пустой части имени пользователя в значении `CURRENT_USER()`). Единственной причиной, почему такое может случиться, является отсутствие учетной записи для `davida` в таблице привилегий.

Функция `CURRENT_USER()` была добавлена в MySQL 4.0.6.

■ **DATABASE()**

Возвращает имя базы данных по умолчанию (текущей базы данных).

```
mysql> SELECT DATABASE();
-> 'test'
```

Если текущей базы данных нет, `DATABASE()` возвращает `NULL`, начиная с версии MySQL 4.1.1, либо пустую строку в более ранних версиях.

■ **FOUND\_ROWS()**

Оператор `SELECT` может включать конструкцию `LIMIT` для ограничения количества строк, которые сервер возвращает клиенту. В некоторых случаях желательно знать, сколько строк сервер вернул бы без конструкции `LIMIT`, но без повторного выполнения запроса. Чтобы получить значение счетчика строк, включите опцию `SQL_CALC_FOUND_ROWS` в состав оператора `SELECT`, после чего вызовите `FOUND_ROWS()`:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name
-> WHERE id > 100 LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

Второй оператор `SELECT` вернет число, показывающее, сколько строк первый оператор `SELECT` вернул бы, будь он без конструкции `LIMIT`. (Если предшествующий оператор `SELECT` не содержит опции `SQL_CALC_FOUND_ROWS`, то `FOUND_ROWS()` может вернуть другое значение.)

Следует отметить, что когда используется `SELECT SQL_CALC_FOUND_ROWS`, то MySQL приходится посчитать, сколько строк будет в полном результирующем наборе. Однако это делается быстрее, чем если запустить запрос снова без конструкции `LIMIT`, поскольку результирующий набор не приходится отправлять клиенту.

`SQL_CALC_FOUND_ROWS` и `FOUND_ROWS()` могут оказаться удобными в ситуациях, когда нужно ограничить число строк, возвращаемых запросом, но при этом определить количество строк в полном результирующем наборе, не запуская запрос снова. Примером может служить Web-сценарий, который представляет постранич-

ный список ссылок на страницы, содержащие другие разделы результата поиска. Функция `FOUND_ROWS()` позволяет определить, сколько других страниц необходимо для отображения оставшейся части результата.

Применение `SQL_CALC_FOUND_ROWS` и `FOUND_ROWS()` более сложно для запросов с `UNION`, чем для простых операторов `SELECT`, потому что `LIMIT` может встретиться в `UNION` во многих местах. Они могут касаться отдельных операторов `SELECT` в составе `UNION` либо общего результата `UNION` в целом.

Цель `SQL_CALC_FOUND_ROWS` для `UNION` состоит в том, что он должен вернуть количество строк, которые будут возвращены без глобального `LIMIT`. Условия применения `SQL_CALC_FOUND_ROWS` с `UNION` перечислены ниже:

- Ключевое слово `SQL_CALC_FOUND_ROWS` должно указываться в первом операторе `SELECT`.
- Значение `FOUND_ROWS()` будет точным только при условии применения `UNION ALL`. Если указано `UNION` без `ALL`, происходит исключение дубликатов, и значение `FOUND_ROWS()` будет лишь приблизительным.
- Если в `UNION` не присутствует `LIMIT`, то `SQL_CALC_FOUND_ROWS` игнорируется и возвращается количество строк во временной таблице, которая создается для выполнения `UNION`.

`SQL_CALC_FOUND_ROWS` и `FOUND_ROWS()` доступны, начиная с MySQL 4.0.0.

#### ■ `LAST_INSERT_ID()`

`LAST_INSERT_ID(выражение)`

Возвращает последнее автоматически сгенерированное значение, которое было вставлено в столбец `AUTO_INCREMENT`.

```
mysql> SELECT LAST_INSERT_ID();
-> 195
```

Последний идентификатор, который был сгенерирован, поддерживается на сервере на основе соединений. Это означает, что значение, возвращаемое клиенту, соответствует последнему сгенерированному значению `AUTO_INCREMENT` в сеансе данного клиента. Оно никак не может быть затронуто другими клиентами, равно как не требует блокировок или транзакций.

Значение, возвращаемое `LAST_INSERT_ID()` не изменяется, если вы обновляете столбец `AUTO_INCREMENT` в строке не с помощью “магических” значений (то есть, не `NULL` и не `0`).

Если вы вставляете много строк одним оператором, `LAST_INSERT_ID()` возвращает значение для первой вставленной строки. Цель этого состоит в том, чтобы облегчить воспроизведение того же оператора `INSERT` на другом сервере.

Если указан аргумент *выражение*, значение аргумента возвращается функцией и запоминается как следующее значение, которое `LAST_INSERT_ID()` вернет при следующем вызове. Это можно использовать для эмуляции последовательностей:

- Создать таблицу для хранения счетчика последовательности и инициализировать его:

```
mysql> CREATE TABLE sequence (id INT NOT NULL);
mysql> INSERT INTO sequence VALUES (0);
```

- Использовать таблицу для генерации последовательности чисел:

```
mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);  
mysql> SELECT LAST_INSERT_ID();
```

Оператор UPDATE увеличивает счетчик последовательности и заставляет следующий вызов LAST\_INSERT\_ID() возвращать измененное значение. Функция C API по имени mysql\_insert\_id() также может быть использована для получения этого значения.

Вы можете генерировать последовательности без вызова LAST\_INSERT\_ID(), но польза от ее применения заключается в том, что значение идентификатора подерживается сервером как последнее автоматически сгенерированное значение. Это обеспечивает безопасное использование в многопользовательской среде, поскольку множество клиентов могут отправлять операторы UPDATE и получать свои собственные значения последовательности через оператор SELECT (или mysql\_insert\_id()), никак не влияя и не подвергаясь влиянию других клиентов, для которых генерируются их собственные значения последовательности.

- SESSION\_USER()

SESSION\_USER() – это синоним для USER().

- SYSTEM\_USER()

SYSTEM\_USER() – это синоним для USER().

- USER(). Возвращает имя текущего пользователя MySQL и имя хоста, с которого он подключился.

```
mysql> SELECT USER();  
-> 'davida@localhost'
```

Значение представляет имя пользователя, которое было указано во время подключения к серверу, и имя компьютера-хоста, с которого произошло подключение. Возвращаемое значение может отличаться от того, которое выдает CURRENT\_USER().

До версии MySQL 3.22.11 значение, возвращаемое функцией, не включало в себя имени хоста. Вы можете извлечь имя пользователя, независимо от того, включает ли значение имя хоста или нет, следующим образом:

```
mysql> SELECT SUBSTRING_INDEX(USER(), '@', 1);  
-> 'davida'
```

Начиная с MySQL 4.1, USER() возвращает значение в наборе символов utf8, поэтому вы также должны убедиться, что строковый литерал '@' интерпретируется в рамках этого набора символов:

```
mysql> SELECT SUBSTRING_INDEX(USER(), _utf8'@', 1);  
-> 'davida'
```

- VERSION(). Возвращает строку, содержащую информацию о версии сервера MySQL:

```
mysql> SELECT VERSION();  
-> '4.1.2-alpha-log'
```

Следует отметить, что если строка версии заканчивается на '-log', это означает, что регистрация в журнале включена.

## 5.8.4. Различные функции

- **FORMAT(*X*, *D*)**. Форматирует число *X* в формате, подобном '#,###,###.##', округленное до *D* разрядов, и возвращает результат в виде строки. Если *D* равно 0, результат не имеет десятичной точки или дробной части.

```
mysql> SELECT FORMAT(12332.123456, 4);
-> '12,332.1235'
mysql> SELECT FORMAT(12332.1,4);
-> '12,332.1000'
mysql> SELECT FORMAT(12332.2,0);
-> '12,332'
```

- **GET\_LOCK(*строка*, *таймаут*)**. Пытается получить блокировку по имени, заданном строкой *строка*, с таймаутом длительностью *таймаут* секунд. Возвращает 1, если блокировка получена успешно, 0, если время ожидания превысило таймаут (например, из-за того, что другой клиент уже заблокировал это имя), либо NULL, если произошла ошибка (такая как переполнение памяти или уничтожение потока командой `mysqladmin kill`). Если у вас есть блокировка, полученная через **GET\_LOCK()**, она снимается после выполнения **RELEASE\_LOCK()**, нового вызова **GET\_LOCK()** либо разрыва соединения (как нормального, так и не нормального).

Эта функция может использоваться для реализации блокировок приложений или для эмуляции блокировок записей. Имена блокируются в глобальном контексте сервера. Если имя заблокировано одним клиентом, **GET\_LOCK()** блокирует любой запрос другого клиента на получение блокировки с тем же именем. Это позволяет клиентам согласовывать попытки доступа к общим ресурсам.

```
mysql> SELECT GET_LOCK('lock1',10);
-> 1
mysql> SELECT IS_FREE_LOCK('lock2');
-> 1
mysql> SELECT GET_LOCK('lock2',10);
-> 1
mysql> SELECT RELEASE_LOCK('lock2');
-> 1
mysql> SELECT RELEASE_LOCK('lock1');
-> NULL
```

Следует отметить, что второй вызов **RELEASE\_LOCK()** возвращает NULL, поскольку блокировка 'lock1' была автоматически снята вторым вызовом **GET\_LOCK()**.

- **INET\_ATON(*выражение*)**. Принимает сетевой адрес, представленный четырьмя числами с разделителем-точкой, и возвращает целое, представляющее числовое значение адреса. Адрес может быть 4- или 8-байтным.

```
mysql> SELECT INET_ATON('209.207.224.40');
-> 3520061480
```

Сгенерированное число всегда содержит байты в порядке, заданном в сетевом адресе. Для только что приведенного примера оно вычисляется как  $209 * 256^3 + 207 * 256^2 + 224 * 256 + 40$ .

Начиная с версии MySQL 4.1.2, **INET\_ATON()** также понимает IP-адреса в сокращенной форме:



```
mysql> SELECT INET_ATON('127.0.0.1'), INET_ATON('127.1');  
-> 2130706433, 2130706433
```

Функция `INET_ATON()` была добавлена в MySQL 3.23.15.

- `INET_NTOA(выражение)`. Принимает сетевой адрес в виде числа (4- или 8-байтного), возвращает адрес, представленный строкой, состоящей из четырех чисел, разделенных точкой.

```
mysql> SELECT INET_NTOA(3520061480);  
-> '209.207.224.40'
```

Функция `INET_NTOA()` появилась в MySQL 3.23.15.

- `IS_FREE_LOCK(строка)`. Проверяет, свободна ли блокировка с именем строка. Возвращает 1, если блокировка свободна (никем не используется), 0, если занята, и NULL в случае ошибки.

Функция `IS_FREE_LOCK()` была добавлена в MySQL 4.0.2.

- `IS_USED_LOCK(строка)`. Проверяет, используется ли блокировка с именем строка (то есть, установлена ли она). Если это так, возвращает идентификатор соединения клиента, который удерживает блокировку. В противном случае возвращает NULL.

Функция `IS_USED_LOCK()` появилась в версии MySQL 4.1.0.

- `MASTER_POS_WAIT(имя_журнала, позиция_в_журнале[, таймаут])`

Эта функция удобна для управления синхронизацией главный/подчиненный. Блокирует главный сервер до тех пор, пока подчиненный сервер не прочтает и не проведет все изменения вплоть до указанной позиции в бинарном журнале главного сервера. Возвращаемое значение представляет количество событий в журнале, обработку которых нужно выполнить системе синхронизации, чтобы дойти до указанной позиции. Функция возвращает NULL, если поток SQL подчиненного сервера не запущен, либо информация о главном сервере не инициализирована на подчиненном, либо указаны неправильные аргументы. Возвращает -1, если истекло время таймаута. Если подчиненный сервер уже достиг указанной позиции, функция возвращает управление немедленно.

Если задано значение *таймаут*, `MASTER_POS_WAIT()` прекращает ожидание, когда истекают *таймаут* секунд. Значение *таймаут* может быть больше 0, а если его значение равно 0 или является отрицательным, то ожидания нет.

Функция `MASTER_POS_WAIT()` добавлена в MySQL 3.23.32. Аргумент *таймаут* появился в версии 4.0.10.

- `RELEASE_LOCK(строка)`. Снимает блокировку с именем строка, которая была получена с помощью функции `GET_LOCK()`. Возвращает 1, если блокировка снята, 0, если блокировка была установлена другим потоком (а значит, не может быть снята), и NULL, если блокировка с таким именем не существует. Блокировка не существует, если не была установлена вызовом `GET_LOCK()`, либо она уже снята.

Вместе с `RELEASE_LOCK()` удобно использовать оператор `DO` (см. раздел 6.1.2).

- `UUID()`. Возвращает Универсальный Уникальный Идентификатор (Universal Unique Identifier – UUID), сгенерированный в соответствии со спецификациями “DCE 1.1: Remote Procedure Call” (Appendix A) CAE (Common Applications Environment) (“DCE 1.1: Удаленный вызов процедур” (Приложение A) CAE (Об-

щая среда приложений)), опубликованными Open Group в октябре 1987 года (документ под номером C706).

Идентификатор UUID спроектирован как число, которое является глобально уникальным во времени и пространстве. Ожидается, что два вызова `UUID()` сгенерируют два разных значения, даже если эти два вызова произойдут на двух разных компьютерах, которые не подключены друг к другу.

UUID – это 128-разрядное число, представленное в виде строки, состоящей из пяти шестнадцатеричных чисел в формате `aaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee`:

- Первые три числа генерируются на основе временной метки.
- Четвертое число предохраняет темпоральную уникальность в случае, если значение временной метки теряет монотонность (например, из-за перехода на летнее время и обратно).
- Пятое число – это номер узла IEEE 802, который представляет пространственную уникальность. Случайное число подставляется в случае, если последнее недоступно (например, если компьютер-хост не имеет сетевой платы Ethernet, или нет возможности извлечь аппаратный адрес интерфейса вашего компьютера). В этом случае пространственная уникальность не может быть гарантирована. Однако, несмотря на это, коллизии крайне маловероятны. 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.

```
mysql> SELECT UUID();  
-> '6ccd780c-baba-1026-9564-0040f4311e29'
```

Следует отметить, что `UUID()` пока не работает с репликацией.

Функция `UUID()` появилась в версии MySQL 4.1.2.

## 5.9. Функции и модификаторы, применяемые в конструкции GROUP BY

### 5.9.1. Агрегатные функции GROUP BY

Если вы используете групповую функцию в операторе, содержащем конструкцию `GROUP BY`, это эквивалентно группировке всех строк.

- **AVG(выражение)**. Возвращает среднее значение выражения, указанного в аргументе *выражение*.

```
mysql> SELECT student_name, AVG(test_score)  
-> FROM student  
-> GROUP BY student_name;
```

- **BIT\_AND(выражение)**. Возвращает результат поразрядного И для всех битов в *выражение*. Вычисления выполняются с 64-разрядной (BIGINT) точностью.

Начиная с MySQL 4.0.17, эта функция возвращает 18446744073709551615, если нет подходящих строк. (Это беззнаковое значение BIGINT, в котором все биты установлены в 1). До версии 4.0.17 функция возвращала -1, если не было найдено подходящих строк.

- **BIT\_OR(выражение)**. Возвращает результат поразрядного ИЛИ для всех битов в выражение. Вычисления выполняются с 64-разрядной (BIGINT) точностью. Возвращает 0, если подходящие строки не найдены.
- **BIT\_XOR(выражение)**. Возвращает результат поразрядного исключающего ИЛИ для всех битов в выражение. Вычисления выполняются с 64-разрядной (BIGINT) точностью. Возвращает 0, если подходящие строки не найдены. Функция доступна, начиная с MySQL 4.1.1.
- **COUNT(выражение)**. Возвращает количество значений, отличных от NULL, в строках, извлеченных оператором SELECT.

```
mysql> SELECT student.student_name, COUNT(*)  
-> FROM student, course  
-> WHERE student.student_id=course.student_id  
-> GROUP BY student_name;
```

COUNT(\*) несколько отличается в том, что возвращает количество извлеченных строк, независимо от того, содержат они значения NULL или нет.

Функция COUNT(\*) оптимизирована для очень быстрого возврата, если SELECT обращается к одной таблице, и в нем нет конструкции WHERE. Например:

```
mysql> SELECT COUNT(*) FROM student;
```

Эта оптимизация касается только таблиц MyISAM и ISAM, поскольку для этих типов таблиц явное количество строк сохраняется и может быть извлечено очень быстро. Для транзакционных механизмов хранения (InnoDB, BDB) сохранение явного количества строк проблематично, поскольку одновременно может быть активно множество транзакций, каждая из которых изменяет количество строк.

- **COUNT(DISTINCT выражение[, выражение...])**. Возвращает подсчитанное количество различных значений, отличных от NULL.

В MySQL вы можете получить количество различных комбинаций выражений, которые не содержат NULL, передавая список этих выражений. В стандартном языке SQL потребуется выполнить конкатенацию всех выражений внутри COUNT(DISTINCT ...). COUNT(DISTINCT ...) появилась в MySQL 3.23.2.

- **GROUP\_CONCAT(выражение)**. Эта функция возвращает результирующую строку с объединенными значениями из группы. Полный синтаксис выглядит так:

```
GROUP_CONCAT([DISTINCT] выражение [, выражение ...]  
[ORDER BY {беззнаковое_целое | имя_столбца | выражение}  
[ASC | DESC] [, столбец ...])  
[SEPARATOR строковое_значение])
```

Пример:

```
mysql> SELECT student_name,  
-> GROUP_CONCAT(test_score)  
-> FROM student  
-> GROUP BY student_name;
```

или:

```
mysql> SELECT student_name,  
-> GROUP_CONCAT(DISTINCT test_score  
-> ORDER BY test_score DESC SEPARATOR ' ')
```

```
-> FROM student
-> GROUP BY student_name;
```

В MySQL вы можете получить сцепленное значение комбинации выражений. Дублирующие значения можно исключить с помощью DISTINCT. Если вы желаете отсортировать значения в результате, то должны воспользоваться конструкцией ORDER BY. Чтобы отсортировать в обратном порядке, добавляйте ключевое слово DESC к имени столбца, который сортируется конструкцией ORDER BY. По умолчанию принимается порядок по возрастанию. Это можно специфицировать явно ключевым словом ASC. После SEPARATOR следует строковое значение, которое должно вставляться между значениями результата.

По умолчанию применяется запятая. Вы можете вообще убрать разделитель, указав SEPARATOR ''.

Максимально возможную длину можно установить через системную переменную group\_concat\_max\_len. Синтаксис установки ее значения во время выполнения (здесь значение – беззнаковое целое) показан ниже:

```
SET [SESSION | GLOBAL] group_concat_max_len = значение;
```

Если максимальная длина установлена, результат усекается до этой длины.

### На заметку!

Все еще существует небольшое ограничение для GROUP\_CONCAT(), когда она используется с DISTINCT вместе с ORDER BY и значениями типа BLOB.

Функция GROUP\_CONCAT() была добавлена в MySQL 4.1.

- MIN(выражение)
- MAX(выражение)

Возвращает минимальное и максимальное значение *выражение* в группе. MIN() и MAX() могут принимать строковый аргумент, в этих случаях возвращаются минимальное и максимальное строковые значения.

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
-> FROM student
-> GROUP BY student_name;
```

Для MIN(), MAX() и других агрегатных функций в настоящее время MySQL сравнивает столбцы типа ENUM и SET по их строчным значениям вместо относительно положения строки в наборе. Это отличается от того, как их сравнивает конструкция ORDER BY. Подобное положение будет исправлено.

- STD(выражение)
- STDDEV(выражение)

Возвращает стандартное квадратичное отклонение *выражение* (корень квадратный из VARIANCE()). Это расширение стандарта SQL. Форма STDDEV() этой функции введена для совместимости с Oracle.

- SUM(выражение). Возвращает сумму *выражение*. Отметьте, что если результирующий набор пуст, функция возвращает NULL!
- VARIANCE(выражение). Возвращает стандартное отклонение *выражение* (рассматривая строки как сплошную популяцию, которая имеет количество строк в качестве знаменателя). Это расширение стандарта SQL, доступное только в MySQL 4.1 и последующих версиях.

## 5.9.2. Модификаторы GROUP BY

Начиная с MySQL 4.1.1, конструкцию GROUP BY предусматривает модификатор WITH ROLLUP, который добавляет несколько дополнительных строк к итоговому выводу. Эти строки представляют итоговые операции высшего уровня (суперагрегатные). ROLLUP, таким образом, позволяет ответить на вопросы на многих уровнях анализа в пределах одного запроса. Это может быть использовано, например, для представления поддержки OLAP-операций (Online Analytical Processing – онлайн-аналитической обработки).

Для примера предположим, что в таблице sales имеются столбцы year, country, product и profit для хранения информации о рентабельности торговли:

```
CREATE TABLE sales
(
    year INT NOT NULL,
    country VARCHAR(20) NOT NULL,
    product VARCHAR(32) NOT NULL,
    profit INT
);
```

Итоговая информация по годам может быть получена простым GROUP BY следующим образом:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year;
+-----+-----+
| year | SUM(profit) |
+-----+-----+
| 2000 |         4525 |
| 2001 |         3010 |
+-----+-----+
```

Здесь мы видим общую прибыль за каждый год, но если нужно просмотреть суммарную прибыль за все года, придется просуммировать отдельные показатели вручную, либо выдать еще один запрос.

Вместо этого можно запустить ROLLUP, который представляет оба уровня анализа в пределах одного запроса. Добавление модификатора WITH ROLLUP к конструкции GROUP BY заставляет запрос сгенерировать еще одну строку, которая показывает общий итог по всем годам:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year WITH ROLLUP;
+-----+-----+
| year | SUM(profit) |
+-----+-----+
| 2000 |         4525 |
| 2001 |         3010 |
| NULL |         7535 |
+-----+-----+
```

Строка общего итога идентифицируется значением NULL столбца year. ROLLUP порождает более сложный эффект, когда в конструкции GROUP BY участвуют несколько столбцов. В этом случае каждый раз, когда появляется “разрыв” (изменение в значении) в любом группируемом столбце кроме последнего, запрос генерирует дополнительную суперагрегатную строку.

Например, с использованием ROLLUP итоговая информация по таблице sales, сгруппированная по year, country и product, выглядит так:

```
mysql> SELECT year, country, product, SUM(profit)
-> FROM sales
-> GROUP BY year, country, product;
```

year	country	product	SUM(profit)
2000	Finland	Computer	1500
2000	Finland	Phone	100
2000	India	Calculator	150
2000	India	Computer	1200
2000	USA	Calculator	75
2000	USA	Computer	1500
2001	Finland	Phone	10
2001	USA	Calculator	50
2001	USA	Computer	2700
2001	USA	TV	250

Вывод показывает суммарные значения только на уровне анализа по году/стране/продукту. Когда добавляется ROLLUP, запрос выдает несколько дополнительных строк:

```
mysql> SELECT year, country, product, SUM(profit)
-> FROM sales
-> GROUP BY year, country, product WITH ROLLUP;
```

year	country	product	SUM(profit)
2000	Finland	Computer	1500
2000	Finland	Phone	100
2000	Finland	NULL	1600
2000	India	Calculator	150
2000	India	Computer	1200
2000	India	NULL	1350
2000	USA	Calculator	75
2000	USA	Computer	1500
2000	USA	NULL	1575
2000	NULL	NULL	4525
2001	Finland	Phone	10
2001	Finland	NULL	10
2001	USA	Calculator	50
2001	USA	Computer	2700
2001	USA	TV	250
2001	USA	NULL	3000
2001	NULL	NULL	3010
NULL	NULL	NULL	7535

Добавление ROLLUP к этому запросу привело к включению суммарной информации на четырех уровнях анализа, а не только одном. Вот как следует интерпретировать вывод ROLLUP:

- После каждого множества строк о продукте для данного года и страны генерируется дополнительная итоговая строка, представляющая итог по всем продуктам. Эти строки имеют в столбце product значения NULL.
- После каждого множества строк по данному году добавляется дополнительная итоговая строка, представляющая итог по всем странам и продуктам. В этих строках столбцы country и product имеют значения NULL.
- И, наконец, после всех остальных добавляется строка, показывающая общий итог для всех лет, стран и продуктов. Значения столбцов year, country и product в этой строке равны NULL.

### Другие соглашения относительно использования ROLLUP

Ниже описаны некоторые специфичные для MySQL аспекты реализации ROLLUP.

Когда вы применяете ROLLUP, вы не можете одновременно использовать конструкцию ORDER BY для сортировки результатов. Другими словами, ROLLUP и ORDER BY являются взаимоисключающими. Однако у вас все же есть определенная возможность повлиять на порядок сортировки. В MySQL конструкция GROUP BY сортирует результаты, и вы можете явно указывать ключевые слова ASC и DESC для столбцов, перечисленных в конструкции GROUP BY, чтобы указать порядок сортировки индивидуальных столбцов.

(Итоговые строки высшего уровня, добавленные ROLLUP, по-прежнему появляются после строк, по которым они вычисляются, независимо от порядка сортировки.)

LIMIT может быть использовано для ограничения количества возвращаемых клиенту строк. LIMIT указывается после ROLLUP, таким образом влияя и на дополнительные строки, сгенерированные ROLLUP, например:

```
mysql> SELECT year, country, product, SUM(profit)
-> FROM sales
-> GROUP BY year, country, product WITH ROLLUP
-> LIMIT 5;
```

year	country	product	SUM(profit)
2000	Finland	Computer	1500
2000	Finland	Phone	100
2000	Finland	NULL	1600
2000	India	Calculator	150
2000	India	Computer	1200

Применение LIMIT вместе с ROLLUP может дать результат, который труднее интерпретировать, поскольку теряется контекст, необходимый для понимания суперагрегатных строк.

Индикатор NULL в каждой суперагрегатной строке генерируется, когда строка отправляется клиенту. Сервер смотрит на крайние левые имена столбцов, перечисленные в конструкции GROUP BY, у которых изменяется значение. Для любого столбца результирующего набора с именем, лексически соответствующим любому из этих имен, устанавливается значение NULL. (Если вы указываете группируемые столбцы по номерам, сервер идентифицирует, какие столбцы устанавливать в NULL, по их номерам.)

Так как значения NULL в суперагрегатных столбцах помещаются в результирующий набор на поздней стадии обработки запроса, вы не можете проверять их на предмет зна-

чения NULL внутри самого запроса. Например, нельзя добавить `HAVING product IS NULL` к запросу, чтобы исключить из результата все, кроме суперагрегатных строк.

С другой стороны, значения NULL появляются именно как NULL на клиентской стороне и могут быть протестированы с использованием любого клиентского программного интерфейса MySQL.

### 5.9.3. GROUP BY со скрытыми полями

MySQL расширяет применение GROUP BY таким образом, что вы можете использовать столбцы или вычисления в списке SELECT, которые не появляются в конструкции GROUP BY. То есть предполагается применение *любого возможного значения для данной группы*. Вы можете использовать это для получения более высокой производительности за счет отсутствия необходимости в сортировке и группировании по ненужным позициям. Например, вам не потребуется группировать по `customer.name` в следующем запросе:

```
mysql> SELECT order.custid, customer.name, MAX(payments)
-> FROM order, customer
-> WHERE order.custid = customer.custid
-> GROUP BY order.custid;
```

В стандартном SQL вы должны добавить `customer.name` в конструкцию GROUP BY. В MySQL это излишне, если только вы не работаете в режиме ANSI.

*Не используйте* это средство, если значения столбцов, пропущенных в GROUP BY, не являются уникальными в пределах группы. В таком случае могут быть получены непредсказуемые результаты.

В некоторых случаях можно использовать MIN() и MAX() для получения специфических значений столбцов, если они не уникальны. Приведенное ниже выражение выдает значение `column` из строки, содержащей наименьшее значение столбца `sort`:

```
SUBSTR(MIN(CONCAT(RPAD(sort, 6, ' '), column)), 7)
```

Следует отметить, что если вы используете MySQL версии 3.22 или предшествующей, либо если вы пытаетесь следовать стандарту SQL, то нельзя использовать выражения в предложениях ORDER BY и GROUP BY. Эти ограничения можно обойти с помощью псевдонимов выражений:

```
mysql> SELECT id, FLOOR(value/100) AS val FROM имя_таблицы
-> GROUP BY id, val ORDER BY val;
```

В MySQL 3.23 и выше псевдонимы не обязательны. В этих версиях в конструкциях ORDER BY и GROUP BY можно использовать выражения, например:

```
mysql> SELECT id, FLOOR(value/100) FROM имя_таблицы ORDER BY RAND();
```



## Синтаксис операторов SQL

**В** настоящей главе описан синтаксис операторов SQL, который поддерживается в MySQL.

### 6.1. Операторы манипуляции данными

#### 6.1.1. Синтаксис DELETE

Однотабличный синтаксис:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM имя_таблицы
      [WHERE определение_where]
      [ORDER BY ...]
      [LIMIT количество_строк]
```

Многотабличный синтаксис:

```
DELETE {LOW_PRIORITY} {QUICK} [IGNORE]
      имя_таблицы[.*] [, имя_таблицы[.*] ...]
FROM табличные_ссылки
      [WHERE определение_where]
```

Или:

```
DELETE {LOW_PRIORITY} [QUICK] [IGNORE]
      FROM имя_таблицы[.*] [, имя_таблицы[.*] ...]
      USING табличные_ссылки
      [WHERE определение_where]
```

DELETE удаляет строки из таблицы *имя\_таблицы*, удовлетворяющие условию *определение\_where* и возвращает количество удаленных строк.

В случае выдачи оператора DELETE без конструкции WHERE удаляются все строки. Если вам не нужно знать количество удаленных строк, то это можно сделать быстрее с помощью оператора TRUNCATE TABLE. См. раздел 6.1.9.

В MySQL 3.23 оператор DELETE без конструкции WHERE возвращает ноль.

В версии MySQL 3.23, если вы действительно хотите знать количество удаленных записей, и согласны на некоторое снижение производительности, можете использовать DELETE с конструкцией WHERE, условие которой является истинным для каждой записи.

Например:

```
mysql> DELETE FROM имя_таблицы WHERE 1>0;
```

Это гораздо медленнее, чем DELETE FROM имя\_таблицы без конструкции WHERE, поскольку строки удаляются по одной.

Если вы удаляете строку, содержащую максимальное значение столбца AUTO\_INCREMENT, это значение будет повторно использовано в таблицах ISAM и BDB, но не в таблицах MyISAM или InnoDB. Если вы удаляете все строки таблицы оператором DELETE FROM имя\_таблицы (без конструкции WHERE) в режиме AUTOCOMMIT, последовательность стартует для всех типов таблиц, кроме InnoDB и (начиная с MySQL 4.0) MyISAM. Существуют некоторые исключения упомянутого поведения, которые подробно рассматриваются в главе, посвященной InnoDB, книги *MySQL. Руководство администратора* (М.: Издательский дом "Вильямс", 2005, ISBN 5-8459-0805-1).

Для таблиц MyISAM и BDB можно объявить вторичный столбец AUTO\_INCREMENT в составном ключе. В этом случае повторное использование значений, удаленных с вершины последовательности, происходит даже для таблиц MyISAM.

Оператор DELETE поддерживает следующие модификаторы:

- Если указать ключевое слово LOW\_PRIORITY, выполнение DELETE откладывается до тех пор, пока другие клиенты не завершат чтение таблицы.
- Для таблиц MyISAM, если указано ключевое слово QUICK, механизм хранения не объединяет листья индекса в процессе удаления, что ускоряет некоторые типы операций DELETE.
- Ключевое слово IGNORE заставляет MySQL игнорировать все ошибки в процессе удаления строк. (Ошибки, обнаруженные на стадии синтаксического анализа, обрабатываются обычным образом.) Об ошибках, которые игнорируются вследствие применения этой опции, сообщается в виде предупреждений. Эта опция появилась в MySQL 4.1.1.

На скорость выполнения операции удаления могут также повлиять факторы, которые обсуждаются в главе, посвященной оптимизации, книги *MySQL. Руководство администратора*.

В таблицах MyISAM удаленные записи помещаются в связный список, и последующий оператор INSERT повторно использует старые позиции записей. Чтобы вернуть неиспользуемое пространство и уменьшить размеры файлов, применяйте оператор OPTIMIZE TABLE или утилиту myisamchk для реорганизации таблиц. OPTIMIZE TABLE проще, но myisamchk быстрее. См. раздел 6.5.2.5.

Специфичная для MySQL опция LIMIT количество\_строк оператора DELETE сообщает серверу максимальное число удаляемых строк перед возвратом управления клиенту. Это применяется для того, чтобы гарантировать, что выполнение какого-то особенного оператора DELETE не займет слишком много времени. Вы можете просто повторять этот оператор DELETE до тех пор, пока количество удаленных строк не окажется меньше, чем указано в значении LIMIT.

Если оператор DELETE включает конструкцию ORDER BY, то строки удаляются в указанном порядке. Это действительно удобно только в сочетании с опцией LIMIT. Например, следующий оператор находит строки, удовлетворяющие условию WHERE, сортирует их по порядку timestamp и удаляет первую (наиболее старую) из них:

```
DELETE FROM somelog
WHERE user = 'jcole'
ORDER BY timestamp
LIMIT 1
```

ORDER BY может использоваться вместе с DELETE, начиная с MySQL 4.0.0.

Начиная с MySQL 4.0, в операторе DELETE можно указывать множество таблиц, чтобы удалять строки из одной или более таблиц, в зависимости от определенного условия во множестве таблиц. Однако в многотабличном DELETE нельзя указывать ORDER BY или LIMIT.

Первый многотабличный синтаксис DELETE поддерживается, начиная с MySQL 4.0.0. Второй поддерживается, начиная с MySQL 4.0.2. Часть *табличные\_ссылки* перечисляет таблицы, участвующие в соединении. Этот синтаксис подробно описан в разделе 6.1.7.1.

При использовании первого синтаксиса удаляются только соответствующие строки из таблиц, перечисленных перед конструкцией FROM. При втором синтаксисе удаляются соответствующие строки из таблиц, перечисленных в конструкции FROM (перед USING).

Эффект заключается в том, что вы можете удалять строки из многих таблиц одновременно и также использовать дополнительные таблицы для поиска:

```
DELETE t1,t2 FROM t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```

или

```
DELETE FROM t1,t2 USING t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```

Эти операторы используют три таблицы для поиска строк, подлежащих удалению, но удаляют подходящие строки только в двух таблицах – t1 и t2.

Примеры показывают вложенные соединения, используя операцию запятой, но многотабличные операторы DELETE могут также применять любой тип соединения, допустимый для операторов SELECT, такой, например, как LEFT JOIN.

Синтаксис допускает '.' после имен таблиц для достижения совместимости с Access.

Если вы используете многотабличный оператор DELETE с таблицами InnoDB, у которых есть ограничения внешних ключей, оптимизатор MySQL может обрабатывать таблицы в порядке, отличающемся от заданного их отношением родительский/дочерний. В этом случае оператор завершается ошибкой и транзакция откатывается. Вместо этого удалите нужные записи из одной таблицы, и рассчитывайте на возможность ON DELETE, предоставляемую InnoDB, чтобы соответствующим образом модифицировать зависимые таблицы.

### На заметку!

В MySQL 4.0 для удаления записей вы должны обращаться к таблицам по их реальным именам. В MySQL 4.1 для обращения к именам таблиц необходимо использовать псевдонимы (если они указаны):

В MySQL 4.0:

```
DELETE test FROM test AS t1, test2 WHERE ...
```

В MySQL 4.1:

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

Причина того, что это не было реализовано в 4.0, состоит в том, что мы не хотели разрушать приложения MySQL 4.0, которые использовали старый синтаксис.

## 6.1.2. Синтаксис DO

DO выражение [, выражение] ...

Оператор DO выполняет выражения, но не возвращает никаких результатов. Это сокращение для SELECT выражение, ..., которое обладает тем преимуществом, что работает немного быстрее, если вас не интересует результат.

DO удобен в основном для функций, которые имеют сторонний эффект, таких как RELEASE\_LOCK().

## 6.1.3. Синтаксис HANDLER

```
HANDLER имя_таблицы OPEN [ AS псевдоним ]
HANDLER имя_таблицы READ имя_индекса { = | >= | <= | < }
(значение1, значение2, ...)
[ WHERE условие_where ] [ LIMIT ... ]
HANDLER имя_таблицы READ имя_индекса { FIRST | NEXT | PREV | LAST }
[ WHERE условие_where ] [ LIMIT ... ]
HANDLER имя_таблицы READ { FIRST | NEXT }
[ WHERE условие_where ] [ LIMIT ... ]
HANDLER имя_таблицы CLOSE
```

Оператор HANDLER предоставляет прямой доступ к интерфейсам механизма хранения таблиц. Он доступен для таблиц MyISAM, начиная с версии MySQL 4.0.0, и для таблиц InnoDB, начиная с версии MySQL 4.0.3.

Оператор HANDLER...OPEN открывает таблицу, делая ее доступной для последующих операторов HANDLER...READ. Этот табличный объект не разделяется другими потоками сервера и не закрывается до тех пор, пока не будет вызван HANDLER...CLOSE, либо не будет прервано выполнение потока. Если вы открываете таблицу, используя псевдоним, последующие обращения к ней оператором HANDLER также должны использовать псевдоним вместо реального имени таблицы.

Первый синтаксис HANDLER...READ извлекает строку, в которой указанный индекс удовлетворяет заданным значениям и выполняется условие WHERE. Если у вас есть индекс с составным ключом, указывайте значения индексных столбцов в виде списка с разделителями-запятыми. Можно указывать либо все столбцы индекса, либо левый (начальный) префикс списка индексных столбцов. Предположим, что индекс включает три столбца с именами col\_a, col\_b и col\_c, причем именно в таком порядке. В операторе HANDLER можно специфицировать значения для трех столбцов индекса либо для их левого подмножества, например:

```
HANDLER ... имя_индекса = (значение_col_a, значение_col_b, значение_col_c) ...
HANDLER ... имя_индекса = (значение_col_a, значение_col_b) ...
HANDLER ... имя_индекса = (значение_col_a) ...
```

Второй синтаксис HANDLER...READ извлекает строку из таблицы, удовлетворяющую условию WHERE, в порядке, заданном индексом.

Третий синтаксис HANDLER...READ извлекает строку из таблицы, удовлетворяющую условию WHERE, в натуральном порядке. Натуральный порядок – это тот порядок, в котором строки хранятся в файле табличных данных MyISAM. Эти операторы также работают и с таблицами InnoDB, но там нет такой концепции, поскольку нет отдельного файла данных таблицы.

Без конструкции `LIMIT` все формы оператора `HANDLER ... READ` извлекают одну строку, если она доступна. Чтобы вернуть определенное число строк, включите конструкцию `LIMIT`. Она имеет тот же синтаксис, что и в операторе `SELECT`. См. раздел 6.1.7.

`HANDLER ... CLOSE` закрывает таблицу, которая была открыта `HANDLER ... OPEN`.

### На заметку!

Чтобы использовать интерфейс `HANDLER` для обращения к первичному ключу (`PRIMARY KEY`) таблицы, указывайте идентификатор `'PRIMARY'` в кавычках:

```
HANDLER имя_таблицы READ 'PRIMARY' > (...);
```

В определенном смысле `HANDLER` является оператором низкого уровня. Например, он не обеспечивает целостности. То есть, `HANDLER ... OPEN` не делает снимка таблицы и не блокирует таблицу. Это означает, что после того, как этот оператор отправлен серверу, данные таблицы могут быть модифицированы (этим же или любым другим потоком) и такие модификации могут быть отражены лишь частично при сканировании с помощью `HANDLER ... NEXT` и `HANDLER ... PREV`.

Есть несколько причин, по которым стоит использовать интерфейс `HANDLER` вместо обычного оператора `SELECT`:

- `HANDLER` быстрее, чем `SELECT`.
  - Назначенный `handler`-объект выделяется механизму хранения при вызове `HANDLER ... OPEN`, и используется повторно при последующих вызовах `HANDLER` для таблицы; его не нужно повторно инициализировать каждый раз.
  - Меньше работы по синтаксическому анализу.
  - Никакой дополнительной нагрузки, связанной с проверкой запросов или оптимизацией.
  - Таблицу не надо блокировать между двумя вызовами `HANDLER`.
  - Интерфейс `HANDLER` не должен обеспечивать целостность данных (например, допускаются недействительные чтения), поэтому механизм хранения может использовать оптимизацию, которую обычно не позволяет `SELECT`.
- `HANDLER` значительно упрощает перенос в MySQL приложений, использующих ISAM-подобный интерфейс.
- `HANDLER` позволяет пересекать базу данных способом, который непросто (или даже невозможно) реализовать с помощью оператора `SELECT`. Интерфейс `HANDLER` — это более естественный способ доступа к данным из приложений, представляющих интерактивный пользовательский интерфейс доступа к базе данных.

## 6.1.4. Синтаксис INSERT

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
  [INTO] имя_таблицы [(имя_столбца, ...)]
  VALUES ((выражение | DEFAULT), ...), (...), ...
  [ ON DUPLICATE KEY UPDATE имя_столбца=выражение, ... ]
```

или:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
  [INTO] имя_таблицы
  SET имя_столбца={выражение | DEFAULT}, ...
  [ ON DUPLICATE KEY UPDATE имя_столбца=выражение, ... ]
```

или:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
[INTO] имя_таблицы [(имя_столбца,...)]
SELECT ...
```

Оператор INSERT вставляет новые строки в существующую таблицу. Формы INSERT...VALUES и INSERT...SET этого оператора вставляют строки на основании явно указанных значений. Форма INSERT...SELECT вставляет строки, выбирая их из другой таблицы или таблиц. Форма INSERT...VALUES со многими списками значений поддерживается MySQL 3.22.10 и более поздними версиями. INSERT...SELECT описывается далее в разделе 6.1.4.1.

имя\_таблицы — это имя таблицы, в которую нужно вставить строки. Столбцы, которым оператор присваивает значения, могут быть указаны следующим образом:

- Список имен столбцов в конструкции SET задает их явно.
- Если список столбцов не указан для INSERT...VALUES или INSERT...SELECT, то значения для всех столбцов таблицы должны быть представлены в списке VALUES() или с помощью SELECT. Если вы не знаете порядка следования столбцов в таблице, воспользуйтесь DESCRIBE имя\_таблицы для его получения.

Списки значений могут быть указаны несколькими способами:

- Любому столбцу, для которого не указано значение явно, присваивается значение по умолчанию. Например, если приведен список столбцов, который не включает в себя все столбцы таблицы, неназванные столбцы получают свои значения по умолчанию. Присвоение значений по умолчанию описано в разделе 6.2.5.

MySQL всегда имеет значения по умолчанию для всех столбцов. Это продиктовано необходимостью для MySQL работать как с транзакционными, так и с нетранзакционными таблицами.

С нашей точки зрения проверка контекста столбцов должна выполняться в приложении, а не на сервере базы данных.

### На заметку!

Если вы хотите заставить оператор INSERT генерировать ошибку, когда не указаны явно значения для всех столбцов таблицы, требующих не-NULL значений, можете сконфигурировать MySQL с использованием опции DONT\_USE\_DEFAULT\_USE. Это возможно, только при компиляции MySQL из исходных текстов.

- Вы можете воспользоваться ключевым словом DEFAULT, чтобы явно присвоить столбцу значение по умолчанию (новая возможность в MySQL 4.0.3). Это упрощает написание операторов INSERT, которые присваивают значения всем столбцам, кроме некоторых, поскольку позволяет избежать написания неполных списков VALUES, которые не включают всех столбцов таблицы. Иначе вам пришлось бы писать список имен столбцов, соответствующих каждому элементу в списке VALUES.
- Если и список столбцов, и список значений VALUES пусты, оператор INSERT создает строку, в которой все столбцы принимают значения по умолчанию.

```
mysql> INSERT INTO имя_таблицы () VALUES ();
```

- Выражение *выражение* может ссылаться на любой столбец, который упомянут ранее в списке значений. Например, вы можете делать это потому, что значение для col2 ссылается на col1, значение которого уже установлено:

```
mysql> INSERT INTO имя_таблицы (col1,col2) VALUES (15,col1*2);
```

Однако следующий оператор выполнить невозможно, так как col1 ссылается на col2, значение которого присваивается после col1:

```
mysql> INSERT INTO имя_таблицы (col1,col2) VALUES (col2*2,15);
```

Оператор INSERT поддерживает следующие модификаторы:

- Если вы указываете ключевое слово DELAYED, сервер помещает строку или строки, которые подлежат вставке, в буфер, и клиент, приславший этот оператор, может продолжать свою работу. Если таблица занята, сервер удерживает строки. Когда таблица освободится, он начнет их вставку, проверяя периодически, нет ли новых запросов на чтение этой таблицы. Если они есть, обслуживание очереди отложенных строк для вставки приостанавливается до тех пор, пока таблица не освободится вновь. См. раздел 6.1.4.2.
- Если указано ключевое слово LOW\_PRIORITY, выполнение вставки откладывается до тех пор, пока все другие клиенты не завершат чтение таблицы. Это касается также клиентов, которые начали чтение в то время, когда существующие клиенты уже читали, и тех, что обратились к таблице во время ожидания оператора INSERT LOW\_PRIORITY. Таким образом, есть вероятность, что клиент, приславший запрос INSERT LOW\_PRIORITY, будет ожидать весьма длительное время (или даже бесконечно долго) в среде с высокой нагрузкой по чтению. (Это отличается от оператора INSERT DELAYED, который позволяет клиенту продолжать работу.) См. раздел 6.1.4.2. Имейте в виду, что LOW\_PRIORITY обычно не должно использоваться с таблицами MyISAM, поскольку это препятствует параллельным вставкам.
- Если указано ключевое слово IGNORE в операторе INSERT со многими строками, любая строка, в которой дублируется значение столбцов уникального индекса или первичного ключа, игнорируется и не вставляется. Если вы не указываете IGNORE, операция вставки прерывается при обнаружении дублированных строк. Количество строк, вставленных в таблицу, можно определить вызовом функции C API `mysql_info()`.

Если вы используете конструкцию ON DUPLICATE KEY UPDATE (новая в MySQL 4.1.0), и вставляется строка с дублированным значением ключа уникального индекса или первичного ключа, то выполняется операция UPDATE старой строки. Например, если столбец объявлен как UNIQUE и уже содержит значение 1, то два следующих оператора дадут один и тот же эффект:

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3)
-> ON DUPLICATE KEY UPDATE c=c+1;

mysql> UPDATE table SET c=c+1 WHERE a=1;
```

#### На заметку!

Если столбец b тоже уникальный, то INSERT будет эквивалентен такому оператору UPDATE:

```
mysql> UPDATE table SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

Если условию `a=1 OR b=2` соответствуют несколько строк, то обновляется только одна строка. Вообще вам следует избегать применения конструкции `ON DUPLICATE KEY` с таблицами, у которых несколько уникальных ключей.

Начиная с MySQL 4.1.1, в конструкции `UPDATE` вы можете использовать функцию `VALUES (имя_столбца)`, чтобы сослаться на значения столбцов из части `INSERT` оператора `INSERT...UPDATE`. Другими словами, `VALUES (имя_столбца)` в конструкции `UPDATE` ссылается на значение `имя_столбца`, которое должно быть вставлено, если не обнаружится никаких конфликтов дублирования ключей. Эта функция особенно удобна при многострочных вставках. Функция `VALUES` имеет смысл только в операторе `INSERT...UPDATE`, и возвращает `NULL` во всех остальных случаях.

Ниже представлен пример:

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3), (4,5,6)
      -> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

Этот оператор идентичен следующим двум:

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3)
      -> ON DUPLICATE KEY UPDATE c=3;
mysql> INSERT INTO table (a,b,c) VALUES (4,5,6)
      -> ON DUPLICATE KEY UPDATE c=9;
```

Когда вы используете `ON DUPLICATE KEY UPDATE`, опция `DELAYED` игнорируется.

Вы можете получить значение, использованное для присвоения столбцу с атрибутом `AUTO_INCREMENT`, обратившись к функции `LAST_INSERT_ID()`. В программном интерфейсе `C API` для этого предусмотрена функция `mysql_insert_id()`. Однако, помните, что эти две функции ведут себя не во всех случаях одинаково. Поведение операторов `INSERT` со столбцами `AUTO_INCREMENT` обсуждается далее в разделе 5.8.3.

Если вы применяете оператор `INSERT...VALUES` с множественными списками значений или `INSERT...SELECT`, этот оператор возвращает информационную строку в следующем формате:

```
Records: 100 Duplicates: 0 Warnings: 0
```

`Records` (записи) обозначает количество строк, обработанных оператором. (Это не обязательно будет количество действительно вставленных строк. `Duplicates` может быть ненулевым.) `Duplicates` (дубликаты) означает количество строк, которые не могут быть вставлены из-за того, что они дублируют какие-то уникальные значения индексов. `Warnings` (предупреждения) означает количество попыток вставки значений столбцов, которые оказались по каким-то причинам проблематичными. Предупреждения могут возникать при следующих условиях:

- При попытке вставить `NULL` в столбец, объявленный как `NOT NULL`. Для многострочных операторов `INSERT` или `INSERT...SELECT` таким столбцам присваиваются значения по умолчанию, в соответствии с их типом. Это 0 для числовых типов, пустая строка (') для строковых типов и "нулевые" значения для типов времени и даты.
- При присвоении числовому столбцу значения, лежащего вне пределов допустимого диапазона. Такие значения смещаются к ближайшим границам в рамках допустимого диапазона.
- При присвоении значения вроде '10.34 а' числовым столбцам. Завершающие нецифровые символы отбрасываются, а оставшаяся цифровая часть вставляется.



Если строчное значение не имеет ведущих цифровых символов, то столбцу присваивается 0.

- При вставке символьных значений в строковые столбцы (CHAR, VARCHAR, TEXT или BLOB), которые превышают максимальную длину столбца. Такие значения усекаются до максимальной длины столбца.
- При вставке в столбец даты или времени значения, которое недопустимо для данного типа столбца. В этом случае в столбец записывается соответствующее этому типу нулевое значение.

Если вы используете программный интерфейс C API, информационная строка может быть получена с помощью функции `mysql_info()`.

#### 6.1.4.1. Синтаксис INSERT...SELECT

```
INSERT [LOW_PRIORITY] [IGNORE] [INTO] имя_таблицы [(список_столбцов)]  
SELECT ...
```

С помощью `INSERT ... SELECT` можно быстро вставить множество строк в одну таблицу из другой или других.

Например:

```
INSERT INTO tbl_temp2 (fld_id)  
  SELECT tbl_temp1.fld_order_id  
  FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

В отношении оператора `INSERT...SELECT` соблюдаются следующие условия:

- До версии MySQL 4.0.1 `INSERT...SELECT` неявно работал в режиме `IGNORE`. Начиная с MySQL 4.0.1, следует явно указывать `IGNORE`, чтобы игнорировать строки, которые нарушают условия уникальности ключей.
- Не использовать `DELAYED` с `INSERT...SELECT`.
- До MySQL 4.0.14 целевая таблица оператора `INSERT` не могла встречаться в конструкции `FROM` части `SELECT`. Это ограничение снято в версии 4.0.14.
- Столбцы `AUTO_INCREMENT` работают как обычно.
- Чтобы гарантировать, что бинарный журнал сможет быть использован для пересоздания оригинальных таблиц, MySQL не разрешает параллельные вставки во время выполнения `INSERT...SELECT`.

Вы можете использовать `REPLACE` вместо `INSERT`, чтобы перезаписывать старые строки. `REPLACE` — это дополнение к `INSERT IGNORE` при обработке новых строк, которые содержат значения уникальных ключей, дублирующих старые строки. Новые строки используются для замены старых вместо того, чтобы просто отвергаться.

#### 6.1.4.2. Синтаксис INSERT DELAYED

```
INSERT DELAYED ...
```

Опция `DELAYED` оператора `INSERT` — это расширение MySQL стандарта SQL, которое очень удобно, если у вас есть клиенты, которые не могут ждать завершения выполнения `INSERT`. Это общая проблема, когда MySQL используется для регистрации в журнале с параллельным периодическим выполнением операторов `SELECT` и `UPDATE`, которые работают подолгу. `DELAYED` было представлено в MySQL 3.22.15.

Когда клиент применяет `INSERT DELAYED`, он получает от сервера подтверждение сразу, а вставляемая строка становится в очередь с тем, чтобы реально добавиться в таблицу, когда она не будет занята другими потоками.

Другая существенная выгода использования `INSERT DELAYED` состоит в том, что вставки от многих клиентов связываются вместе и записываются в один блок. Это намного быстрее, чем выполнять множество отдельных вставок.

Существуют некоторые ограничения при использовании `DELAYED`:

- `INSERT DELAYED` работает только с таблицами `MyISAM` и `ISAM`. Для таблиц `MyISAM` если нет свободных блоков в середине файла данных, то поддерживаются параллельные операторы `INSERT` и `SELECT`. В таких условиях вам очень редко понадобится использовать `INSERT DELAYED` с таблицами `MyISAM`.
- `INSERT DELAYED` должен применяться только с операторами `INSERT`, в которых указаны списки значений. Это требование введено в `MySQL 4.0.18`. Сервер игнорирует `DELAYED` для `INSERT DELAYED...SELECT`.
- Сервер игнорирует `DELAYED` в операторах `INSERT DELAYED...ON DUPLICATE UPDATES`.
- Поскольку оператор возвращает управление немедленно, до того как строки фактически будут вставлены, вы не можете использовать `LAST_INSERT_ID()` для получения последнего значения `AUTO_INCREMENT`, которое может быть сгенерировано оператором.
- Строки `DELAYED` невидимы для `SELECT` до тех пор, пока они действительно не будут вставлены в таблицу.

Следует отметить, что поставленные в очередь записи хранятся только в памяти до того, как будут вставлены в таблицу. Это означает, что если вы прервете работу `mysqld` принудительно (например, командой `kill -9`), либо он завершит работу аварийно, то все записи, поставленные в очередь, будут утеряны!

Ниже приведено детальное описание того, что происходит, если применять опцию `DELAYED` к операторам `INSERT` или `REPLACE`. В этом описании под “поток” имеется в виду поток сервера, принимающий оператор `INSERT DELAYED`, а “обработчик” (`handler`) – это поток, который обрабатывает все операторы `INSERT DELAYED` для отдельной таблицы.

- Когда поток выполняет `DELAYED`-оператор на конкретной таблице, создается поток обработчика для выполнения всех этих отложенных операторов для таблицы, если только она еще не существовала на этот момент.
- Поток проверяет, получил ли обработчик `DELAYED`-блокировку. Если нет, он дает команду потоку обработчика сделать это. `DELAYED`-блокировка может быть получена, даже если другие потоки имеют на этой таблице блокировку чтения или записи. Однако обработчик будет ожидать блокировок `ALTER TABLE` или `FLUSH TABLES`, чтобы гарантировать, что структура таблицы синхронизирована.
- Поток выполняет оператор `INSERT`, но вместо записи строки в таблицу помещает ее копию в очередь в памяти, которая управляется потоком обработчика. Любые синтаксические ошибки потоком отмечаются и сообщаются клиентской программе.
- Клиент не может получить от сервера количество дублированных строк или значение `AUTO_INCREMENT` результирующей строки, потому что `INSERT` возвращает управление прежде, чем вставка строки будет выполнена. (Если вы используете

программный интерфейс C API, по той же причине функция `mysql_info()` не вернет ничего вразумительного.)

- Бинарный журнал обновляется потоком обработчика, когда строка вставляется в таблицу. В случае многострочной вставки бинарный журнал обновляется, когда вставляется первая строка.
- После того, как `delayed_insert_limit` строк вставлено, обработчик проверяет, нет ли ожидающих операторов `SELECT`. Если есть, он позволяет им выполниться, прежде чем продолжит свою работу.
- Когда обработчик больше не имеет строк в очереди, блокировка с таблицы снимается. Если больше никаких новых операторов `INSERT DELAYED` не получено в течение `delayed_insert_timeout` секунд, то поток обработчика прерывается.
- Если в конкретной очереди накапливается более `delayed_queue_size` строк, то поток, который запрашивает выполнение очередного оператора `INSERT DELAYED`, ожидает, пока не освободится место в очереди. Это делается для того, чтобы гарантировать, что сервер `mysqld` не захватит всю свободную память под эту очередь.
- Поток обработчика показывается в списке процессов MySQL со значением столбца `Command`, равным `delayed_insert`. Он будет прерван, если выполнить оператор `FLUSH TABLES` или прервать его посредством `KILL идентификатор_потока`. Однако перед выходом поток сначала запишет все строки, находящиеся в очереди, в таблицу. В это время он не будет принимать никаких операторов `INSERT` от других потоков. Если выдать оператор `INSERT DELAYED` после этого, будет создан новый поток обработчика.

Следует отметить, что это означает, что операторы `INSERT DELAYED` имеют более высокий приоритет, чем нормальные операторы `INSERT`, если существует работающий поток обработчика `INSERT DELAYED`. Все остальные операторы обновления данных должны будут ожидать до тех пор, пока очередь `INSERT DELAYED` не очистится, либо кто-нибудь не прервет поток обработчика (с помощью `kill идентификатор_потока`), либо же будет выполнен оператор `FLUSH TABLES`.

- Следующие переменные состояния предоставляют информацию об операторах `INSERT DELAYED`:

Переменная состояния	Значение
<code>Delayed_insert_threads</code>	Количество потоков обработчиков.
<code>Delayed_writes</code>	Количество строк, записанных <code>INSERT DELAYED</code> .
<code>Not_flushed_delayed_rows</code>	Количество строк, ожидающих записи.

Вы можете просмотреть эти переменные, выполнив оператор `SHOW STATUS` либо команду `mysqladmin extend-status`.

Оператор `INSERT DELAYED` медленнее, чем обычный `INSERT`, если таблица не используется. Поддерживать отдельный поток для каждой таблицы, для которой существуют отложенные операторы вставки — это создает дополнительную нагрузку на сервер. Это значит, что вы должны применять `INSERT DELAYED` только в случаях, когда вы действительно уверены, что это необходимо!

### 6.1.5. Синтаксис LOAD DATA INFILE

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'имя_файла.txt'
[REPLACE | IGNORE]
INTO TABLE имя_таблицы
[FIELDS
  [TERMINATED BY '\t']
  [[OPTIONALLY] ENCLOSED BY '"']
  [ESCAPED BY '\\']
]
[LINES
  [STARTING BY '"']
  [TERMINATED BY '\n']
]
[IGNORE количество LINES]
[(имя_столбца, ...)]
```

Оператор `LOAD DATA INFILE` читает строки из текстового файла и загружает их в таблицу на очень высокой скорости.

Вы можете также загружать файлы данных с помощью утилиты `mysqlimport`. Она работает, посылая на сервер оператор `LOAD DATA INFILE`. Опция `--local` заставляет утилиту `mysqlimport` читать файл данных с клиентского хоста. Вы можете указать опцию `--compress` для повышения производительности в медленных сетях, если клиент и сервер поддерживают сжатый протокол.

Если указано ключевое слово `LOW_PRIORITY`, выполнение оператора `LOAD DATA` откладывается до тех пор, пока все остальные клиенты не завершат чтение.

Если указано ключевое слово `CONCURRENT` с таблицей `MyISAM`, которая удовлетворяет условию параллельных вставок (то есть не имеет свободных блоков в середине файла), то другие потоки смогут извлекать данные из таблицы одновременно с выполнением `LOAD DATA`. Применение этой опции немного сказывается на производительности `LOAD DATA`, даже если ни один другой поток с этой таблицей не работает.

Если указано ключевое слово `LOCAL`, оно касается клиентской части соединения.

- Если слово `LOCAL` указано, файл читается клиентской программой на хосте клиента и отправляется на сервер.
- Если слово `LOCAL` не указано, загружаемый файл должен находиться на хосте сервера, и читается сервером непосредственно.

`LOCAL` доступно в MySQL 3.22.6 и более поздних версиях.

Из соображений безопасности при чтении текстовых файлов, расположенных на сервере, файлы должны либо находиться в каталоге данных, либо быть доступными всем по чтению. Кроме того, чтобы использовать `LOAD DATA` с серверными файлами, вы должны иметь привилегию `FILE`.

Загрузка с опцией `LOCAL` идет несколько медленнее, чем когда вы даете серверу возможность непосредственного доступа к загружаемым файлам, потому что в этом случае содержимое файлов передается по сети через клиент-серверное соединение. С другой стороны, в этом случае вам не нужны привилегии `FILE`.

Начиная с версий MySQL 3.23.49 и MySQL 4.0.2 (4.0.13 для Windows), `LOCAL` работает, только если и клиент, и сервер разрешают это. Например, если `mysqld` запущен с опцией `--local-infile=0`, то `LOCAL` работать не будет.

Если вам нужно с помощью `LOAD DATA` читать из программного канала, вы можете применить следующую технику:

```
mkfifo /mysql/db/x/x
chmod 666 /mysql/db/x/x
cat < /dev/tcp/10.1.1.12/4711 > /mysql/db/x/x
mysql -e "LOAD DATA INFILE 'x' INTO TABLE x" x
```

Если вы работаете с версией MySQL, предшествующей 3.23.25, то эту технику можно применять только с `LOAD DATA LOCAL INFILE`.

Если у вас версия MySQL, предшествующая 3.23.24, то вы не сможете читать с помощью оператора `LOAD DATA INFILE` из FIFO. Если вам нужно читать из FIFO (например, из выходного потока `gunzip`), применяйте вместо этого `LOAD DATA LOCAL INFILE`.

При поиске файла в своей файловой системе сервер руководствуется следующими правилами:

- Если задан абсолютный путь, сервер его использует, как есть.
- Если задан относительный путь с одним или более ведущими компонентами, сервер ищет файлы относительно своего каталога данных.
- Если задано имя файла без ведущих компонентов пути, сервер ищет файл в каталоге данных базы данных по умолчанию.

Отметим, что из этих правил следует, что файл с именем `./myfile.txt` читается из каталога данных сервера, в то время как файл с именем `myfile.txt` читается из каталога данных базы данных по умолчанию. Например, следующий оператор `LOAD DATA INFILE` читает файл `data.txt` из каталога данных базы `db1`, потому что `db1` — текущая база данных, несмотря на то, что оператор загружает данные в базу данных `db2`:

```
mysql> USE db1;
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

Ключевые слова `REPLACE` и `IGNORE` управляют работой с входными строками, которые дублируют существующие по значению уникальных ключей.

Если указано `REPLACE`, входные строки заменяют существующие строки (другими словами, строки, которые имеют те же значения первичных или уникальных ключей, как и существующие в таблице строки). См. раздел 6.1.6.

Если указано `IGNORE`, то входные строки, которые дублируют существующие строки с теми же значениями первичных или уникальных ключей, пропускаются. Если не указана ни одна, ни другая опции, то поведение зависит от того, указано ли ключевое слово `LOCAL`. При отсутствии `LOCAL`, в случае обнаружения дублирования ключа генерируется ошибка, а остаток текстового файла игнорируется. При наличии `LOCAL`, поведение по умолчанию будет таким же, как если бы было указано `IGNORE`. Это объясняется тем, что сервер не в состоянии остановить передачу файла в процессе выполнения этой операции.

Если вы хотите игнорировать ограничения внешних ключей в процессе операции загрузки данных, вы можете выполнить оператор `SET FOREIGN_KEY_CHECKS=0` перед запуском `LOAD DATA`.

Если вы запускаете `LOAD DATA` для пустой таблицы MyISAM, все неуникальные индексы создаются в отдельном задании (как для `REPAIR TABLE`). Обычно это приводит к тому, что при наличии многих индексов `LOAD DATA` выполняется гораздо быстрее. Как правило, это работает очень быстро, но в некоторых особых случаях вы можете создать индексы даже еще быстрее, выключив их через `ALTER TABLE...DISABLE KEYS` перед загрузкой

файла в таблицу, пересоздав индексы и включив их с помощью `ALTER TABLE...ENABLE KEYS` после окончания загрузки.

`LOAD DATA INFILE` — это дополнение для `SELECT...INTO OUTFILE`. См. раздел 6.1.7. Для записи данных из таблицы в файл пользуйтесь `SELECT...INTO OUTFILE`. Чтобы прочитать данные обратно из файла в таблицу, воспользуйтесь `LOAD DATA INFILE`. Синтаксис конструкций `FIELDS` и `LINES` одинаков для обоих операторов. Обе эти конструкции не обязательны, но `FIELDS` должна предшествовать `LINES`, если указаны обе.

Если указана конструкция `FIELDS`, то все ее параметры (`TERMINATED BY`, `[OPTIONALLY] ENCLOSED BY` и `ESCAPED BY`) также не обязательны, за исключением требования, что обязательно должен присутствовать хотя бы один параметр.

Если конструкция `FIELDS` не указана, по умолчанию принимается следующий вид:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
```

Если не указана конструкция `LINES`, по умолчанию принимается такой вариант:

```
LINES TERMINATED BY '\n' STARTING BY ''
```

Другими словами, поведение по умолчанию `LOAD DATA INFILE` при чтении ввода таково:

- Искать разделители строк в начале строк.
- Не пропускать никаких префиксов строки.
- Разбивать строку на поля по знакам табуляции.
- Не ожидать, что поля будут заключены в кавычки.
- Интерпретировать появление знака табуляции, перевода строки или символа `'\'`, которым предшествует `'\'`, как литеральные символы, являющиеся частью значения поля.

И наоборот, `SELECT...INTO OUTFILE` по умолчанию ведет себя следующим образом:

- Пишет знаки табуляции между полями.
- Не окружает значения полей кавычками.
- Использует `'\'` для выделения знаков табуляции, перевода строк или `'\'`, встречающихся внутри значений полей.
- Пишет символ перевода строки в конце строк.

Следует отметить, что для написания `FIELDS ESCAPED BY '\\'` потребуется указать два знака обратной косой черты для значений, в которых нужно читать одну обратную косую черту.

### На заметку!

Если вы сгенерировали текстовый файл в системе Windows, возможно, вам понадобится задать `LINES TERMINATED BY '\\r\\n'`, чтобы правильно прочитать файл, поскольку программы Windows обычно используют эти два символа в качестве разделителя строк. Некоторые программы, подобные WordPad, при записи файлов могут использовать символ `'\\r'` в качестве разделителя строк. Чтобы читать такие файлы, используйте `LINES TERMINATED BY '\\r'`.

Если все строки читаемого файла имеют общий префикс, который вы хотите игнорировать, используйте `LINES STARTING BY 'строка_префикса'` для того, чтобы пропускать этот префикс. Если строка не содержит префикса, она пропускается вся целиком.

Опция `IGNORE количество LINES` служит для игнорирования заданного количества строк в начале файла. Например, вы можете воспользоваться `IGNORE 1 LINES`, чтобы пропустить начальную строку, содержащую имена столбцов:

```
mysql> LOAD DATA INFILE '/tmp/test.txt'
-> INTO TABLE test IGNORE 1 LINES;
```

Когда вы применяете `SELECT...INTO OUTFILE` в связке с `LOAD DATA INFILE` для записи данных из базы в файл и последующего его чтения и загрузки обратно в базу, опции управления строками и полями для обоих операторов должны совпадать. В противном случае `LOAD DATA INFILE` не сможет правильно интерпретировать содержимое текстового файла. Предположим, что вы с помощью `SELECT...INTO OUTFILE` вывели данные в текстовый файл, разделяя поля запятыми:

```
mysql> SELECT * INTO OUTFILE 'data.txt'
-> FIELDS TERMINATED BY ','
-> FROM table2;
```

Чтобы прочитать разделенный запятыми файл обратно, правильно будет поступить так:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
-> FIELDS TERMINATED BY ',';
```

Если вместо этого вы попытаетесь прочитать его оператором, приведенным ниже, это не сработает, потому что `LOAD DATA INFILE` будет искать символы табуляции между значениями полей:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
-> FIELDS TERMINATED BY '\t';
```

Наиболее вероятным результатом будет интерпретация входной строки как единственного поля.

`LOAD DATA INFILE` также может использоваться для чтения файлов из внешних источников. Например, некоторый файл может иметь поля, разделенные запятыми и заключенные в двойные кавычки. Если строки в файле разделены символом новой строки, приведенный ниже пример иллюстрирует, какие должны быть установлены опции разделителей строк и столбцов для загрузки файла:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE имя_таблицы
-> FIELDS TERMINATED BY ',' ENCLOSED BY '"'
-> LINES TERMINATED BY '\n';
```

Любым опциям, задающим ограничители строк и столбцов, можно указывать в качестве аргументов пустые строки (''). Если же аргументы – не пустые строки, то значения для `FIELDS [OPTIONALLY] ENCLOSED BY` и `FIELDS ESCAPED BY` должны быть односимвольными. Аргументы опций `FIELDS TERMINATED BY`, `LINES STARTING BY` и `LINES TERMINATED BY` могут иметь длину более одного символа. Например, чтобы писать строки, разделенные символами возврат каретки/перевод строки, либо чтобы читать файлы, содержащие такие строки, указывайте конструкцию `LINES TERMINATED BY '\r\n'`.

Чтобы прочитать файл, разделенный по строкам символами `%%`, можно поступить следующим образом:

```
mysql> CREATE TABLE jokes
-> (a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
-> joke TEXT NOT NULL);
```





- Символа `FIELDS ESCAPED BY`.
- Символа `FIELDS [OPTIONALLY] ENCLOSED BY`.
- Первого символа последовательностей `FIELDS TERMINATED BY` и `LINES TERMINATED BY`.
- ASCII 0 (который пишется вслед за символом отмены как ASCII '0', а не нулевой байт).

Если символ `FIELDS ESCAPED BY` пуст, никакие символы не предваряются символами отмены, и `NULL` выводится как `NULL`, а не `\N`. Вероятно, это не очень хорошая мысль — оставлять пустым аргумент `FIELDS ESCAPED BY`, особенно, если значения полей ваших данных содержат любой из упомянутых символов.

При вводе, если `FIELDS ESCAPED BY` не пуст, то при появлении этого символа в строке значения он удаляется, а следующий за ним символ читается литерально, как часть значения поля. Исключениями являются последовательности '0' или 'N' (\0 или \N, если символом отмены выбран '\'). Эти последовательности интерпретируются, соответственно, как ASCII NUL (нулевой байт) и `NULL`. Правила обращения с `NULL` описаны ниже в настоящем разделе.

Более подробную информацию о синтаксисе отмены '\ ' можно найти в разделе 2.1.

В некоторых случаях опции, управляющие полями и строками, взаимодействуют между собой:

- Если указана пустая строка для `LINES TERMINATED BY`, а `FIELDS TERMINATED BY` не пуст, то разделителем строк также служит `LINES TERMINATED BY`.
- Если пусты и `FIELDS TERMINATED BY` и `FIELDS ENCLOSED BY`, используется формат фиксированной строки (без разделителей). В этом формате не применяется никаких разделителей между полями (но можно иметь разделитель строк). Вместо этого значения столбцов пишутся и читаются с использованием ширины отображения столбцов. Например, если столбец объявлен как `INT(7)`, значения столбца записываются в семисимвольное поле. При вводе значения столбца извлекаются чтением семи символов.

`LINES TERMINATED BY` по-прежнему используется для разделения строк. Если строка не содержит всех полей, остальным столбцам присваиваются их значения по умолчанию. Если у вас нет терминатора строки, его значение нужно установить в ' '. В этом случае текстовый файл должен содержать все поля в каждой строке. Формат с фиксированной длиной строки также касается работы со значениями `NULL`, как описано ниже. Следует отметить, что формат фиксированной длины не работает, если используется многобайтный набор символов (например, Unicode).

Обработка значений `NULL` варьируется в зависимости от применяемых опций `FIELDS` и `LINES`:

- При значениях `FIELDS` и `LINES` по умолчанию `NULL` пишется как значение поля в виде `\N` для вывода и это же значение `\N` читается как `NULL` при вводе (предполагая, что символ `ESCAPED BY` установлен в '\').
- Если `FIELDS ENCLOSED BY` не пустой, то поле, содержащее литеральное слово `NULL`, читается как значение `NULL`. Это отличается от случая, когда слово `NULL` ограничено символами `FIELDS ENCLOSED BY`, когда значение читается, как строка 'NULL'.
- Если `FIELDS ESCAPED BY` пустое, `NULL` пишется как слово `NULL`.

- При формате с фиксированной длиной строки (что случается, когда и `FIELDS TERMINATED BY`, и `FIELDS ENCLOSED BY` пустые) `NULL` записывается как пустая строка. Отметим, что это приводит к тому, что значения `NULL` и пустые строки в таблице становятся неразличимы при записи в файл, поскольку в обоих случаях пишутся пустые строки. Если вам необходимо делать различие между ними, избегайте использования формата с фиксированной длиной строки.

Ниже представлены некоторые случаи, не поддерживаемые `LOAD DATA INFILE`:

- Строки фиксированной длины (`FIELDS TERMINATED BY` и `FIELDS ENCLOSED BY` пустые) при наличии столбцов типа `TEXT` или `BLOB`.
- Если вы указываете разделитель, который совпадает с префиксом другого, `LOAD DATA INFILE` не может правильно интерпретировать входной поток. Например, следующий вариант приведет к проблемам:  
`FIELDS TERMINATED BY ''' ENCLOSED BY '''`
- Если `FIELDS ESCAPED BY` пуст, значения полей, которые включают в себя символы `FIELDS ENCLOSED BY` или `LINES TERMINATED BY` с последующим символом `LINES TERMINATED BY`, заставят `LOAD DATA INFILE` слишком рано прекратить чтение файла или строки. Это произойдет потому, что `LOAD DATA INFILE` не может правильно определить, где завершается значение поля или строки.

Следующий пример загружает все столбцы таблицы `persondata`:

```
mysql> LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

По умолчанию, если в конце оператора `LOAD DATA INFILE` не приведен список столбцов, ожидается, что во входящей строке содержатся поля для каждого столбца таблицы. Если вы хотите загрузить только некоторые из столбцов таблицы, указывайте список столбцов:

```
mysql> LOAD DATA INFILE 'persondata.txt'  
-> INTO TABLE persondata (col1,col2,...);
```

Вы также должны указывать список столбцов, если порядок полей во входном файле отличается от порядка столбцов в таблице. В противном случае MySQL не сможет установить соответствие между входными полями и столбцами таблиц.

Если входной файл имеет слишком мало полей в строках, то недостающим столбцам будут присвоены значения по умолчанию. Присвоение значений по умолчанию описано в разделе 6.2.5.

Пустые значения полей интерпретируются иначе, чем пропущенные:

- Для строковых типов – столбцу присваивается пустая строка.
- Для числовых типов – столбцу присваивается 0.
- Для типов даты и времени – столбец устанавливается в соответствующее типу “нулевое” значение. См. раздел 4.3.

Это те же значения, что получаются в результате явного присвоения пустой строки столбцам этих типов в операторах `INSERT` или `UPDATE`.

Значения столбцов типа `TIMESTAMP` устанавливаются в текущую дату и время, только если им присваивается значение `NULL` (то есть, `\N`), либо если столбец этого типа пропущен в списке полей, если список полей приведен.

LOAD DATA INFILE рассматривает весь ввод, как строковый, поэтому вы не можете использовать числовые значения для столбцов типа ENUM или SET, как это допускается в операторах INSERT. Все значения ENUM или SET должны указываться как строки!

Когда оператор LOAD DATA INFILE завершает работу, он возвращает информационную строку в следующем формате:

```
Records: 1 Deleted: 0 Skipped: 0 Warnings: 0
```

Если вы работаете с программным интерфейсом C API, то можете получить информацию об этом операторе, обратившись к функции `mysql_info()`.

Предупреждения, которые появляются при некоторых условиях, такие же, как при вставке значений оператором INSERT (см. раздел 6.1.4), за исключением того, что LOAD DATA INFILE кроме них генерирует предупреждения о том, что во входном файле слишком мало или слишком много полей. Предупреждения нигде не сохраняются, количество предупреждений может быть использовано только как признак того, что все прошло успешно.

Начиная с MySQL 4.1.1, вы можете использовать SHOW WARNINGS для получения списка первых `max_error_count` предупреждений в качестве информации о том, что при загрузке прошло не так, как надо. См. раздел 6.5.3.20.

До MySQL 4.1.1 только количество предупреждений было признаком того, что загрузка прошла не гладко. Если вы получаете предупреждение и хотите знать точно, почему оно появились, единственный путь сделать это — с помощью SELECT...INTO OUTFILE выгрузить дамп таблицы в другой файл и сравнить его с оригинальным входным файлом.

## 6.1.6. Синтаксис REPLACE

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] имя_таблицы [(имя_столбца,...)]
VALUES ({выражение | DEFAULT},...), (...),...
```

или:

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] имя_таблицы
SET имя_столбца={выражение | DEFAULT}, ...
```

или:

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] имя_таблицы [(имя_столбца,...)]
SELECT ...
```

REPLACE работает точно так же, как INSERT, за исключением того, что если строка с тем же значением первичного или уникального ключа в таблице существует, то старая строка удаляется перед вставкой новой. См. раздел 6.1.4.

Следует отметить, что использование REPLACE не имеет смысла, если только таблица не имеет индексов PRIMARY KEY или UNIQUE. В этом случае оператор полностью эквивалентен INSERT. Вы можете обращаться к значениям в старой строке и использовать их в новой. Может показаться, что это можно было делать и в некоторых старых версиях MySQL, однако там содержалась ошибка, которая позже была исправлена.

Для использования REPLACE необходимо иметь привилегии INSERT и DELETE для таблицы.

Оператор REPLACE возвращает количество строк, которые будут обработаны. Это количество равно сумме удаленных и вставленных строк. Если количество больше 1 для однострочного оператора REPLACE, это значит, что строка была вставлена и ни одной строки не было удалено перед вставкой. Существует возможность, что одна новая строка заменит более одной старой, если у таблицы есть несколько уникальных индексов и новая строка по значению индексных ключей разных уникальных индексов дублирует более чем одну старую строку.

Счетчик обработанных строк позволяет легко определить, выполнил ли оператор REPLACE только добавление новой строки, или производил также замену. При значении 1 выполнялась только вставка, в противном случае – замена.

Если вы пользуетесь программным интерфейсом C API, счетчик обработанных строк можно получить вызовом функции `mysql_affected_rows()`.

Ниже приведено более детальное описание применяемого алгоритма (это касается также и `LOAD DATA ... REPLACE`):

1. Выполняется попытка вставки новой строки в таблицу.
2. Если вставка не удастся из-за дублирования первичного или уникальных ключей, то:
  - а) конфликтующие строки удаляются из таблицы;
  - б) выполняется новая попытка вставки строки.

## 6.1.7. Синтаксис SELECT

```
SELECT
  [ALL | DISTINCT | DISTINCTROW]
  [HIGH_PRIORITY]
  [STRAIGHT_JOIN]
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
  [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS] выражение_select,...
  [INTO OUTFILE 'имя_файла' опции_экспорта
   | INTO DUMPFILE 'имя_файла']
  [FROM табличные_ссылки
   {WHERE определение_where}
   {GROUP BY {имя_столбца | выражение | позиция}
    [ASC | DESC], ... [WITH ROLLUP]}
   {HAVING определение_where}
   {ORDER BY {имя_столбца | выражение | позиция}
    [ASC | DESC], ...}
   {LIMIT [смещение, {количество_строк | количество_строк OFFSET смещение}]
   {PROCEDURE имя_процедуры(список_аргументов)}
   {FOR UPDATE | LOCK IN SHARE MODE}]
```

SELECT применяется для извлечения строк из одной или более таблиц. Поддержка операторов UNION и подзапросов доступна, начиная с версий MySQL 4.0 и 4.1, соответственно. См. разделы 6.1.7.2 и 6.1.8.

- Каждое выражение *выражение\_select* указывает столбец, который необходимо извлечь.
- *табличные\_ссылки* указывают таблицу или таблицы, из которых извлекаются строки. Соответствующий синтаксис описан в разделе 6.1.8.

- определение `where` задает любые условия, которым должны удовлетворять выбранные строки.

`SELECT` также может использоваться для извлечения вычисляемых строк без обращения к какой-либо таблице.

Например:

```
mysql> SELECT 1 + 1;
-> 2
```

Все используемые в операторе конструкции даются точно в том порядке, как приведено в описании синтаксиса. Например, конструкция `HAVING` должна следовать за конструкцией `GROUP BY` и перед `ORDER BY`.

- Выражению `выражение_select` может быть дан псевдоним `AS имя_псевдонима`. Псевдоним используется как имя столбца, заданного выражением, и может присутствовать в конструкциях `GROUP BY`, `ORDER BY` и `HAVING`, например:

```
mysql> SELECT CONCAT(last_name, ', ', first_name) AS full_name
-> FROM mytable ORDER BY full_name;
```

При присвоении псевдонима столбцу указывать ключевое слово `AS` не обязательно. Предыдущий пример может быть переписан следующим образом:

```
mysql> SELECT CONCAT(last_name, ', ', first_name) full_name
-> FROM mytable ORDER BY full_name;
```

Поскольку `AS` необязательно, здесь может возникнуть одна тонкая проблема, если вы забудете поставить запятую между двумя выражениями `SELECT`: MySQL интерпретирует второе как псевдоним. Например, в следующем операторе `columnb` рассматривается как псевдоним:

```
mysql> SELECT columna columnb FROM mytable;
```

- Не допускается указание псевдонима столбца в конструкции `WHERE`, поскольку значение столбца еще может быть неопределенным, когда выполняется конструкция `WHERE`. См. раздел A.1.4.
- Конструкция `FROM табличные_ссылки` перечисляет таблицы, из которых извлекаются строки. Если вы указываете более чем одну таблицу, выполняется соединение. Информацию о синтаксисе соединений можно найти в разделе 6.1.7.1. Для каждой из перечисленных таблиц можно указать необязательный псевдоним.

```
имя_таблицы [[AS] псевдоним]
[[USE INDEX (список_ключей)]
 | [IGNORE INDEX (список_ключей)]
 | [FORCE INDEX (список_ключей)]]
```

Применение `USE INDEX`, `IGNORE INDEX`, `FORCE INDEX` для указания подсказок оптимизатору по использованию индексов описано в разделе 6.1.7.1.

Начиная с MySQL 4.0.14, можно использовать `SET max_seeks_for_key=значение`, как альтернативный способ заставить MySQL предпочесть поиск по ключу вместо сканирования таблицы.

- К таблице внутри текущей базы данных можно обращаться как `имя_таблицы`, либо как `имя_базы_данных.имя_таблицы` для явного указания базы данных. Вы можете сослаться на столбец как `имя_столбца`, `имя_таблицы.имя_столбца` или

*имя\_базы\_данных.имя\_таблицы.имя\_столбца*. Префиксы столбцов *имя\_таблицы* или *имя\_базы\_данных.имя\_таблицы* указывать необязательно, если только ссылка на столбец не является неоднозначной. В разделе 2.2 представлены примеры неоднозначных ссылок, которые требуют более явных спецификаций.

- Начиная с MySQL 4.1.0, допускается указывать DUAL как имя псевдотаблицы в ситуациях, когда не нужны ссылки на реальные таблицы:

```
mysql> SELECT 1 + 1 FROM DUAL;
-> 2
```

DUAL введено исключительно совместимости ради. Некоторые другие серверы баз данных требуют такого синтаксиса.

- Ссылка на таблицу может быть заменена псевдонимом: *имя\_таблицы* [AS] *имя\_псевдонима*.

```
mysql> SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
-> WHERE t1.name = t2.name;
mysql> SELECT t1.name, t2.salary FROM employee t1, info t2
-> WHERE t1.name = t2.name;
```

- В конструкции WHERE можно использовать любые функции, поддерживаемые MySQL, за исключением агрегатных (итоговых) функций. См. главу 5.
- На столбцы, выбранные для вывода, можно сослаться в конструкциях ORDER BY и GROUP BY, используя при этом имена столбцов, псевдонимы столбцов либо их номера позиций. Позиции столбцов указываются целыми числами, начиная с 1:

```
mysql> SELECT college, region, seed FROM tournament
-> ORDER BY region, seed;
mysql> SELECT college, region AS r, seed AS s FROM tournament
-> ORDER BY r, s;
mysql> SELECT college, region, seed FROM tournament
-> ORDER BY 2, 3;
```

Чтобы сортировать в обратном порядке, в конструкции ORDER BY потребуется добавить ключевое слово DESC к имени столбца, по которому выполняется сортировка. По умолчанию применяется порядок сортировки по возрастанию. Это можно указать явно ключевым словом ASC.

Применение номеров позиций столбцов считается устаревшим, поскольку этот синтаксис исключен из стандарта SQL.

- Если используется GROUP BY, выходные строки сортируются в соответствии со столбцами, указанными в GROUP BY, как если бы было указано ORDER BY для тех же столбцов. MySQL имеет расширенный вариант конструкции GROUP BY, начиная с версии сервера 3.23.34, позволяющий специфицировать ASC и DESC после столбцов, названных в конструкции:

```
SELECT a, COUNT(b) FROM test_table GROUP BY a DESC
```

- MySQL расширяет применение GROUP BY, позволяя выбирать поля, не перечисленные в конструкции GROUP BY. Если вы не получаете ожидаемого результата на запрос, прочтите описание использования GROUP BY в разделе 5.9.

- Начиная с MySQL 4.1.1, GROUP BY допускает модификатор WITH ROLLUP. См. раздел 5.9.
- Конструкция HAVING может ссылаться на любой столбец или псевдоним из выражение\_select. Оно указывается почти в конце, непосредственно перед отправкой результата клиенту, без оптимизации (только LIMIT находится после HAVING).
- Не используйте HAVING для элементов, которые должны быть в конструкции WHERE. Например, не делайте так:

```
mysql> SELECT имя_столбца FROM имя_таблицы HAVING имя_столбца > 0;
```

Вместо этого лучше написать:

```
mysql> SELECT имя_столбца FROM имя_таблицы WHERE имя_столбца > 0;
```

- Конструкция HAVING может ссылаться на агрегатные функции, в то время как WHERE — нет:

```
mysql> SELECT user, MAX(salary) FROM users
-> GROUP BY user HAVING MAX(salary)>10;
```

Однако это не работает в старых версиях MySQL (до 3.22.5). Вместо этого вы можете использовать псевдоним в списке выбираемых столбцов, чтобы сослаться на него в конструкции HAVING:

```
mysql> SELECT user, MAX(salary) AS max_salary FROM users
-> GROUP BY user HAVING max_salary>10;
```

- Конструкция LIMIT может использоваться для ограничения количества строк, возвращаемых оператором SELECT. LIMIT принимает один или два числовых аргумента, которые должны быть целочисленными константами. Когда указываются два аргумента, первый означает смещение в результирующем списке первой строки, которую нужно вернуть, а второй — максимальное количество возвращаемых строк. Смещение начальной строки равно 0 (не 1):

```
mysql> SELECT * FROM table LIMIT 5,10; # Извлечение строк 6-15
```

Для совместимости с PostgreSQL система MySQL также поддерживает синтаксис LIMIT количество\_строк OFFSET смещение.

Чтобы извлечь все строки, начиная с определенного смещения и до конца результирующего набора, можно использовать какое-нибудь большое число во втором параметре. Этот оператор извлекает все строки, начиная с 96-й и до последней:

```
mysql> SELECT * FROM table LIMIT 95,18446744073709551615;
```

Если указан один аргумент, то он обозначает количество строк, которые нужно вернуть от начала результирующего набора:

```
mysql> SELECT * FROM table LIMIT 5; # Извлечь первые 5 строк
```

Другими словами, LIMIT n эквивалентно LIMIT 0, n.

- Синтаксис SELECT...INTO outfile 'имя\_файла' пишет извлекаемые строки в файл. Файл создается на хосте сервера, поэтому вы должны иметь привилегию FILE, чтобы использовать эту форму SELECT. Файл не должен существовать на момент выполнения оператора, что предотвращает случайное разрушение важных системных файлов, вроде /etc/passwd или файлов таблиц.

Оператор `SELECT...INTO OUTFILE` предназначен главным образом для того, чтобы позволять быстро выгружать дампы таблицы на машине сервера. Если вы хотите создать результирующий файл на хосте клиента, вы не можете применить для этого `SELECT...INTO OUTFILE`. В этом случае вы должны вместо этого использовать на клиентской машине что-то вроде `mysql -e "SELECT..." > имя_файла` для генерации файла.

`SELECT...INTO OUTFILE` — это дополнение `LOAD DATA INFILE`. Синтаксис части опции экспорта этого оператора состоит из тех же предложений `FIELDS` и `LINES`, которые применяются в `LOAD DATA INFILE`. См. раздел 6.1.5.

`FIELDS ESCAPED BY` управляет записью специальных символов. Если аргумент `FIELDS ESCAPED BY` не пуст, он используется в качестве префикса при выводе следующих символов:

- Самого символа `FIELDS ESCAPED BY`.
- Символа `FIELDS [OPTIONALLY] ENCLOSED BY`.
- Первого символа значений `FIELDS TERMINATED BY` и `LINES TERMINATED BY`.
- ASCII 0 (который пишется вслед за символом отмены как ASCII '0', а не нулевой байт).

Если `FIELDS ESCAPED BY` пуст, никакие символы не предваряются символами отмены, и `NULL` выводится как `NULL`, а не `\N`. Вероятно, это не очень хорошая идея — указывать пустой символ отмены, особенно, если значения полей в ваших данных содержат любые из перечисленных выше символов.

Причина этого состоит в том, что вы обязаны предварять символами отмены любые символы из `FIELDS TERMINATED BY`, `ENCLOSED BY`, `ESCAPED BY` и `LINES TERMINATED BY`, чтобы гарантированно иметь возможность прочесть файл в будущем. ASCII NUL предваряется символом отмены для того, чтобы упростить просмотр файла некоторыми программами постраничного вывода.

Результирующий файл не должен соответствовать SQL-синтаксису, поэтому ничего другого более не должно предваряться символами отмены.

Ниже приведен пример, который генерирует файл в формате с запятой в качестве разделителя полей, который используется многими программами:

```
SELECT a,b,a+b INTO OUTFILE '/tmp/result.text'
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM test_table;
```

- Если вы используете `INTO DUMPFILE` вместо `INTO OUTFILE`, MySQL пишет только одну строку в файл, без каких-либо ограничителей строк или столбцов, без какой бы то ни было обработки с помощью символов отмены. Это удобно, если нужно поместить в файл значения типа `BLOB`.
- Следует отметить, что любой файл, созданный `INTO DUMPFILE` или `INTO OUTFILE`, доступен по записи всем пользователям хоста сервера. Причина этого в том, что сервер MySQL не может создать файл, принадлежащий кому-нибудь другому, а не пользователю, от имени которого он запущен (никогда не запускайте `mysqld` от имени `root`). Поэтому файл должен быть доступным всем по записи, чтобы вы могли манипулировать его содержимым.



- Конструкция `PROCEDURE` определяет процедуру, которая должна обрабатывать данные в результирующем наборе.
- Если вы применяете `FOR UPDATE` с механизмом хранения, который использует блокировки страниц или строк, то строки, проверяемые запросом, блокируются по записи до завершения текущей транзакции.

После ключевого слова `SELECT` можно указать множество опций, влияющих на выполнение оператора.

Опции `ALL`, `DISTINCT` и `DISTINCTROW` определяют, должны ли возвращаться дублированные строки. Если ни одна из этих опций не указана, по умолчанию принимается `ALL` (возвращаются все соответствующие строки). `DISTINCT` и `DISTINCTROW` – синонимы; они указывают, что дублированные строки в результирующем наборе исключаются.

`HIGH_PRIORITY`, `STRAIGHT_JOIN` и опции, начинающиеся с `SQL_`, являются расширениями MySQL стандарта SQL.

- `HIGH_PRIORITY` назначает оператору `SELECT` более высокий приоритет, чем операторам обновления таблицы. Вы должны использовать это только для запросов, которые выполняются однократно и очень быстро. Запрос `SELECT HIGH_PRIORITY`, отправленный, когда таблица заблокирована по чтению, будет выполняться, даже если есть ожидающий освобождения таблицы оператор обновления данных. `HIGH_PRIORITY` не может использоваться с оператором `SELECT`, являющимся частью `UNION`.
- `STRAIGHT_JOIN` заставляет оптимизатор объединять таблицы в том порядке, в котором они перечислены в конструкции `FROM`. Это можно использовать для ускорения запросов, когда оптимизатор объединяет таблицы не в оптимальном порядке. `STRAIGHT_JOIN` также может применяться в списке *табличные\_ссылки*. См. раздел 6.1.7.1.
- `SQL_BIG_RESULT` может применяться с `GROUP BY` или `DISTINCT`, чтобы сообщить оптимизатору, что результирующий набор будет иметь много строк. В этом случае MySQL будет непосредственно использовать при необходимости временные таблицы на диске. А также в этом случае MySQL предпочтет сортировку по ключу элементов `GROUP BY` с использованием временных таблиц.
- `SQL_BUFFER_RESULT` принудительно помещает результат во временный файл. Это помогает MySQL пораньше освободить табличные блокировки и оказывается полезным в случаях, когда на отправку результирующего набора клиенту тратится много времени.
- `SQL_SMALL_RESULT` может применяться вместе с `GROUP BY` или `DISTINCT`, чтобы сообщить оптимизатору, что результирующий набор будет маленьким. В этом случае MySQL использует быстрые временные таблицы, чтобы хранить результирующую таблицу вместо применения сортировки. В MySQL 3.23 и выше это обычно не требуется.
- `SQL_CALC_FOUND_ROWS` (доступно в MySQL 4.0.0 и выше) сообщает серверу MySQL, что нужно посчитать количество строк в результирующем наборе, независимо от конструкции `LIMIT`. Количество строк затем может быть извлечено с помощью `SELECT FOUND_ROWS()`. См. раздел 5.8.3.

До MySQL 4.1.0 эта опция не работала с LIMIT 0, что было оптимизировано для немедленного возврата (со значением счетчика строк, равным 0).

- SQL\_CACHE заставляет MySQL сохранять результат в кэше запросов, если используется значение query\_cache\_type, равное 2, или DEMAND. Для запросов, использующих UNION или подзапросы, эта опция влияет на все операторы SELECT в запросе.
- SQL\_NO\_CACHE сообщает MySQL, что результаты запросов не нужно сохранять в кэше. Для запросов, которые используют UNION или подзапросы, эта опция влияет на все операторы SELECT в запросе.

### 6.1.7.1. Синтаксис JOIN

MySQL поддерживает следующие варианты синтаксиса JOIN в части табличные\_ссылки операторов SELECT, а также многотабличных операторов DELETE и UPDATE:

```
табличная_ссылка, табличная_ссылка
табличная_ссылка [INNER | CROSS] табличная_ссылка [условие_соединения]
табличная_ссылка STRAIGHT JOIN табличная_ссылка
табличная_ссылка LEFT [OUTER] JOIN табличная_ссылка [условие_соединения]
табличная_ссылка NATURAL [LEFT [OUTER]] JOIN табличная_ссылка
{ OJ табличная_ссылка LEFT OUTER JOIN табличная_ссылка ON условное_выражение }
табличная_ссылка RIGHT [OUTER] JOIN табличная_ссылка [условие_соединения]
табличная_ссылка NATURAL [RIGHT [OUTER]] JOIN табличная_ссылка
```

табличная\_ссылка определено как:

```
имя_таблицы {[AS] псевдоним}
    {[USE INDEX (список_ключей)}
    | [IGNORE INDEX (список_ключей)}
    | [FORCE INDEX (список_ключей)]}
```

условие\_соединения определено как:

```
ON условное_выражение | USING (список_столбцов)
```

Обычно вы не должны иметь в части ON никаких условий, ограничивающих строки для результирующего набора. Эти условия указываются в конструкции WHERE.

Однако существуют исключения из этого правила.

Отметим, что синтаксис INNER JOIN предусматривает условие\_соединения только начиная с версии MySQL 3.23.17 и выше. То же самое верно и для JOIN и CROSS JOIN, но только начиная с версии MySQL 4.0.11.

Синтаксис {OJ...LEFT OUTER JOIN...}, представленный выше, введен только для совместимости с ODBC.

- Ссылки на таблицы могут быть заменены псевдонимами с использованием имя\_таблицы AS имя\_псевдонима или имя\_таблицы имя\_псевдонима:

```
mysql> SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
-> WHERE t1.name = t2.name;
mysql> SELECT t1.name, t2.salary FROM employee t1, info t2
-> WHERE t1.name = t2.name;
```

- Условия ON – это любые условные выражения в той же форме, что применяются в конструкции WHERE.

- Если не найдено соответствующих записей в правой таблице части ON или USING конструкции LEFT JOIN, для правой таблицы используется строка со всеми столбцами, установленными в NULL. Этим фактом можно воспользоваться для поиска записей в таблице, для которой не существует дополнений в другой таблице:

```
mysql> SELECT table1.* FROM table1
-> LEFT JOIN table2 ON table1.id=table2.id
-> WHERE table2.id IS NULL;
```

Этот пример находит все строки таблицы table1 со значениями id, которых нет в таблице table2 (то есть, всех строк таблицы table1, для которых нет связанных строк в table2). Предполагается, что table2.id объявлен как NOT NULL.

- Конструкция USING(список\_столбцов) перечисляет список столбцов, которые должны присутствовать в обеих таблицах. Следующие две конструкции семантически 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
```

- NATURAL [LEFT] JOIN для двух таблиц определено как семантический эквивалент INNER JOIN или LEFT JOIN с конструкцией USING, которая перечисляет все столбцы, присутствующие в обеих таблицах.
- INNER JOIN и , (запятая) семантически эквивалентны при отсутствии условия соединения: оба выполняют декартово произведение указанных таблиц (то есть каждая строка первой таблицы соединяется со всеми строками второй таблицы).
- RIGHT JOIN работает аналогично LEFT JOIN. Чтобы сохранять код переносимым между системами управления базами данных, рекомендуется использовать LEFT JOIN вместо RIGHT JOIN.
- STRAIGHT\_JOIN идентично JOIN за исключением того, что левая таблица всегда читается раньше, чем правая. Это можно использовать в тех (немногих) случаях, когда оптимизатор располагает таблицы в неправильном порядке.

Начиная с MySQL 3.23.12, вы можете указывать подсказки (hints) о том, как MySQL должен использовать индексы при извлечении данных из таблицы. Указав USE INDEX (список\_ключей), вы можете принудить MySQL использовать только один из всех индексов, чтобы искать строки в таблице. Альтернативный синтаксис IGNORE INDEX (список\_ключей) может применяться, чтобы запретить MySQL использовать какой-то отдельный индекс.

Эти подсказки удобны, если EXPLAIN показывает, что MySQL использует неправильные индексы из списка возможных.

Начиная с MySQL 4.0.9, вы также можете использовать FORCE INDEX. Это работает подобно USE INDEX (список\_ключей), но с тем отличием, что сканирование таблиц рассценивается, как очень дорогая операция. Другими словами, сканирование таблицы допускается только в том случае, если нет способа использования индекса для поиска строк в таблице.

USE KEY, IGNORE KEY и FORCE KEY – синонимы для USE INDEX, IGNORE INDEX и FORCE INDEX.

**На заметку!**

USE INDEX, IGNORE INDEX и FORCE INDEX влияют только на то, какие индексы будут использоваться, когда MySQL принимает решение о том, как искать строки в таблице и как выполнять соединение. Они не влияют на то, как будет использоваться индекс при вычислении ORDER BY или GROUP BY.

Ниже представлены примеры соединений:

```
mysql> SELECT * FROM table1,table2 WHERE table1.id=table2.id;
mysql> SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id;
mysql> SELECT * FROM table1 LEFT JOIN table2 USING (id);
mysql> SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id
->      LEFT JOIN table3 ON table2.id=table3.id;
mysql> SELECT * FROM table1 USE INDEX (key1,key2)
->      WHERE key1=1 AND key2=2 AND key3=3;
mysql> SELECT * FROM table1 IGNORE INDEX (key3)
->      WHERE key1=1 AND key2=2 AND key3=3;
```

**6.1.7.2. Синтаксис UNION**

```
SELECT ...
UNION [ALL | DISTINCT]
SELECT ...
[UNION [ALL | DISTINCT]
SELECT ...]
```

UNION применяется для комбинирования результатов множества операторов SELECT в один результирующий набор. Оператор UNION доступен, начиная с MySQL 4.0.0.

Выбранные столбцы, перечисленные в соответствующих позициях каждого оператора SELECT, должны иметь одинаковый тип (например, первый столбец, выбираемый первым оператором, должен иметь тот же тип, что и первый столбец, выбранный другими операторами). Имена столбцов первого оператора SELECT используются как имена столбцов для всех возвращаемых результатов.

Операторы SELECT — это обычные операторы извлечения строк, но со следующими ограничениями:

- Только один из операторов SELECT может иметь INTO outfile.
- HIGH\_PRIORITY не может использоваться с операторами SELECT, которые входят в UNION. Если вы указываете его в первом операторе SELECT, это не даст никакого эффекта. Если указать его в последующих операторах SELECT, результатом будет сообщение о синтаксической ошибке.

Если вы не используете ключевое слово ALL в UNION, все возвращенные строки будут уникальными, как если бы был указан DISTINCT для общего результирующего набора. Если ALL будет указан, вы получите все соответствующие строки от всех входящих в UNION операторов SELECT.

Ключевое слово DISTINCT является необязательным (оно введено в версии MySQL 4.0.17). Оно не делает ничего, но добавлено для обеспечения соответствия синтаксиса стандарту SQL.

**На заметку!**

Нельзя смешивать UNION ALL и UNION DISTINCT в одном запросе. Если указано ALL для одного UNION, оно применяется ко всем.

Если вы хотите применить ORDER BY для сортировки результата UNION, необходимо использовать скобки:

```
(SELECT a FROM имя_таблицы WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
UNION
(SELECT a FROM имя_таблицы WHERE a=11 AND B=2 ORDER BY a LIMIT 10)
ORDER BY a;
```

Типы и длина столбцов в результирующем наборе UNION принимают во внимание значения, извлеченные всеми операторами SELECT. До MySQL 4.1.1 существовало ограничение, заключающееся в том, что значения первого SELECT использовались для определения типов и длин результирующих столбцов. Это могло приводить к усечению значений, если, например, первый SELECT извлекал значения, более короткие, чем второй:

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+-----+
| REPEAT('a',1) |
+-----+
| a              |
| b              |
+-----+
```

Это ограничение было снято в MySQL 4.1.1:

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+-----+
| REPEAT('a',1) |
+-----+
| a              |
| bbbbbbbbbb    |
+-----+
```

### 6.1.8. Синтаксис подзапросов

Подзапрос – это оператор SELECT внутри другого оператора.

Начиная с MySQL 4.1, поддерживаются все формы подзапросов, которых требует стандарт SQL, равно как и некоторые средства, специфичные для MySQL.

В ранних версиях MySQL приходилось искать избежать подзапросов вообще либо искать обходные пути, но разработчики, которые начинают писать новый код сейчас, обнаружат, что подзапросы – очень удобная часть инструментального набора MySQL.

В MySQL до 4.1 большинство подзапросов могут быть успешно реализовано в виде соединений или другими способами. См. раздел 6.1.8.11.

Вот пример подзапроса:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

В этом примере SELECT \* FROM t1 является *внешним запросом* (или *внешним оператором*), а (SELECT column1 FROM t2) – *подзапросом*. Говорят, что подзапрос вложен во внешний запрос. Фактически, можно вкладывать подзапросы в подзапросы на большую глубину. Подзапрос всегда должен появляться в скобках.

Основные преимущества подзапросов:

- Они позволяют писать *структурированные* запросы таким образом, что можно изолировать части оператора.

- Они представляют альтернативный способ выполнения операций, которые требуют применения сложных соединений и слияний (JOIN и UNION).
- По мнению многих, они более читабельны. Действительно, новизна подзапросов в том, что они наглядно представляют людям исходную идею SQL как структурированного языка запросов.

Ниже приведен пример оператора, который демонстрирует основные понятия о синтаксисе подзапросов, поддерживаемых MySQL, как того требует стандарт SQL:

```
DELETE FROM t1
WHERE s11 > ANY
  (SELECT COUNT(*) /* без подсказок */ FROM t2
   WHERE NOT EXISTS
    (SELECT * FROM t3
     WHERE ROW(5*t2.s1,77)=
      (SELECT 50,11*s1 FROM t4 UNION SELECT 50,77 FROM
       (SELECT * FROM t5) AS t5)));
```

### 6.1.8.1. Подзапрос, как скалярный операнд

В своей простейшей форме (*скалярный* подзапрос представляет собой противоположность *строчным* или *табличным* подзапросам, описанным ниже), подзапрос – это простой операнд. То есть вы можете использовать его в любом месте, где допустимо значение столбца или литерал, и вы можете ожидать, что он имеет те же характеристики, которые имеют все операнды: тип данных, длину, признак того, может ли он принимать значение NULL, и так далее. Например:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);
SELECT (SELECT s2 FROM t1);
```

Подзапрос в этом запросе имеет тип данных CHAR, длину 5, набор символов и порядок сопоставления по умолчанию такие, как были при выполнении CREATE TABLE, и признак того, что значение столбца допускает значение NULL. Фактически почти все подзапросы могут быть NULL, поскольку если таблица пуста, как в этом примере, значение подзапроса будет равно NULL. Существует несколько ограничений.

- Внешний оператор подзапроса может быть одним из следующих: SELECT, INSERT, UPDATE, DELETE, SET или DO.
- Подзапрос может содержать любые ключевые слова и конструкции, которые допустимы в обычном операторе SELECT: DISTINCT, GROUP BY, ORDER BY, LIMIT, конструкции JOIN, UNION, подсказки, комментарии, функции и так далее.

Поэтому, когда вы видите примеры, приведенные ниже, которые включают довольно краткие конструкции подзапросов (SELECT column1 FROM t1), то представляйте себе свой собственный код, который будет содержать куда более разнообразные и сложные конструкции.

Например, представим, что созданы две таблицы:

```
CREATE TABLE t1 (s1 INT);
INSERT INTO t1 VALUES (1);
CREATE TABLE t2 (s1 INT);
INSERT INTO t2 VALUES (2);
```

Затем выполняется такой запрос:

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

Результатом будет 2, потому что существует строка в t2, содержащая столбец s1, который имеет значение 2.

Подзапрос может быть частью выражения. Если это операнд функции, не забудьте указать скобки. Например:

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

### 6.1.8.2. Сравнения с использованием подзапросов

Наиболее часто подзапросы применяются в такой форме:

*операнд\_не\_подзапроса операция\_сравнения (подзапрос)*

Здесь *операция\_сравнения* – это одна из следующих операций:

= > < >= <= <>

Например:

```
... 'a' = (SELECT column1 FROM t1)
```

Единственное разрешенное место для подзапроса – в правой части выражения сравнения, и вы можете найти некоторые старые системы управления базами данных, которые настаивают на этом.

Ниже приведен пример общей формы сравнения с подзапросом, которую нельзя применять в соединении. Он найдет все значения таблицы t1, которые равны максимальному значению в таблице t2:

```
SELECT column1 FROM t1
WHERE column1 = (SELECT MAX(column2) FROM t2);
```

А вот другой пример, также невозможный в виде соединения, поскольку включает агрегатную функцию в одной из таблиц. Он найдет все строки таблицы t1, содержащие значение, которое встречается дважды:

```
SELECT * FROM t1
WHERE 2 = (SELECT COUNT(column1) FROM t1);
```

### 6.1.8.3. Подзапросы с ANY, IN и SOME

Синтаксис:

*операнд операция\_сравнения ANY (подзапрос)*

*операнд IN (подзапрос)*

*операнд операция\_сравнения SOME (подзапрос)*

Ключевое слово ANY, которое должно следовать за операцией сравнения, означает “возвратить TRUE, если сравнение дает TRUE для ЛЮБОЙ из строк, которые возвращает подзапрос”. Например:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

Предположим, что в таблице t1 есть строка, которая содержит (10). Выражение истинно, если таблица t2 содержит (21,14,7), поскольку в t2 есть значение 7, которое меньше 10. Выражение ложно, если таблица t2 содержит (20,10), либо таблица t2 пуста. Выражение равно UNKNOWN, если таблица t2 содержит значения (NULL, NULL, NULL).

Слово IN – это псевдоним для = ANY, поэтому следующие два оператора одинаковы:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN (SELECT s1 FROM t2);
```

Слово **SOME** – это псевдоним для **ANY**, поэтому показанные ниже два оператора одинаковы:

```
SELECT s1 FROM t1 WHERE s1 <> ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

Слово **SOME** применяется редко, но предыдущий пример показывает, почему оно может оказаться удобным. Для слуха большинства людей английская фраза “a is not equal to any b” (“a не равно любому b”) означает “there is no b which is equal to a” (“нет таких b, которые равны a”), но это не то, что подразумевает синтаксис SQL. Применение **<> SOME** помогает всем правильно понимать действительный смысл запроса.

#### 6.1.8.4. Подзапросы с ALL

Синтаксис:

операнд операция\_сравнения **ALL** (подзапрос)

Слово **ALL**, которое должно следовать за операцией сравнения, означает “возвратить TRUE, если сравнение дает TRUE для всех строк, возвращаемых подзапросом”. Например:

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

Предположим, что в таблице **t1** есть строка, которая содержит (10). Выражение истинно, если таблица **t2** содержит (-5, 0, +5), потому что 10 больше, чем все три значения из таблицы **t2**. Выражение ложно, если таблица **t2** содержит (12, 6, NULL, -100), поскольку только одно значение 12 в таблице **t2** больше 10. Выражение неопределенно (UNKNOWN), если таблица **t2** содержит (0, NULL, 1).

Наконец, если таблица **t2** пуста, результат равен TRUE. Вы можете подумать, что результат будет UNKNOWN, но, сожалеем, он будет именно TRUE. Поэтому, как ни странно, следующий оператор вернет TRUE, если таблица **t2** пуста:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

Однако следующий оператор вернет UNKNOWN, если таблица **t2** пуста:

```
SELECT * FROM t1 WHERE 1 > (SELECT s1 FROM t2);
```

И вдобавок следующий оператор также вернет UNKNOWN, если таблица **t2** пуста:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);
```

В общем случае, *таблицы со значениями NULL и пустые таблицы* – это крайние случаи. Поэтому при написании кода подзапросов всегда принимайте во внимание две упомянутых возможности.

#### 6.1.8.5. Коррелированные подзапросы

Коррелированный подзапрос – это такой подзапрос, который содержит ссылку на столбец, который есть во внешнем запросе. Например:

```
SELECT * FROM t1 WHERE column1 = ANY
(SELECT column1 FROM t2 WHERE t2.column2 = t1.column2);
```

Обратите внимание, что подзапрос содержит ссылку на столбец таблицы **t1**, даже несмотря на то, что конструкция **FROM** подзапроса не упоминает таблицу **t1**. Таким обра-



зом, MySQL выполняет просмотр за пределами подзапроса и находит t1 во внешнем запросе.

Предположим, что таблица t1 содержит строку, в которой column1 = 5 и column2 = 6, в то же время таблица t2 содержит строку, в которой column1 = 5 и column2 = 7. Простое выражение ...WHERE column1=ANY(SELECT column1 FROM t2) будет TRUE, но в этом примере конструкция WHERE внутри подзапроса даст FALSE (поскольку 7 не равно 5), а поэтому и весь подзапрос вернет FALSE.

**Правило видимости:** MySQL вычисляет выражения от внутреннего к внешнему.

Например:

```
SELECT column1 FROM t1 AS x
WHERE x.column1 = (SELECT column1 FROM t2 AS x
WHERE x.column1 = (SELECT column1 FROM t3
WHERE x.column2 = t3.column1));
```

В этом запросе x.column2 должен быть столбцом таблицы t2, потому что SELECT column1 FROM t2 AS x ... переименовывает t2. Это не столбец таблицы t1, так как SELECT column1 FROM t1 ... — другой запрос, который находится во внешнем контексте.

Для подзапросов, находящихся в конструкциях HAVING или ORDER BY, MySQL также ищет имена в списке столбцов внешнего запроса.

В некоторых случаях коррелированный подзапрос оптимизируется. Например:

```
значение IN (SELECT значение_ключа FROM имя_таблицы
WHERE коррелированное_условие)
```

Иначе они могут оказаться неэффективными и, скорее всего, медленными. Если переписать запрос в виде соединения, это может увеличить производительность.

### 6.1.8.6. EXISTS и NOT EXISTS

Если подзапрос вообще возвращает какие-нибудь значения, то EXISTS подзапрос возвращает TRUE, а NOT EXISTS подзапрос — FALSE, например:

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

Традиционно подзапрос в EXISTS начинается с SELECT \*, но он может начинаться с SELECT 5 или SELECT column1, либо с еще чего-нибудь. MySQL игнорирует список SELECT в таком подзапросе, потому что это не важно.

Для предыдущего примера, если t2 содержит любые строки, даже строки, в которых нет ничего кроме значений NULL, то условие EXISTS истинно. Вообще это неправдоподобный пример, поскольку почти всегда подзапрос [NOT] EXISTS содержит корреляцию. Ниже представлены более реалистичные примеры:

- Какие типы магазинов есть в одном или более городов?

```
SELECT DISTINCT store_type FROM Stores
WHERE EXISTS (SELECT * FROM Cities_Stores
WHERE Cities_Stores.store_type = Stores.store_type);
```

- Каких типов магазинов нет ни в одном городе?

```
SELECT DISTINCT store_type FROM Stores
WHERE NOT EXISTS (SELECT * FROM Cities_Stores
WHERE Cities_Stores.store_type = Stores.store_type);
```

- Какой тип магазинов есть во всех городах?

```
SELECT DISTINCT store_type FROM Stores S1
WHERE NOT EXISTS (
    SELECT * FROM Cities WHERE NOT EXISTS (
        SELECT * FROM Cities_Stores
        WHERE Cities_Stores.city = Cities.city
        AND Cities_Stores.store_type = Stores.store_type));
```

В последнем примере представлен дважды вложенный подзапрос NOT EXISTS. То есть конструкция NOT EXISTS содержится внутри другой конструкции NOT EXISTS. Формально он отвечает на вопрос “есть ли город с магазином, которого нет в Stores?”. Но проще сказать, что вложенный NOT EXISTS отвечает на вопрос “истинно ли x для всех y?”.

### 6.1.8.7. Подзапросы, возвращающие строку

До сих пор мы обсуждали *подзапросы, возвращающие столбец (скалярные)*, то есть подзапросы, возвращающие единственное значение столбца. *Строчные подзапросы* – это вариант подзапросов, которые возвращают более одного значения столбца. Ниже представлены два примера:

```
SELECT * FROM t1 WHERE (1,2) = (SELECT column1, column2 FROM t2);
SELECT * FROM t1 WHERE ROW(1,2) = (SELECT column1, column2 FROM t2);
```

Оба эти запроса истинны, если в таблице t2 присутствует строка, в которой column1 = 1 и column2 = 2.

Выражения (1,2) и ROW(1,2) иногда называют *конструктором строки*. Эти два выражения эквивалентны. Они вполне корректны и в других контекстах. Например, следующие два оператора семантически эквивалентны (несмотря на то, что только второй из них может быть оптимизирован):

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

Как правило, конструкторы строки используются для сравнения с подзапросами, возвращающими два или более столбцов. Например, представленный ниже запрос выполняет следующую директиву: “найти все строки таблицы t1, которые есть также и в таблице t2”:

```
SELECT column1,column2,column3
FROM t1
WHERE (column1,column2,column3) IN
    (SELECT column1,column2,column3 FROM t2);
```

### 6.1.8.8. Подзапросы в конструкции FROM

Подзапросы разрешены и в конструкции FROM оператора SELECT. Их синтаксис выглядит следующим образом:

```
SELECT ... FROM (подзапрос) AS имя ...
```

Конструкция AS *имя* является обязательной, поскольку каждая таблица в конструкции FROM должна иметь имя. Все столбцы в списке подзапроса *подзапрос* также должны иметь уникальные имена.

Для того чтобы проиллюстрировать это, предположим, что имеется такая таблица:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
```

Вот как использовать подзапросы в конструкции FROM для данного примера таблицы:

```
INSERT INTO t1 VALUES (1, '1', 1.0);
INSERT INTO t1 VALUES (2, '2', 2.0);
SELECT sb1, sb2, sb3
  FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
 WHERE sb1 > 1;
```

Результат: 2, '2', 4.0.

А вот другой пример. Предположим, что вы хотите знать среднее значение сумм в сгруппированной таблице. Следующий вариант работать не будет:

```
SELECT AVG(SUM(column1)) FROM t1 GROUP BY column1;
```

Однако приведенный ниже запрос выдаст нужную информацию:

```
SELECT AVG(sum_column1)
  FROM (SELECT SUM(column1) AS sum_column1
        FROM t1 GROUP BY column1) AS t1;
```

Отметим, что имя столбца, используемое в подзапросе (sum\_column1) распознается во внешнем запросе.

На данный момент подзапросы в конструкции FROM не могут быть коррелированными подзапросами.

Подзапрос в конструкции FROM будет выполнен (то есть будет создана временная таблица) даже для оператора EXPLAIN, потому что запросы верхнего уровня нуждаются в информации обо всех таблицах на стадии оптимизации.

### 6.1.8.9. Ошибки подзапросов

Появился ряд новых типов ошибок, которые имеют отношение к подзапросам. В этом разделе они сгруппированы вместе, поскольку их просмотр поможет запомнить некоторые важные моменты.

- Неподдерживаемый синтаксис подзапроса:

```
ERROR 1235 (ER_NOT_SUPPORTED_YET)
SQLSTATE = 42000
Message = "This version of MySQL doesn't yet support
'LIMIT & IN/ALL/ANY/SOME subquery'"
```

Это означает, что операторы следующей формы работать не будут, хотя это и случается только в ранних версиях, подобных MySQL 4.1.1:

```
SELECT * FROM t1
  WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1)
```

- Неверное число столбцов в подзапросе:

```
ERROR 1241 (ER_OPERAND_COLUMNNF)
SQLSTATE = 21000
Message = "Operand should contain 1 column(s)"
```

Эта ошибка возникает в случаях наподобие следующего:

```
SELECT (SELECT column1, column2 FROM t2) FROM t1;
```

Допустимо применять подзапросы, которые возвращают несколько столбцов с целью сравнения. См. раздел 6.1.8.7. Но в других контекстах подзапрос должен быть скалярным операндом.

- Неверное количество строк в подзапросе:

```
ERROR 1242 (ER_SUBSELECT_NO_1_ROW)
SQLSTATE = 21000
Message = "Subquery returns more than 1 row"
```

Эта ошибка происходит в операторах вроде приведенного ниже, но только если в таблице t2 имеется более одной строки:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

Такая ошибка может вдруг обнаружиться в коде, который работал в течение нескольких лет, по той причине, что кто-то внес модификацию, изменившую количество строк, возвращаемых подзапросом. Помните, что если нужно найти любое число строк, а не только одну, то правильная форма запроса будет выглядеть так:

```
SELECT * FROM t1 WHERE column1 = ANY (SELECT column1 FROM t2);
```

- Неправильно используется таблица в подзапросе:

```
Error 1093 (ER_UPDATE_TABLE_USED)
SQLSTATE = HY000
Message = "You can't specify target table 'x'
for update in FROM clause"
```

Такая ошибка происходит в случае запроса, подобного представленному ниже:

```
UPDATE t1 SET column2 = (SELECT MAX(column1) FROM t1);
```

Вполне корректно использовать подзапрос для присвоения в операторе UPDATE, поскольку подзапросы разрешены в операторах UPDATE и DELETE, равно как и в операторе SELECT. Однако вы не можете использовать одну и ту же таблицу – в данном случае t1 – и в конструкции FROM подзапроса, и в качестве объекта для манипуляции UPDATE.

Обычно сбой подзапроса приводит к сбою всего внешнего оператора.

#### 6.1.8.10. Оптимизация подзапросов

Когда идет разработка, то долгое время никаких подсказок оптимизации запросов не применяется. Но вы можете попробовать некоторые интересные трюки:

- Использовать конструкции подзапросов, которые касаются количества или порядка строк в подзапросе, например:

```
ELECT * FROM t1 WHERE t1.column1 IN
  (SELECT column1 FROM t2 ORDER BY column1);
SELECT * FROM t1 WHERE t1.column1 IN
  (SELECT DISTINCT column1 FROM t2);
SELECT * FROM t1 WHERE EXISTS
  (SELECT * FROM t2 LIMIT 1);
```

- Заменить соединение подзапросом. Например, используйте такой запрос:

```
SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN (
  SELECT column1 FROM t2);
```

вместо такого:

```
SELECT DISTINCT t1.column1 FROM t1, t2
  WHERE t1.column1 = t2.column1;
```

- Переместить конструкции из внешнего запроса в подзапрос. Например, используйте такой запрос:

```
SELECT * FROM t1
  WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
```

вместо такого:

```
SELECT * FROM t1
  WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
```

И другой пример; используйте следующий запрос:

```
SELECT (SELECT column1 + 5 FROM t1) FROM t2;
```

вместо такого:

```
SELECT (SELECT column1 FROM t1) + 5 FROM t2;
```

- Применять строковый подзапрос вместо коррелированного, например, такой:

```
SELECT * FROM t1
  WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);
```

вместо такого:

```
SELECT * FROM t1
  WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1
    AND t2.column2=t1.column2);
```

- Применять NOT (a = ANY (...)) вместо a <> ALL (...).
- Применять x = ANY(таблица\_содержащая(1,2)) вместо x=1 OR x=2.
- Использовать = ANY вместо EXISTS.

Эти трюки могут заставить программы работать быстрее или медленнее. Используя средства MySQL, подобные функции BENCHMARK(), вы можете получить представление о том, что может помочь в вашей конкретной ситуации. Не беспокойтесь слишком о трансформации подзапросов в соединения, если только вам не нужно обеспечить совместимость с версиями MySQL до 4.1, которые подзапросы не поддерживали.

Ниже представлены некоторые меры по оптимизации, которые принимает сам MySQL.

- MySQL выполняет некоррелированные подзапросы только один раз. Используйте EXPLAIN, чтобы убедиться, что ваш запрос некоррелированный.
- MySQL перезаписывает подзапросы IN/ALL/ANY/SOME, пытаясь получить выигрыш от использования индексов.
- MySQL заменяет подзапросы следующей формы функциями просмотра индексов, которые конструкция EXPLAIN будет описывать как специальный тип соединения:  
... IN (SELECT индексированный\_столбец FROM одиночная\_таблица ...)
- MySQL заменяет выражения следующей формы выражениями с применением MIN() или MAX(), если только не вовлечены NULL-значения и пустые наборы:  
значение {ALL|ANY|SOME} {> | < | >= | <=} (не-коррелированный-подзапрос)

например, следующая конструкция WHERE:

```
WHERE 5 > ALL (SELECT x FROM t)
```

может быть преобразована оптимизатором в такую:

```
WHERE 5 > (SELECT MAX(x) FROM t)
```

В руководстве по внутреннему устройству MySQL (“MySQL Internals Manual”) имеется глава “How MySQL Transforms Subqueries” (“Как MySQL трансформирует подзапросы”). Вы можете получить этот документ, выгрузив исходный дистрибутив MySQL и найдя файл `internals.text` в каталоге `Docs`.

### 6.1.8.11. Замена подзапросов соединениями для ранних версий MySQL

До MySQL 4.1 поддерживались только вложенные запросы для форм `INSERT... SELECT...` и `REPLACE... SELECT...`. Конструкция `IN()` может использоваться в других контекстах для проверки вхождения в множество значений.

Часто можно переписать запрос без подзапроса:

```
SELECT * FROM t1 WHERE id IN (SELECT id FROM t2);
```

Это можно заменить на:

```
SELECT DISTINCT t1.* FROM t1,t2 WHERE t1.id=t2.id;
```

Следующие запросы:

```
SELECT * FROM t1 WHERE id NOT IN (SELECT id FROM t2);
```

```
SELECT * FROM t1 WHERE NOT EXISTS (SELECT id FROM t2 WHERE t1.id=t2.id);
```

могут быть переписаны так:

```
SELECT table1.* FROM table1 LEFT JOIN table2 ON table1.id=table2.id
WHERE table2.id IS NULL;
```

`LEFT [OUTER] JOIN` может оказаться быстрее, чем эквивалентный подзапрос, поскольку сервер может его лучше оптимизировать, и это касается не только одного MySQL. До выхода стандарта SQL-92 внешние соединения не допускались, поэтому подзапросы были единственным способом делать определенные вещи в эти прошедшие времена. Сегодня сервер MySQL и многие другие современные системы баз данных предлагают широкий диапазон типов внешних соединений.

Для более сложных подзапросов часто можно создавать временные таблицы, в которых хранятся промежуточные результаты. Однако в некоторых случаях такой способ неприемлем. Наиболее часто это случается с операторами `DELETE`, для которых стандарт SQL не поддерживает соединений (кроме как в подзапросах). В этой ситуации возможны три пути:

- Первый путь – обновить сервер до версии MySQL 4.1, которая поддерживает подзапросы в операторах `DELETE`.
- Второй путь – использовать процедурный язык программирования (такой как Perl или PHP), чтобы выполнить запрос `SELECT` для получения значений первичных ключей записей, подлежащих удалению, а затем использовать эти значения для построения оператора `DELETE` (`DELETE FROM ... WHERE ключевой_столбец IN (ключ1, ключ2, ...)`).
- Третий путь – использовать интерактивный SQL для автоматического построения набора операторов `DELETE` с использованием MySQL-расширения `CONCAT()` (вместо стандартного оператора `||`). Например:

```
SELECT
  CONCAT('DELETE FROM table1 WHERE pkid = ', "'", table1.pkid, "'", ';')
FROM table1, table2
WHERE table1.column1 = table2.column2;
```

Этот запрос можно поместить в файл сценария, указать его в качестве входного для одного экземпляра программы `mysql` и перенаправить его выходной поток на вход другому экземпляру `mysql`:

```
shell> mysql --skip-column-names mydb < myscript.sql | mysql mydb
```

Сервер MySQL 4.0 поддерживает многотабличные операторы `DELETE`, которые могут использоваться для эффективного удаления строк на основе информации из одной или даже многих таблиц одновременно. Многотабличные операторы `UPDATE` также поддерживаются в MySQL 4.0.

### 6.1.9. Синтаксис TRUNCATE

`TRUNCATE TABLE имя_таблицы`

`TRUNCATE TABLE` полностью очищает таблицу. Логически это эквивалент оператора `DELETE`, который удаляет все строки, но в некоторых случаях имеются определенные отличия практического плана.

Для InnoDB оператор `TRUNCATE TABLE` отображается на `DELETE`, потому здесь разницы нет. Для других механизмов хранения `TRUNCATE TABLE` отличается от `DELETE FROM...` в MySQL 4.0 и выше:

- Операции усечения (`truncate`) удаляют и пересоздают таблицы, что значительно быстрее, чем удалять строки одну за другой.
- Операции усечения не являются безопасными в отношении транзакций; вы получите ошибку, если на сервере существуют активные транзакции или активные блокировки таблиц.
- Количество удаленных строк не возвращается.
- До тех пор, пока файл определения таблицы `имя_таблицы.frm` правильный, таблица может быть создана повторно с помощью оператора `TRUNCATE TABLE` как пустая, даже если файл данных или индексные файлы повреждены.
- Дескриптор таблицы не запоминает последнего использованного значения `AUTO_INCREMENT`, а начинает отсчет сначала. Это верно даже для механизма MyISAM, который обычно не использует повторно значений последовательности.

В MySQL 3.23 оператор `TRUNCATE TABLE` отображается на `COMMIT; DELETE FROM имя_таблицы`, то есть ведет себя подобно `DELETE`. См. раздел 6.1.1.

`TRUNCATE TABLE` – это SQL-оператор Oracle. Он был добавлен в MySQL 3.23.28, хотя в версиях от 3.23.28 до 3.23.32 ключевое слово `TABLE` может пропускаться.

### 6.1.10. Синтаксис UPDATE

Однотабличный синтаксис:

```
UPDATE [LOW_PRIORITY] [IGNORE] имя_таблицы
SET имя_столбца1=выражение1 [,имя_столбца2=выражение2 ...]
[WHERE определение_where]
[ORDER BY ...]
[LIMIT количество_строк]
```

Многотабличный синтаксис:

```
UPDATE [LOW_PRIORITY] [IGNORE] имя_таблицы [, имя_таблицы ...]  
SET имя_столбца1=выражение1 [, имя_столбца2=выражение2 ...]  
[WHERE определение_where]
```

Оператор UPDATE обновляет столбцы существующих строк таблицы новыми значениями. Конструкция SET перечисляет столбцы, подлежащие модификации, и значения, которые им присваиваются. Если указана конструкция WHERE, она задает, какие строки должны быть обновлены. В противном случае обновляются все строки таблицы. Если указана конструкция ORDER BY, строки будут обновлены в заданном порядке. Конструкция LIMIT накладывает ограничение на количество обновляемых строк.

Оператор UPDATE поддерживает следующие модификаторы:

- Если указано ключевое слово LOW\_PRIORITY, выполнение UPDATE откладывается до тех пор, пока все другие клиенты завершат чтение таблицы.
- Если указано ключевое слово IGNORE, операция обновления не будет прервана, даже если возникнут ошибки дублирования ключа. Строки, которые приводят к конфликтам, обновлены не будут.

Если вы используете столбцы из таблицы *имя\_таблицы* в выражениях, UPDATE использует текущее значение столбцов. Например, следующий оператор увеличивает значение столбца age на единицу:

```
mysql> UPDATE persondata SET age=age+1;
```

Присвоения в UPDATE выполняются слева направо. Например, следующий оператор удваивает значение столбца age, а затем увеличивает на единицу:

```
mysql> UPDATE persondata SET age=age*2, age=age+1;
```

Если вы устанавливаете значение столбца в то, которое он имеет, MySQL обнаруживает это и не выполняет обновление.

Если вы обновляете столбец, который был объявлен как NOT NULL, присваивая ему значение NULL, он устанавливается в значение по умолчанию, соответствующее конкретному типу данных и увеличивает счетчик предупреждений на единицу. Значение по умолчанию равно 0 для числовых столбцов, пустая строка (') для символьных и "нулевое" значение для столбцов типа даты и времени.

UPDATE возвращает количество строк, которые фактически были обновлены. В MySQL 3.22 и более поздних версиях функция mysql\_info() программного интерфейса C API возвращает количество строк, которые соответствовали запросу и были обновлены, а также количество предупреждений, возникших во время выполнения UPDATE.

Начиная с MySQL 3.23, можно использовать LIMIT количество\_строк для ограничения области действия UPDATE.

Конструкция LIMIT работает следующим образом:

- До MySQL 4.0.13 LIMIT была ограничением количества обработанных строк. Оператор завершал работу, как только обновлял количество\_строк строк, удовлетворявших условию WHERE.
- Начиная с MySQL 4.0.13, LIMIT – ограничение соответствия строк. Оператор завершает работу, как только найдет количество\_строк строк, удовлетворяющих условию WHERE, независимо от того, были ли они действительно обновлены.



Если оператор UPDATE включает конструкцию ORDER BY, то строки обновляются в порядке, заданном этой конструкцией. ORDER BY может применяться, начиная с MySQL 4.0.0.

Начиная с MySQL 4.0.0, также можно выполнять операции UPDATE, которые работают с несколькими таблицами сразу:

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

Этот пример демонстрирует внутреннее объединение, использующее оператор запятой, но многотабличные UPDATE могут использовать любой тип объединений, допустимый в операторе SELECT, например, LEFT JOIN.

#### На заметку!

Вы не можете применять ORDER BY или LIMIT в многотабличных операторах UPDATE.

До версии MySQL 4.0.18 необходимо было иметь привилегию UPDATE для всех таблиц, используемых в многотабличном UPDATE, даже если они фактически не обновлялись. Начиная с MySQL 4.0.18, для таких таблиц, чьи столбцы только читаются, но не обновляются, необходимо иметь только привилегию SELECT.

Если вы используете многотабличный оператор UPDATE в отношении таблиц InnoDB, у которых определены ограничения внешних ключей, оптимизатор MySQL может обрабатывать их в порядке, отличном от того, который задается их отношениями “родительский-дочерний”. В этом случае оператор завершится ошибкой и будет выполнен откат транзакции. Вместо этого обновляйте одну таблицу и полагайтесь на свойство ON UPDATE, которое предоставляет механизм InnoDB для автоматического обновления связанных таблиц.

## 6.2. Операторы определения данных

### 6.2.1. Синтаксис ALTER DATABASE

```
ALTER DATABASE имя_базы_данных
    спецификация_alter [, спецификация_alter] ...
```

спецификация\_alter:

```
[DEFAULT] CHARACTER SET имя_набора_символов
| [DEFAULT] COLLATE имя_порядка_сопоставления
```

ALTER DATABASE позволяет изменять общие характеристики базы данных. Эти характеристики хранятся в файле db.opt, находящемся в каталоге данных. Чтобы использовать ALTER DATABASE, необходимо иметь привилегию ALTER для этой базы данных.

Конструкция CHARACTER SET изменяет набор символов по умолчанию для данной базы данных. Конструкция COLLATE изменяет порядок сопоставления, используемый по умолчанию в базе данных. Наборы символов и порядки сопоставления обсуждаются в главе 3.

ALTER DATABASE появился в версии MySQL 4.1.1.

### 6.2.2. Синтаксис ALTER TABLE

```
ALTER [IGNORE] TABLE имя_таблицы
    спецификация_alter [, спецификация_alter] ...
```

спецификация\_alter:

```

    ADD [COLUMN] определение_столбца [FIRST | AFTER имя_столбца]
  | ADD [COLUMN] {определение_столбца,...}
  | ADD INDEX [имя_индекса] {тип_индекса} (имя_столбца_индекса,...)
  | ADD [CONSTRAINT [символ]]
      PRIMARY KEY [тип_индекса] (имя_столбца_индекса,...)
  | ADD [CONSTRAINT [символ]]
      UNIQUE [имя_индекса] [тип_индекса] (имя_столбца_индекса,...)
  | ADD [FULLTEXT|SPATIAL] [имя_индекса] (имя_столбца_индекса,...)
  | ADD [CONSTRAINT [символ]]
      FOREIGN KEY [имя_индекса] (имя_столбца_индекса,...)
          [определение_ссылки]
  | ALTER [COLUMN] имя_столбца {SET DEFAULT литерал | DROP DEFAULT}
  | CHANGE [COLUMN] старое_имя_столбца определение_столбца
      [FIRST|AFTER имя_столбца]
  | MODIFY [COLUMN] определение_столбца [FIRST | AFTER имя_столбца]
  | DROP [COLUMN] имя_столбца
  | DROP PRIMARY KEY
  | DROP INDEX имя_индекса
  | DROP FOREIGN KEY символ_fk
  | DISABLE KEYS
  | ENABLE KEYS
  | RENAME [TO] новое_имя_таблицы
  | ORDER BY имя_столбца
  | CONVERT TO CHARACTER SET имя_набора_символов
      [COLLATE имя_порядка_сопоставления]
  | [DEFAULT] CHARACTER SET имя_набора_символов
      [COLLATE имя_порядка_сопоставления]
  | DISCARD TABLESPACE
  | IMPORT TABLESPACE
  | опции_таблицы

```

ALTER TABLE позволяет изменить структуру существующей таблицы. Например, можно добавлять или удалять столбцы, создавать или уничтожать индексы, изменять тип существующих столбцов или переименовывать столбцы и сами таблицы. Можно также изменять комментарии для таблиц и их тип.

Синтаксис многих допустимых изменений в этом операторе похож на аналогичные конструкции оператора CREATE TABLE. См. раздел 6.2.5.

Если вы использовали ALTER TABLE для изменения спецификаций столбцов, но DESCRIBE имя\_таблицы показывает, что ваши столбцы не изменились, возможно, MySQL проигнорировал ваши изменения по одной из причин, описанных в разделе 6.2.5.2. Например, если вы пытаетесь изменить тип столбца VARCHAR на CHAR, MySQL будет по-прежнему использовать VARCHAR, если таблица содержит другие столбцы переменной длины.

ALTER TABLE работает, создавая временную копию оригинальной таблицы. Изменения производится над копией, затем исходная таблица удаляется, а новая переименовывается. В процессе работы ALTER TABLE исходная таблица остается доступной по чтению другим клиентам. Обновления и запись в эту таблицу приостанавливаются до тех пор, пока новая таблица не будет готова, затем автоматически перенаправляются на новую таблицу, без каких-либо потерь.

Помните, что если вы используете любые опции ALTER TABLE за исключением RENAME, MySQL всегда создает временную таблицу, даже если нет строгой необходимости в копировании данных (например, когда вы всего лишь переименовываете столбец). Мы планируем исправить это в будущем, но поскольку ALTER TABLE – оператор, используемый нечасто, эта задача не имеет высокого приоритета в наших планах. Для таблиц MyISAM вы можете ускорить процедуру пересоздания индексов (что является самой медленной частью в процессе выполнения этого оператора), присвоив системной переменной `myisam_sort_buffer_size` большее значение.

- Для использования ALTER TABLE необходимо иметь привилегии ALTER, INSERT и CREATE для таблицы.
- IGNORE – это расширение MySQL стандарта SQL. Это слово управляет тем, как ALTER TABLE работает в случае дублирования уникальных ключей в новой таблице. Если IGNORE не указано, копирование прерывается и выполняется откат при обнаружении ошибки дублирования ключей. Если же IGNORE указано, то из строк, дублирующих уникальный ключ, используется только первая. Остальные удаляются.
- Вы можете использовать множество конструкций ADD, ALTER, DROP и CHANGE в пределах одного оператора ALTER TABLE. Это также MySQL-расширение стандарта SQL, в котором разрешается только по одному такому предложению в операторе ALTER TABLE.
- CHANGE *имя\_столбца*, DROP *имя\_столбца* и DROP INDEX – это MySQL-расширения стандарта SQL.
- MODIFY – это расширение ALTER TABLE от Oracle.
- Слово COLUMN – совершенно необязательное и может быть опущено.
- Если вы используете ALTER TABLE *имя\_таблицы* RENAME TO *новое\_имя\_таблицы* без каких-либо опций, MySQL просто переименовывает все файлы, имеющие отношение к таблице *имя\_таблицы*. Нет необходимости создавать временную таблицу. (Вы также можете использовать оператор RENAME TABLE для переименования таблиц. См. раздел 6.2.9.)
- Конструкции *определение\_столбца* используют тот же синтаксис для ADD и CHANGE, что и CREATE TABLE. Следует отметить, что синтаксис включает имя столбца, а не только его тип. См. раздел 6.2.5.
- Вы можете переименовывать столбцы, используя конструкцию CHANGE *старое\_имя\_столбца* *определение\_столбца*. Чтобы сделать это, укажите старые и новые имена, а также типы столбцов. Например, чтобы переименовать столбец INTEGER из *a* в *b*, можно поступить так:

```
mysql> ALTER TABLE t1 CHANGE a b INTEGER;
```

Если вы хотите изменить только тип столбца, не меняя имени, синтаксис CHANGE все равно требует указания старого и нового имени столбца, даже если эти имена одинаковы. Например:

```
mysql> ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

Однако, начиная с версии MySQL 3.22.16a, вы также можете использовать MODIFY для изменения типа столбца без его переименования:

```
mysql> ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

- Если вы используете `CHANGE` или `MODIFY`, чтобы укоротить столбец, который используется в ключе индекса частично (например, если индекс построен по первым 10 символам столбца `VARCHAR`), вы не можете сделать столбец короче, чем количество символов, по которым построен индекс.
- В MySQL 3.22 и более поздних версиях вы можете воспользоваться `FIRST` или `AFTER имя_столбца`, чтобы добавить новый столбец в указанное место строки таблицы. По умолчанию столбцы добавляются в конец структуры. Начиная с MySQL 4.0.1, `FIRST` или `AFTER имя_столбца` можно также применять в операциях `CHANGE` и `MODIFY`.
- `ALTER COLUMN` указывает новое значение по умолчанию для столбца или удаляет старое. Если старое значение по умолчанию удалено, и столбец может принимать значение `NULL`, то значением по умолчанию становится `NULL`. Если столбец не может принимать значение `NULL`, MySQL присваивает ему значение по умолчанию в зависимости от его типа, как описано в разделе 6.2.5.
- `DROP INDEX` удаляет индекс. Это расширение стандарта SQL от MySQL. См. раздел 6.2.7.
- Если столбец удаляется из структуры таблицы, столбец также удаляется из всех индексов, частью ключа которых он была. Если все столбцы, составляющие ключ индекса, удалены, индекс также удаляется.
- Если таблица содержит только один столбец, столбец не может быть удален. Если вы при этом имеете в виду удаление таблицы, используйте вместо этого `DROP TABLE`.
- `DROP PRIMARY KEY` уничтожает индекс первичного ключа. (До версии MySQL 4.1.2, если таблица не имела первичного ключа, оператор `DROP PRIMARY KEY` уничтожал первый уникальный индекс таблицы. MySQL помечает первый уникальный ключ как первичный (`PRIMARY KEY`), если таковой не специфицирован явно. Если вы добавляете таблице `UNIQUE INDEX` или `PRIMARY KEY`, он сохраняется перед любым неуникальным индексом, чтобы MySQL мог обнаруживать случаи дублирования ключа как можно раньше.)
- `ORDER BY` позволяет создавать новую таблицу со строками в определенном порядке. Следует отметить, что последовательность строк изменится после выполнения вставок и удалений. Эта опция в основном применима, когда вы знаете, что запросы будут выполняться в основном так, чтобы извлекать строки в определенном порядке. Используя эту опцию после существенных изменений в таблице, вы можете добиться увеличения производительности. В некоторых случаях можно облегчить MySQL задачу сортировки, если данные в таблице будут расположены в определенном порядке по значению столбца, в котором позже их придется извлекать.
- Если вы используете `ALTER TABLE` для таблицы `MyISAM`, все неуникальные индексы создаются в отдельном пакете (как при выполнении `REPAIR TABLE`). Это может значительно ускорить выполнение `ALTER TABLE`, если индексов много.  
Начиная с MySQL 4.0, это средство может быть активизировано явно. `ALTER TABLE...DISABLE KEYS` сообщает MySQL, что нужно прекратить обновление не-уникальных индексов в таблице `MyISAM`. `ALTER TABLE...ENABLE KEYS` затем может быть применен для воссоздания индексов. MySQL делает это по специальному алгоритму, который гораздо быстрее, нежели при вставке строк по одной, поэто-

му отключение индексов перед операцией пакетной вставки должно дать ощутимое ускорение.

- Конструкции FOREIGN KEY и REFERENCES поддерживаются механизмом хранения InnoDB, который реализует ADD [CONSTRAINT [символ]] FOREIGN KEY (...) REFERENCES... (...). Для других механизмов хранения эти конструкции принимаются, но игнорируются. Конструкция CHECK распознается, но игнорируется всеми механизмами хранения. См. раздел 6.2.5. Причина, по которой этот синтаксис принимается, но игнорируется, связана с обеспечением совместимости, упрощением переноса кода с других SQL-серверов, и необходимостью запуска приложений, которые создают таблицы со ссылками. См. раздел 1.8.5.

- Начиная с версии MySQL 4.0.13, InnoDB поддерживает применение ALTER TABLE для удаления внешних ключей:

```
ALTER TABLE имя_таблицы DROP FOREIGN KEY символ_fk;
```

- ALTER TABLE игнорирует опции DATA DIRECTORY и INDEX DIRECTORY.

- Начиная с MySQL 4.1.2 и последующих версий, если вы хотите изменить набор символов для всех символьных столбцов (CHAR, VARCHAR, TEXT) на другой, используйте оператор вроде следующего:

```
ALTER TABLE имя_таблицы CONVERT TO CHARACTER SET имя_набора_символов;
```

Это удобно, например, после обновления версии MySQL 4.0.x до MySQL 4.1.x. См. раздел 3.10.

### Внимание!

Предыдущая операция преобразует значения столбцов между старым и новым наборами символов. Это не то, что нужно, если у вас есть столбец в одном наборе символов (вроде latin1), но содержит значения, в действительности использующие какой-то другой, несовместимый набор символов (вроде utf8). В этом случае для каждой такой столбцы потребуется выполнить следующие операторы:

```
ALTER TABLE t1 CHANGE c1 c1 BLOB;
```

```
ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8;
```

Это работает, поскольку не происходит никакого преобразования столбцов типа BLOB. Для изменения только набора символов по умолчанию для таблицы воспользуйтесь следующим оператором:

```
ALTER TABLE имя_таблицы DEFAULT CHARACTER SET имя_набора_символов;
```

Слово DEFAULT является необязательным. Набор символов по умолчанию – это тот набор, который применяется для новых столбцов, добавляемых к таблице, если не указывается явно (например, в ALTER TABLE...ADD column).

### Внимание!

Начиная с MySQL 4.1.2 и выше, ALTER TABLE...DEFAULT CHARACTER SET и ALTER TABLE...CHARACTER SET эквивалентны и изменяют только набор символов по умолчанию. В выпусках MySQL 4.1, предшествующих 4.1.2, ALTER TABLE...DEFAULT CHARACTER SET изменяет набор символов по умолчанию, а ALTER TABLE...CHARACTER SET изменяет набор символов по умолчанию и преобразует все столбцы.

- Для таблицы InnoDB, которая была создана в своем собственном табличном пространстве в файле `.ibd`, этот файл может быть отброшен и импортирован. Чтобы отбросить (discard) файл `.ibd`, воспользуйтесь следующим оператором:

```
ALTER TABLE имя_таблицы DISCARD TABLESPACE;
```

Приведенный выше оператор удалит текущий файл `.ibd`, поэтому сначала убедитесь, что у вас есть резервная копия. Попытки обратиться к таблице, когда файл табличного пространства отброшен, приводят к ошибке.

Чтобы импортировать обратно резервную копию файла `.ibd`, скопируйте его в каталог данных и выполните следующий оператор:

```
ALTER TABLE имя_таблицы IMPORT TABLESPACE;
```

- С помощью функции С API `mysql_info()` можно определить, сколько записей было скопировано и (когда применяется IGNORE) сколько записей было удалено из-за дублирования значений уникальных ключей.

Ниже показаны некоторые примеры, демонстрирующие применение оператора `ALTER TABLE`. Начнем с таблицы `t1`, созданной следующим образом:

```
mysql> CREATE TABLE t1 (a INTEGER, b CHAR(10));
```

Чтобы переименовать таблицу из `t1` в `t2`, воспользуйтесь таким оператором:

```
mysql> ALTER TABLE t1 RENAME t2;
```

Чтобы изменить тип столбца `a` с `INTEGER` на `TINYINT NOT NULL` (оставив его имя прежним), и чтобы изменить тип столбца `b` с `CHAR(10)` на `CHAR(20)` и переименовать его на `c`, воспользуйтесь таким оператором:

```
mysql> ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

Чтобы добавить новый столбец типа `TIMESTAMP` с именем `d`, выполните следующий оператор:

```
mysql> ALTER TABLE t2 ADD d TIMESTAMP;
```

Чтобы добавить индексы по столбцу `d` и столбцу `a`, воспользуйтесь таким оператором:

```
mysql> ALTER TABLE t2 ADD INDEX (d), ADD INDEX (a);
```

Чтобы удалить столбец `c`, выполните такой оператор:

```
mysql> ALTER TABLE t2 DROP COLUMN c;
```

Чтобы добавить новый столбец с именем `c` целого типа, имеющий свойство `AUTO_INCREMENT`, воспользуйтесь следующим оператором:

```
mysql> ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,  
-> ADD PRIMARY KEY (c);
```

Следует отметить, что мы проиндексировали столбец `c` (как первичный ключ), поскольку столбцы `AUTO_INCREMENT` должны быть проиндексированы. Кроме того, столбец `c` объявлен как `NOT NULL`, так как столбцы первичного ключа не могут быть `NULL`.

Когда добавляется столбец с атрибутом `AUTO_INCREMENT`, он автоматически заполняется значениями последовательности целых чисел. Для таблиц MyISAM можно указать первое число последовательности, выполнив `SET INSERT_ID=значение` перед запуском `ALTER TABLE`, или воспользовавшись опцией таблицы `AUTO_INCREMENT=значение`. См. раздел 6.5.3.1.

Для таблиц MyISAM, если вы не изменяете столбец AUTO\_INCREMENT, числовая последовательность не будет затронута. Если вы удаляете столбец AUTO\_INCREMENT, а затем добавляете другой столбец AUTO\_INCREMENT, то числа будут перенумерованы, начиная с 1.

В MySQL 3.23.50 и более поздних версиях InnoDB разрешает добавлять новые ограничения внешних ключей к таблице с помощью оператора ALTER TABLE:

```
ALTER TABLE имя_таблицы
  ADD [CONSTRAINT символ] FOREIGN KEY [идентификатор] (имя_столбца_индекса, ...)
  REFERENCES имя_таблицы (имя_столбца_индекса, ...)
  [ON DELETE {CASCADE | SET NULL | NO ACTION | RESTRICT}]
  [ON UPDATE {CASCADE | SET NULL | NO ACTION | RESTRICT}]
```

Не забудьте сначала создать все необходимые индексы. Используя ALTER TABLE, в таблицу можно также добавлять внешние ключи, ссылающиеся на эту же таблицу.

Начиная с MySQL 4.0.13, InnoDB поддерживает применение ALTER TABLE для удаления внешних ключей:

```
ALTER TABLE имя_таблицы DROP FOREIGN KEY символ_fk;
```

Если конструкция FOREIGN KEY включает имя CONSTRAINT, когда создается внешний ключ, то вы можете сослаться на него, чтобы удалить этот внешний ключ. (Имя ограничения внешнего ключа может присваиваться, начиная с MySQL 4.0.18.) В противном случае InnoDB генерирует внутреннее значение символ\_fk при создании внешнего ключа. Чтобы узнать это значение, когда вам понадобится удалить внешний ключ, воспользуйтесь оператором SHOW CREATE TABLE, например:

```
mysql> SHOW CREATE TABLE ibtest11c\G
***** 1. row *****
      Table: ibtest11c
Create Table: CREATE TABLE `ibtest11c` (
  `A` int(11) NOT NULL auto_increment,
  `D` int(11) NOT NULL default '0',
  `B` varchar(200) NOT NULL default '',
  `C` varchar(175) default NULL,
  PRIMARY KEY (`A`,`D`,`B`),
  KEY `B` (`B`,`C`),
  KEY `C` (`C`),
  CONSTRAINT `0_38775` FOREIGN KEY (`A`,`D`)
  REFERENCES `ibtest11a` (`A`,`D`)
  ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `0_38776` FOREIGN KEY (`B`,`C`)
  REFERENCES `ibtest11a` (`B`,`C`)
  ON DELETE CASCADE ON UPDATE CASCADE
) TYPE=InnoDB CHARSET=latin1
1 row in set (0.01 sec)

mysql> ALTER TABLE ibtest11c DROP FOREIGN KEY 0_38775;
```

В версиях MySQL до 3.23.50 операторы ALTER TABLE и CREATE INDEX не должны применяться для таблиц, которые имеют ограничения внешних ключей, либо на которые ссылаются внешние ключи других таблиц. Любой оператор ALTER TABLE удаляет все ограничения внешних ключей, определенные в таблице. Нельзя также применять ALTER TABLE к таблицам, на которые имеются ссылки. Вместо этого нужно пользоваться DROP

TABLE и CREATE TABLE для модификации схемы. Когда сервер MySQL выполняет ALTER TABLE, он может скрыто выполнять RENAME TABLE, и это сбивает с толку ограничения внешних ключей, которые ссылаются на таблицу. В MySQL оператор CREATE INDEX обрабатывается как ALTER TABLE, поэтому вышесказанное касается и этого оператора.

См. раздел А.3.1.

### 6.2.3. Синтаксис CREATE DATABASE

```
CREATE DATABASE [IF NOT EXISTS] имя_базы_данных
[спецификация_create [, спецификация_create] ...]
```

*спецификация\_create*:

```
[DEFAULT] CHARACTER SET имя_набора_символов
| [DEFAULT] COLLATE имя_порядка_сопоставления
```

CREATE DATABASE создает базу данных с указанным именем. Для использования CREATE DATABASE необходимо иметь привилегию CREATE для базы данных.

Правила именования баз данных описаны в разделе 2.2. Если база данных с таким именем существует, и не было указано IF NOT EXISTS, генерируется ошибка.

Начиная с MySQL 4.1.1, опция *спецификация\_create* может указываться для определения характеристик базы данных. Характеристики базы данных сохраняются в файле db.opt, расположенном в каталоге данных. Конструкция CHARACTER SET определяет набор символов для базы данных по умолчанию. Конструкция COLLATION задает порядок сопоставления по умолчанию. Наборы символов и порядки сопоставления обсуждаются в главе 3.

Базы данных в MySQL реализованы в виде каталогов, которые содержат файлы, соответствующие таблицам базы данных. Поскольку изначально в базе нет никаких таблиц, оператор CREATE DATABASE только создает подкаталог в каталоге данных MySQL (и файл db.opt для версии MySQL 4.1.1 и выше).

Для создания базы данных можно также воспользоваться программой mysqladmin.

### 6.2.4. Синтаксис CREATE INDEX

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX имя_индекса [тип_индекса]
ON имя_таблицы (имя_столбца_индекса, ...)
```

*имя\_столбца\_индекса*:

```
имя_столбца [(длина)] [ASC | DESC]
```

В версии MySQL 3.22 и выше CREATE INDEX отображается на оператор ALTER TABLE, чтобы создавать индексы. См. раздел 6.2.2. До MySQL 3.22 оператор CREATE INDEX ничего не делал.

Обычно все индексы таблицы создаются в то же время, когда создается таблицы с помощью оператора CREATE TABLE. См. раздел 6.2.5. Синтаксис CREATE INDEX позволяет добавлять индексы в существующую таблицу.

Список столбцов в форме (*столбец1*, *столбец2*, ...) создает составной индекс. Ключ индекса формируется путем конкатенации значений указанных столбцов.

Для столбцов CHAR и VARCHAR ключом индекса могут служить только части столбцов, если использовать синтаксис *имя\_столбца* (*длина*), чтобы индексировать только префиксы значений столбцов длиной первые *длина* символов. Столбцы типов BLOB и TEXT также могут быть индексированы, но *длина* префикса для них *должна* быть указана.



Оператор, приведенный ниже, создает индекс, используя в качестве ключа первые 10 символов столбца name:

```
CREATE INDEX part_of_name ON customer (name(10));
```

Поскольку большинство имен обычно отличаются друг от друга первыми 10 символами, этот индекс не должен быть намного медленнее, чем индекс, созданный по полным значениям столбца name. Кроме того, применяя части столбцов в качестве ключей индекса, можно значительно сократить размер индексного файла, что экономит дисковое пространство и может ускорить выполнение операции INSERT.

Префикс может быть до 255 символов длиной (или 1000 символов для таблиц MyISAM и InnoDB, начиная с версии MySQL 4.1.2). Отметим, что максимальная длина префикса измеряется в байтах, в то время как длина префикса, которая указывается в операторе CREATE INDEX, интерпретируется как количество символов. Принимайте это во внимание, когда указываете длину префикса для столбца, использующего многобайтный набор символов.

Следует также отметить, что добавлять индекс по столбцу, допускающему значения NULL, можно только в версиях MySQL, начиная с 3.23.2 и только для таблиц типа MyISAM, InnoDB или BDB. Индексы по столбцам BLOB и TEXT можно добавлять, если вы работаете с MySQL версии 3.23.2 и выше и используете таблицы MyISAM или BDB, либо MySQL 4.0.14 и выше и таблицы InnoDB.

Спецификация *имя\_столбца\_индекса* может завершаться ASC или DESC. Эти ключевые слова введены для будущих расширений, чтобы задавать сохранение значения индекса по возрастанию или убыванию. В настоящее время они распознаются, однако игнорируются. Значения индекса всегда сохраняются в порядке возрастания.

Индексы FULLTEXT могут индексировать только столбцы типов CHAR, VARCHAR и TEXT, и только в таблицах MyISAM. Эти индексы доступны в версии MySQL 3.23.23 и выше. См. раздел 5.6.

Индексы SPATIAL могут индексировать только пространственные (spatial) столбцы, и только в таблицах MyISAM. Индексы SPATIAL доступны в версии MySQL 4.1 и выше. Пространственные типы столбцов описаны в главе 7.

## 6.2.5. Синтаксис CREATE TABLE

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] имя_таблицы
  [{определение_create,...}]
  [опции_таблицы] [оператор_select]
```

или

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] имя_таблицы
  [{() LIKE старое_имя_таблицы ()];
```

*определение\_create*:

```
  определение_столбца
  | [CONSTRAINT [символ]] PRIMARY KEY [тип_индекса]
    (имя_столбца_индекса,...)
  | KEY [имя_индекса] [тип_индекса] (имя_столбца_индекса,...)
  | INDEX [имя_индекса] [тип_индекса] (имя_столбца_индекса,...)
  | {CONSTRAINT [символ]] UNIQUE [INDEX]
    [имя_индекса] [тип_индекса] (имя_столбца_индекса,...)
  | {FULLTEXT|SPATIAL} [INDEX] [index_name] (имя_столбца_индекса,...)
  | {CONSTRAINT [символ]] FOREIGN KEY
```

```

    {имя_индекса} (имя_столбца_индекса,...) {определение_ссылки}
| CHECK (выражение)

определение_столбца:
    имя_столбца тип [NOT NULL | NULL] [DEFAULT значение_по_умолчанию]
    [AUTO_INCREMENT] [{PRIMARY} KEY] [COMMENT 'строка']
    {определение_ссылки}

тип:
    TINYINT[(длина)] [UNSIGNED] [ZEROFILL]
| SMALLINT[(длина)] [UNSIGNED] [ZEROFILL]
| MEDIUMINT[(длина)] [UNSIGNED] [ZEROFILL]
| INT[(длина)] [UNSIGNED] [ZEROFILL]
| INTEGER[(длина)] [UNSIGNED] [ZEROFILL]
| BIGINT[(длина)] [UNSIGNED] [ZEROFILL]
| REAL[(длина,цифр_после_точки)] [UNSIGNED] [ZEROFILL]
| DOUBLE[(длина,цифр_после_точки)] [UNSIGNED] [ZEROFILL]
| FLOAT[(длина,цифр_после_точки)] [UNSIGNED] [ZEROFILL]
| DECIMAL(длина,цифр_после_точки) [UNSIGNED] [ZEROFILL]
| NUMERIC(длина,цифр_после_точки) [UNSIGNED] [ZEROFILL]
| DATE
| TIME
| TIMESTAMP
| DATETIME
| CHAR(длина) [BINARY | ASCII | UNICODE]
| VARCHAR(длина) [BINARY]
| TINYBLOB
| BLOB
| MEDIUMBLOB
| LONGBLOB
| TINYTEXT
| TEXT
| MEDIUMTEXT
| LONGTEXT
| ENUM(значение1, значение2, значение3,...)
| SET(значение1, значение2, значение3,...)
| пространственный_тип

имя_столбца_индекса:
    имя_столбца [(длина)] [ASC | DESC]

определение_ссылки:
    REFERENCES имя_таблицы [(имя_столбца_индекса,...)]
    [MATCH FULL | MATCH PARTIAL]
    [ON DELETE опция_ссылки]
    [ON UPDATE опция_ссылки]

опция_ссылки:
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

опции_таблицы: опция_таблицы [опция_таблицы] ...

опция_таблицы:
    {ENGINE|TYPE} = {BDB|HEAP|ISAM|InnoDB|MERGE|MRG_MYISAM|MYISAM}
| AUTO_INCREMENT = значение
| AVG ROW LENGTH = значение
| CHECKSUM = {0 | 1}

```

```

| COMMENT = 'строка'
| MAX_ROWS = значение
| MIN_ROWS = значение
| PACK_KEYS = { 0 | 1 | DEFAULT }
| PASSWORD = 'строка'
| DELAY_KEY_WRITE = { 0 | 1 }
| ROW_FORMAT = { DEFAULT | DYNAMIC | FIXED | COMPRESSED }
| RAID_TYPE = { 1 | STRIPED | RAID0 }
  RAID_CHUNKS = значение
  RAID_CHUNKSIZE = значение
| UNION = (имя_таблицы[, имя_таблицы]...)
| INSERT_METHOD = { NO | FIRST | LAST }
| DATA DIRECTORY = 'абсолютный путь к каталогу'
| INDEX DIRECTORY = 'абсолютный путь к каталогу'
| [DEFAULT] CHARACTER SET имя_набора_символов
| [COLLATE имя_порядка_сопоставления]

```

оператор `select`:

```
[IGNORE | REPLACE] [AS] SELECT ... (любой разрешенный оператор select)
```

`CREATE TABLE` создает таблицу с указанным именем. Необходимо иметь привилегию `CREATE` для таблиц.

Правила именования таблиц описаны в разделе 2.2. По умолчанию таблица создается в текущей базе данных. Если таблица уже существует, если нет текущей базы данных или если вообще нет базы данных, генерируется ошибка.

В MySQL 3.22 и более поздних версиях имя таблицы может специфицироваться как имя\_базы\_данных.имя\_таблицы, чтобы создать таблицу в указанной базе данных. Это работает независимо от того, есть база данных по умолчанию или нет. Если вы используете идентификаторы в кавычках, то берите в кавычки имя базы и имя таблицы отдельно. Например, 'mydb'. 'mytbl' — правильно, а 'mydb.mytbl' — нет.

Начиная с MySQL 3.23 при создании таблицы можно использовать ключевое слово `TEMPORARY`. Временная таблица видима только для текущего сеанса и уничтожается автоматически при закрытии соединения. Это означает, что два разных подключения могут использовать одинаковое имя временной таблицы, не конфликтуя друг с другом и с существующей постоянной таблицей с тем же именем. (Постоянная таблица остается скрытой до тех пор, пока не будет удалена временная.) Начиная с MySQL 4.0.2, для создания временных таблиц нужно иметь привилегию `CREATE TEMPORARY TABLES`.

В MySQL 3.23 и более поздних версиях можно использовать ключевые слова `IF NOT EXISTS`, при этом, если таблицы существовала, ошибка не генерируется. Отметим, что никакой проверки, чтобы существующая таблица имела структуру, идентичную той, что задана в операторе `CREATE TABLE`, не производится.

MySQL представляет каждую таблицу файлом формата таблицы `.frm`, расположенном в каталоге базы данных. Механизм хранения может создавать для таблицы и другие файлы. В случае таблицы `MyISAM` механизм хранения создает три файла для таблицы по имени имя\_таблицы.

Файл	Назначение
<u>имя_таблицы</u> .frm	Файл описания формата таблицы.
<u>имя_таблицы</u> .MYD	Файл данных.
<u>имя_таблицы</u> .MYI	Индексный файл.

Файлы, создаваемые каждым механизмом хранения, описаны в книге *MySQL. Руководство администратора* (М. : Издательский дом "Вильямс", 2005, ISBN 5-8459-0805-1).

Общая информация о свойствах различных типов столбцов представлена в главе 4. Информацию о пространственных типах можно найти в главе 7.

- Если для столбца не указано ни NULL, ни NOT NULL, принимается NULL.
- Столбцы целочисленных типов могут иметь атрибут AUTO\_INCREMENT. Когда вставляется значение NULL (рекомендуется) или 0 в индексированный столбец AUTO\_INCREMENT, ему присваивается следующее значение из числовой последовательности. Обычно это будет значение + 1, где значение — наибольшее значение этого столбца, имеющееся в таблице на этот момент. Числовые последовательности AUTO\_INCREMENT начинаются с 1.

Начиная с MySQL 4.1.1, указание флага NO\_AUTO\_VALUE\_ON\_ZERO для опции сервера --sql-mode либо для системной переменной sql\_mode позволяет сохранять 0 в столбце AUTO\_INCREMENT без генерации следующего значения последовательности.

#### На заметку!

В таблице может быть только один столбец AUTO\_INCREMENT, он должен быть проиндексирован и он не может иметь значения по умолчанию. Начиная с MySQL 3.23, столбец AUTO\_INCREMENT работает правильно, только если содержит положительные значения. Вставка отрицательно значения интерпретируется как вставка очень большого положительного. Это сделано для того, чтобы избежать проблем точности, когда число "заворачивается" из положительного в отрицательное, а также для того, чтобы гарантировать, что вы никогда случайно не получите значение 0 в столбце AUTO\_INCREMENT.

Для таблиц MyISAM и BDB можно определить дополнительный столбец AUTO\_INCREMENT в составном ключе.

Чтобы обеспечить совместимость MySQL с некоторыми ODBC-приложениями, вы можете получить последнее значение AUTO\_INCREMENT, вставленное в строку, с помощью следующего запроса:

```
SELECT * FROM имя_таблицы WHERE столбец_AUTO_INCREMENT IS NULL
```

- Начиная с MySQL 4.1, определения символьных столбцов могут содержать атрибут CHARACTER SET для указания набора символов для столбца и, необязательно, порядка сопоставления. Подробности см. в главе 3.

```
CREATE TABLE t (c CHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
```

Кроме того, MySQL 4.1 интерпретирует спецификацию длины символьных столбцов в символах (ранние версии интерпретировали ее в байтах).

- Значение NULL для столбцов типа TIMESTAMP обрабатывается иным образом, чем для других типов столбцов. Вы не можете вставлять литеральный NULL в столбцы TIMESTAMP. Присвоение значения NULL таким столбцам устанавливает их в текущую дату и время. Поскольку столбцы TIMESTAMP ведут себя подобным образом, атрибуты NULL и NOT NULL для этих столбцов игнорируются.

С другой стороны, чтобы облегчить клиентам MySQL применение столбцов типа TIMESTAMP, сервер сообщает, что этим столбцам можно присваивать значение NULL (что истинно), даже несмотря на то, что столбцы этого типа никогда не содержат такого значения. Это можно увидеть, воспользовавшись DESCRIBE имя\_таблицы для получения информации о структуре таблицы.

Отметим, что присвоение столбцу `TIMESTAMP` значения 0 – это не то же самое, что присвоение `NULL`, поскольку 0 является допустимым значением `TIMESTAMP`.

- Значение `DEFAULT` должно быть константой, оно не может быть функцией или выражением. Это означает, например, что нельзя установить в качестве значения по умолчанию для столбца типа даты функцию `NOW()` или `CURRENT_DATE`. Если никакого значения `DEFAULT` для столбца не указано, то MySQL автоматически присваивает свое, как описано ниже.

Если столбец допускает `NULL`, значением по умолчанию становится `NULL`.

Если столбец объявлен как `NOT NULL`, значение по умолчанию зависит от типа:

- Для числовых типов, кроме тех, что имеют атрибут `AUTO_INCREMENT`, устанавливается 0. Для столбцов `AUTO_INCREMENT` значение по умолчанию – это следующее значение в числовой последовательности.
- Для типов дат и времени, кроме `TIMESTAMP`, умолчанием будет соответствующее “нулевое” значение данного типа. Для первого столбца `TIMESTAMP` в таблице значением по умолчанию является текущая дата и время. См. раздел 4.3.
- Для строковых типов, кроме `ENUM`, значением по умолчанию является пустая строка. Для `ENUM` значение по умолчанию – первое в перечислении.

Столбцам типа `BLOB` и `TEXT` не может быть назначено значение по умолчанию.

- Комментарий для столбца может быть указан в опции `COMMENT`. Комментарий отображается операторами `SHOW CREATE TABLE` и `SHOW FULL COLUMNS`. Эта опция действительна, начиная с MySQL 4.1 (в более ранних версиях она разрешалась, но игнорировалась).
- Начиная с MySQL 4.1.0, атрибут `SERIAL` может использоваться в качестве псевдонима для `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`. Это средство обеспечения совместимости.
- Обычно `KEY` – синоним для `INDEX`. Начиная с MySQL 4.1, ключевой атрибут `PRIMARY KEY` также может быть указан для столбцов, как и просто `KEY`. Это было введено для совместимости с другими системами управления базами данных.
- В MySQL индекс `UNIQUE` – это такой индекс, в котором все значения ключа должны быть уникальными. При попытке вставить строку, ключ индекса которой повторяет уже существующий, генерируется ошибка. Исключением из этого правила являются столбцы, входящие в ключ индекса и допускающие значения `NULL`. Это исключение не касается таблиц `BDB`, в которых уникально индексированный столбец может содержать только один `NULL`.
- Первичный ключ (`PRIMARY KEY`) – это уникальный `KEY`, в котором все ключевые столбцы определены как `NOT NULL`. Если они не объявлены явно как `NOT NULL`, MySQL сделает это неявно (и молча). Если у вас нет первичного ключа в таблице, а приложение его требует, MySQL возвращает первый `UNIQUE`-индекс, который не содержит в ключей `NULL`-столбцов, в качестве `PRIMARY KEY`.
- В созданной таблице `PRIMARY KEY` размещается первым, за ним следуют все уникальные индексы, затем неуникальные индексы. Это помогает оптимизатору MySQL правильно расставлять приоритеты в выборе индексов и быстрее обнаруживать дублирующие уникальные ключи.

- PRIMARY KEY может быть индексом с составным ключом. Однако вы не можете создать составной индекс, используя атрибут PRIMARY KEY в спецификации столбца. Если это сделать, только первый столбец будет отмечен как первичный ключ. Вы должны использовать отдельную конструкцию PRIMARY KEY (имя\_столбца\_индекса, ...).
- Если PRIMARY KEY или UNIQUE-индекс построены по одному столбцу целочисленного типа, вы также можете ссылаться на этот столбец в операторах SELECT как на \_rowid (введено в MySQL 3.23.11).
- Начиная с MySQL 4.1.0, некоторые механизмы хранения позволяют специфицировать тип индекса при его создании. Синтаксис спецификации типа индекса выглядит как USING имя\_типа. Допустимые значения имя\_типа, которые поддерживаются различными механизмами хранения, показаны ниже. Там, где перечислено несколько типов индексов, по умолчанию (при отсутствии спецификации типа) является первый из них:

Механизм хранения	Допустимые типы индексов
MyISAM	BTREE
InnoDB	BTREE
MEMORY/HEAP	HASH, BTREE

Пример:

```
CREATE TABLE lookup
  (id INT, INDEX USING BTREE (id))
ENGINE = MEMORY;
```

При спецификации типа индекса в качестве синонима USING имя\_типа можно использовать TYPE имя\_типа. Однако USING – предпочтительная форма. Кроме того, если спецификация индекса определена словом TYPE, то имя индекса в этом случае обязательно, так как в отличие от USING, слово TYPE не является зарезервированным словом, и при отсутствии имени индекса будет интерпретироваться как это имя.

Если вы указываете недопустимый для данного механизма хранения тип индекса, но существует другой допустимый тип, который механизм может использовать без влияния на результаты запросов, механизм использует этот тип.

- Только механизмы хранения MyISAM, InnoDB, BDB и (начиная с MySQL 4.0.2) MEMORY поддерживают индексы по столбцам, которые могут содержать значения NULL. В остальных случаях вы должны декларировать столбцы, входящие в ключ индекса, как NOT NULL, иначе возникнет ошибка.
- Используя синтаксис col\_name (длина) в спецификации индекса, вы можете создать индекс, который в ключе использует только первые длина символов значений столбца CHAR или VARCHAR. Индексирование только префикса значений столбца может существенно уменьшить размер индексного файла.

Механизм хранения MyISAM и (начиная с MySQL 4.0.14) InnoDB также поддерживают индексацию столбцов TEXT и BLOB. При индексации столбцов TEXT и BLOB длину индексируемого префикса указывать обязательно, например:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

Префиксы могут быть до 255 символов длиной (или 1000 символов для таблиц MyISAM и InnoDB в MySQL версии 4.1.2 и выше). Следует отметить, что максимальная длина индексируемого префикса измеряется в байтах, в то время как длина префикса в спецификации CREATE TABLE интерпретируется как количество символов. Об этом необходимо помнить, когда индексируемый столбец использует многобайтный набор символов.

- Спецификация *имя\_столбца\_индекса* может завершаться словом ASC или DESC. Эти слова предусмотрены для будущих расширений, чтобы указывать порядок хранения индексируемых значений – по возрастанию или по убыванию. В настоящее время они распознаются, но игнорируются. Индексные значения всегда хранятся в возрастающем порядке.
- При использовании конструкций ORDER BY или GROUP BY для столбцов BLOB или TEXT, сервер сортирует значения, используя только определенное число начальных байтов, которое задается системной переменной *max\_sort\_length*. См. раздел 4.4.2.
- В MySQL 3.23.23 и более поздних версиях есть возможность создавать индексы FULLTEXT. Они используются для полнотекстового поиска. Индексы FULLTEXT поддерживают только таблицы MyISAM. Они могут быть созданы только на столбцах CHAR, VARCHAR или TEXT. Индексация при этом выполняется по полной длине значения столбца. Частичная индексация префикса не поддерживается, и если указать длину префикса, она игнорируется. См. детали в разделе 5.6.
- В MySQL 4.1 и выше можно создавать индексы SPATIAL по столбцам пространственных (spatial) типов. Пространственные типы поддерживаются только механизмом хранения MyISAM и индексируемые столбцы должны быть объявлены как NOT NULL. См. главу 7.
- В MySQL 3.23.44 и выше таблицы InnoDB поддерживают проверку ограничений целостности внешних ключей. Отметим, что синтаксис FOREIGN KEY для InnoDB более ограничивающий, чем синтаксис оператора CREATE TABLE, представленный в начале раздела. Столбцы таблиц, на которые ссылается внешний ключ, должны быть всегда именованы явно. InnoDB поддерживает действия ON DELETE и ON UPDATE для внешних ключей, начиная с версии MySQL 3.23.50 и MySQL 4.0.8 соответственно. Точный синтаксис представлен в разделе 6.2.5.1.

Для других механизмов хранения сервер MySQL разбирает синтаксис FOREIGN KEY и REFERENCES в операторе CREATE TABLE, но без каких-либо действий. Конструкция CHECK распознается, однако игнорируется всеми механизмами хранения.

- Для таблиц MyISAM и ISAM каждый NULL-столбец требует дополнительного бита, который округляется до ближайшего байта. Максимальная длина в байтах может быть рассчитана следующим образом:

$$\begin{aligned} \text{длина строки} &= 1 \\ &+ (\text{сумма длин столбцов}) \\ &+ (\text{количество NULL-столбцов} + \text{флаг\_удаления} + 7) / 8 \\ &+ (\text{количество столбцов переменной длины}) \end{aligned}$$

*флаг\_удаления* равен 1 для таблиц со статическим форматом записи. Статические таблицы используют бит в записи строки в качестве признака того, что запись удалена. *флаг\_удаления* равен 0 для динамических таблиц, потому что флаг сохраняется в динамической заголовке строки.

Эти вычисления неприменимы к таблицам InnoDB, в которых размер хранения не отличается для столбцов NULL и NOT NULL.

Часть опции `таблицы` синтаксиса `CREATE TABLE` может применяться в MySQL 3.23 и выше.

Опции `ENGINE` и `TYPE` специфицируют механизм хранения для таблицы. Опция `ENGINE` появилась в MySQL 4.0.18 (для серии 4.0) и в 4.1.2 (для серии 4.1). Это предпочтительный вариант наименования опции в этих версиях, а `TYPE` считается устаревшей. `TYPE` еще поддерживается в серии выпусков 4.x, но вероятно, будет исключена в MySQL 5.1.

Опции `ENGINE` и `TYPE` принимают следующие значения:

Механизм хранения	Описание
BDB	Таблицы, безопасные к транзакциям, со страничной блокировкой.
BerkeleyDB	Псевдоним для BDB.
HEAP	Данные таблицы хранятся только в памяти.
ISAM	Исходный механизм хранения MySQL.
InnoDB	Таблицы, безопасные к транзакциям, с построчной блокировкой и внешними ключами.
MEMORY	Псевдоним для HEAP. (MEMORY – предпочтительная форма, начиная с MySQL 4.1)
MERGE	Коллекция таблиц MyISAM, используемых как одна таблица.
MRG_MyISAM	Псевдоним для MERGE.
MyISAM	Бинарный переносимый механизм хранения; усовершенствованная замена ISAM.

Если указывается несуществующий механизм хранения, вместо него MySQL использует MyISAM. Например, если определение таблицы включает опцию `ENGINE=BDB`, а сервер MySQL не поддерживает таблиц BDB, то таблица создается как MyISAM. Это дает возможность настроить среду репликации, в которой применяются транзакционные таблицы на главном сервере и нетранзакционные – на подчиненном (чтобы получить более высокую скорость). В MySQL 4.1.1 генерируется предупреждение, если заданный механизм хранения не поддерживается.

Характеристики механизмов хранения подробно обсуждаются в книге *MySQL. Руководство администратора* (М.: Издательский дом “Вильямс”, 2005, ISBN 5-8459-0805-1).

Другие опции таблиц служат для оптимизации поведения таблицы. В большинстве случаев вам не придется указывать ни одну из них. Опции работают со всеми механизмами хранения, если не указано другое:

- **AUTO\_INCREMENT.** Начальное значение `AUTO_INCREMENT` для таблицы. Это работает только с таблицами MyISAM. Для того чтобы установить первое значение последовательности для таблицы InnoDB, необходимо вставить строку со значением соответствующей столбцы, на единицу меньшим, чем желательное начальное значение, а затем удалить ее.
- **AVG\_ROW\_LENGTH.** Предполагаемое среднее значение длины строки в таблице. Это необходимо устанавливать только для больших таблиц с записями переменной длины.



Когда создается таблица `MyISAM`, `MySQL` использует произведение значений опций `MAX_ROWS` на `AVG_ROW_LENGTH`, чтобы рассчитать предполагаемый размер таблицы. Если не указана любая из этих опций, максимальный размер таблицы составит 4 Гбайт (или 2 Гбайт, если операционная система поддерживает файлы размером только 2 Гбайт). Смысл этого состоит в уменьшении размера указателя записей, чтобы сделать индексы меньшими и быстрыми в случае, когда нет необходимости в большой таблице. Если вы хотите всем таблицам обеспечить возможность роста за пределами ограничения в 4 Гбайт, и согласны с тем, что даже маленькие таблицы будут занимать несколько больше места и работать медленнее, чем необходимо, можете увеличить размер указателя записей по умолчанию, установив системную переменную `myisam_data_pointer_size`, которая была добавлена в `MySQL 4.1.2`.

- **CHECKSUM.** Установите значение этой опции равным 1, если хотите, чтобы `MySQL` поддерживал живую контрольную сумму всех строк таблицы (то есть контрольную сумму, которую `MySQL` обновляет автоматически при каждом изменении таблицы). Это немного замедлит обновления данных в таблице, но облегчит поиск поврежденных таблиц. Оператор `CHECKSUM TABLE` сообщает эту контрольную сумму (только для `MyISAM`).
- **COMMENT.** Комментарий к таблице, не более 60 символов длиной.
- **MAX\_ROWS.** Максимальное количество строк, которое планируется хранить в таблице.
- **MIN\_ROWS.** Минимальное количество строк, которое планируется хранить в таблице.
- **PACK\_KEYS.** Эту опцию нужно установить в 1, если необходимо сделать индексы поменьше. Обычно это немного замедляет запись, однако ускоряет чтение. Установка этой опции в 0 отключает упаковку ключей. Установка ее в `DEFAULT` (`MySQL 4.0`) сообщает механизму хранения, что паковать нужно только длинные столбцы типа `CHAR/VARCHAR` (только для `ISAM` и `MyISAM`).

Если не указывать опцию `PACK_KEYS`, по умолчанию пакуются только строки, но не числа. Если указать `PACK_KEYS=1`, то числа тоже пакуются.

При упаковке двоичных числовых ключей `MySQL` применяет сжатие префиксов:

- Каждый ключ требует дополнительного байта для указания того, сколько байт предыдущего ключа повторяются в следующем.
- Указатель строки сохраняется в порядке “старший байт первый” непосредственно после ключа, чтобы улучшить сжатие.

Это означает, что если у вас много равных ключей в двух последовательных строках, все последующие “такие же” ключи обычно займут только два байта (включая указатель на строку).

Сравните это с обычным случаем, когда каждый следующий ключ потребует для хранения `размер_хранения_для_ключа + размер_указателя` (размер указателя обычно равен 4). Однако выгода от сжатия префиксов будет ощутимой только в том случае, когда есть много равных значений числового ключа. Когда все ключи абсолютно разные, то на каждый ключ потребуются на один байт больше, если ключ не может иметь значений `NULL`. (В этом случае длина упакованного ключа будет сохранена в том же байте, который служит для хранения признака того, что ключ `NULL`).

- **PASSWORD.** Шифрует файл `.frm` с использованием пароля. В стандартной версии MySQL эта опция не делает ничего.
- **DELAY\_KEY\_WRITE.** Эту опцию нужно установить в 1, если вы хотите отложить обновления ключей таблицы до момента ее закрытия (только для MyISAM).
- **ROW\_FORMAT.** Определяет, как должна сохраняться строка. В настоящее время эта опция работает только с таблицами MyISAM. Ее значением может быть `FIXED` или `DYNAMIC`, соответственно, для статического формата строки или формата с переменной длиной. `myisampack` устанавливает тип `COMPRESSED`.

- **RAID\_TYPE.** Опция `RAID_TYPE` помогает преодолеть лимит размера файла данных MyISAM в 2 Гбайт/4 Гбайт (не касается индексных файлов) в операционных системах, которые не поддерживают большие файлы.

Вы можете повысить скорость операций ввода-вывода, поместив RAID-каталоги на разные физические диски. На данный момент допускается только `RAID_TYPE` и `STRIPPED`. 1 и `RAID0` – это псевдонимы для `STRIPPED`.

Если вы указываете опцию `RAID_TYPE` для таблицы MyISAM, указывайте также `RAID_CHUNKS` и `RAID_CHUNKSIZE`. Максимальное значение `RAID_CHUNKS` равно 255. MyISAM создаст подкаталоги `RAID_CHUNKS` с именами 00, 01, 02, ... 09, 0a, 0b, ... в каталоге базы данных. В каждом из этих подкаталогов MyISAM создаст файл *имя\_таблицы*.MYD. При записи в файл данных обработчик RAID отображает первые `RAID_CHUNKSIZE*1024` байт на первый файл, следующие `RAID_CHUNKSIZE*1024` байт на второй файл и так далее.

Опция `RAID_TYPE` работает под управлением любой операционной системы, если MySQL был собран с помощью `configure --with-raid`. Чтобы определить, поддерживает ли сервер RAID-таблицы, выполните `SHOW VARIABLES LIKE 'have_raid'` и посмотрите, установлено ли значение переменной `have_raid` в `YES`.

- **UNION.** Применяется, когда нужно использовать коллекцию идентичных таблиц как одну. Это работает только с таблицами `MERGE`.

На данный момент вы должны иметь привилегии `SELECT`, `UPDATE` и `DELETE` на таблицы, отображаемые в `MERGE`. Первоначально все используемые таблицы должны были располагаться в том же каталоге, что и таблица `MERGE`. В MySQL 4.1.1 это ограничение снято.

- **INSERT\_METHOD.** Если необходимо вставлять данные в таблицу `MERGE`, вы должны указывать с помощью опции `INSERT_METHOD`, в какую физическую таблицу должна вставляться строка. `INSERT_METHOD` – опция, применимая только к таблицам `MERGE`. Она была представлена впервые в MySQL 4.0.0.
- **DATA DIRECTORY, INDEX DIRECTORY**

Используя `DATA DIRECTORY='каталог'` или `INDEX DIRECTORY='каталог'`, вы можете указать, где механизм хранения MyISAM должен размещать файлы данных и индексов. Имейте в виду, что нужно указывать полный путь, а не относительный.

Эти опции работают только с таблицами MyISAM, начиная с MySQL 4.0 и выше, когда вы не применяете опцию `--skip-symbolic-links`. Ваша операционная система также должна иметь работающий, безопасный в отношении потоков системный вызов `realpath()`.

Начиная с MySQL 3.23, вы можете создавать одну таблицу из другой, добавляя **SELECT** в конец оператора **CREATE TABLE**:

```
CREATE TABLE имя_новой_таблицы SELECT * FROM имя_исходной_таблицы;
```

MySQL создаст новые столбцы для всех элементов **SELECT**. Например:

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
->   PRIMARY KEY (a), KEY(b))
->   TYPE=MyISAM SELECT b,c FROM test2;
```

Здесь создается таблица **MyISAM** с тремя столбцами, **a**, **b** и **c**. Следует отметить, что столбцы из оператора **SELECT** добавляются в правую часть таблицы, не перемешиваясь с имеющимися. Рассмотрим следующий пример:

```
mysql> SELECT * FROM foo;
+----+
| n |
+----+
| 1 |
+----+

mysql> CREATE TABLE bar (m INT) SELECT n FROM foo;
Query OK, 1 row affected (0.02 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM bar;
+-----+----+
| m | n |
+-----+----+
| NULL | 1 |
+-----+----+
1 row in set (0.00 sec)
```

Для каждой строки таблицы **foo** вставляется строка в **bar**, со значениями, выбранными из **foo**, и значениями по умолчанию для новых столбцов.

Если при этом происходит какая-либо ошибка, новая таблица не создается.

**CREATE TABLE...SELECT** не создает автоматически какие-либо индексы. Так сделано намерено, чтобы обеспечить максимально возможную гибкость данного оператора. Если вы хотите иметь индексы в создаваемой таблице, то должны будете указать следующее перед оператором **SELECT**:

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

При этом могут происходить некоторые преобразования типов столбцов. Например, не сохраняется атрибут **AUTO\_INCREMENT**, и столбцы **VARCHAR** меняют тип на **CHAR**.

При создании таблицы оператором **CREATE TABLE...SELECT** убедитесь, что все вызовы функций и выражения, используемые в запросе, снабжены псевдонимами. Если этого не сделать, то оператор **CREATE** завершится аварийно, или же вы получите нежелательные имена столбцов в новой таблице.

```
CREATE TABLE artists_and_works
SELECT artist.name, COUNT(work.artist_id) AS number_of_works
FROM artist LEFT JOIN work ON artist.id = work.artist_id
GROUP BY artist.id;
```

Начиная с MySQL 4.1, типы сгенерированных столбцов можно задавать явно:

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

В MySQL 4.1 вы также можете применить LIKE для создания пустой таблицы на основе определения другой таблицы, включая любые атрибуты столбцов и индексы, которые имеются в исходной таблице:

```
CREATE TABLE имя_новой_таблицы LIKE имя_исходной_таблицы;
```

CREATE TABLE...LIKE не копирует опции DATA DIRECTORY или INDEX DIRECTORY, которые были специфицированы в исходной таблице.

Вы можете предварить SELECT словами IGNORE или REPLACE, чтобы указать, как нужно обрабатывать записи с дублированными значениями уникальных ключей. Если указано IGNORE, новые записи с дублированными уникальными ключами отбрасываются. Если же указано REPLACE, новые записи заменяют существующие с такими же значениями уникальных ключей. Если не указано ни IGNORE, ни REPLACE, то появление записей с дублированными значениями ключей приводит к ошибке.

Чтобы гарантировать, что журнал обновления/бинарный журнал могут быть использованы для пересоздания исходных таблиц, MySQL не разрешает параллельные вставки во время работы CREATE TABLE...SELECT.

### 6.2.5.1. Создание внешних ключей

Начиная с MySQL 3.23.44, механизм InnoDB оснащен ограничениями внешних ключей. Синтаксис определения внешнего ключа InnoDB выглядит так:

```
[CONSTRAINT символ] FOREIGN KEY [идентификатор] (имя_столбца_индекса, ...)
REFERENCES имя_таблицы (имя_столбца_индекса, ...)
[ON DELETE {CASCADE | SET NULL | NO ACTION | RESTRICT | SET DEFAULT}]
[ON UPDATE {CASCADE | SET NULL | NO ACTION | RESTRICT | SET DEFAULT}]
```

Обе таблицы должны иметь тип InnoDB. В ссылающейся таблице должен быть индекс, у которого столбец внешнего ключа является первым в списке ключевых столбцов. У таблицы, на которую осуществляется ссылка, должен быть индекс, в ключе которого столбец, на который осуществляется ссылка, указан первым. Индексы на префиксах столбцов для внешних ключей не поддерживаются.

InnoDB требует индексов на внешних и ссылаемых ключах, чтобы проверка ограничений внешнего ключа выполнялась быстрее и не требовала сканирования таблиц.

Начиная с MySQL 4.1.2, эти индексы создаются автоматически. В более старых версиях индексы должны были создаваться явно, в противном случае попытки создания внешних ключей завершались ошибкой.

Связанные столбцы во внешнем ключе и в ключе, на который они ссылаются, должны иметь одинаковый внутренний тип данных InnoDB, чтобы их можно было сравнивать без преобразования типов. Размер и наличие знака столбцов целочисленных типов должны быть одинаковыми. Длины строковых типов не обязательно должны быть равны. Если вы выполняете действие SET NULL, убедитесь, что столбец дочерней таблицы не объявлен как NOT NULL.

Если MySQL сообщает об ошибке 1005 при попытке выполнить CREATE TABLE, и сообщение об ошибке ссылается на номер errno 150, это означает, что создание таблицы завершилось неудачей, потому что ограничение внешнего ключа не было корректно сформировано. Подобным же образом, если ALTER TABLE завершается аварийно и возвращает ссылку на номер errno 150, это означает, что определение внешнего ключа неверно сформировано для изменяемой таблицы. Начиная с MySQL 4.0.13, вы можете ис-

пользовать `SHOW INNODB STATUS`, чтобы отобразить детальное объяснение последней произошедшей на сервере ошибки, связанной с внешним ключом.

Начиная с MySQL 3.23.50, InnoDB не проверяет ссылочную целостность внешних ключей или ключей, на которые осуществляется ссылка, если их столбцы содержат значение NULL.

**Отклонение от стандарта SQL.** Если в родительской таблице есть несколько строк, имеющих одинаковые значения ключей, на которые ссылается внешний ключ, InnoDB при проверке ограничений внешнего ключа поступает так, будто родительских строк с одинаковыми значениями ключей не существует. Например, если вы определили ограничение типа `RESTRICT`, и существуют дочерние записи с несколькими родительскими записями, InnoDB не позволяет удалять ни одну из этих родительских записей.

Начиная с MySQL 3.23.50, вы также можете ассоциировать конструкции `ON DELETE CASCADE` или `ON DELETE SET NULL` с ограничением внешнего ключа. Соответствующая опция `ON UPDATE` доступна, начиная с версии 4.0.8. Если указано `ON DELETE CASCADE` и удаляется строка в родительской таблице, то дочерние записи, у которых значение внешнего ключа совпадает со значением ссылаемого ключа в родительской записи, также автоматически удаляются. Если указано `ON DELETE SET NULL` и удаляется строка в родительской таблице, то дочерние записи автоматически обновляются таким образом, что столбцы внешнего ключа принимают значение NULL.

`SET DEFAULT` распознается, но игнорируется.

InnoDB выполняет каскадные операции по алгоритму поиска в глубину, на основе записей в индексах, соответствующих ограничениям внешних ключей.

**Отклонение от стандарта SQL.** Если `ON UPDATE CASCADE` или `ON UPDATE SET NULL` рекурсивно обновляют *ту же таблицу*, которая уже обновляется каскадным способом, это работает как при `RESTRICT`. Это означает, что вы не можете иметь операции `ON UPDATE CASCADE` или `ON UPDATE SET NULL`, ссылающиеся сами на себя. Так сделано для предотвращения бесконечного заикливания каскадных обновлений. С другой стороны, ссылающиеся на себя `ON DELETE SET NULL` допускаются, начиная с версии 4.0.13. Ссылающиеся на себя `ON DELETE CASCADE` возможны с тех пор, как реализовано `ON DELETE`.

Простой пример, в котором связываются родительская и дочерняя таблицы через одностолбцовый внешний ключ:

```
CREATE TABLE parent(id INT NOT NULL,  
                     PRIMARY KEY (id)  
) TYPE=INNODB;  
CREATE TABLE child(id INT, parent_id INT,  
                    INDEX par_ind (parent_id),  
                    FOREIGN KEY (parent_id) REFERENCES parent(id)  
                    ON DELETE CASCADE  
) TYPE=INNODB;
```

Более сложный пример, в котором таблица `product_order` имеет внешний ключ на две другие таблицы. Один из них ссылается на двухстолбцовый индекс в таблице `product`, а другой – на одностолбцовый в таблице `customer`:

```
CREATE TABLE product (category INT NOT NULL, id INT NOT NULL,  
                       price DECIMAL,  
                       PRIMARY KEY(category, id)) TYPE=INNODB;
```

```
CREATE TABLE customer (id INT NOT NULL,
                        PRIMARY KEY (id)) TYPE=INNODB;
CREATE TABLE product_order (no INT NOT NULL AUTO_INCREMENT,
                             product_category INT NOT NULL,
                             product_id INT NOT NULL,
                             customer_id INT NOT NULL,
                             PRIMARY KEY(no),
                             INDEX (product_category, product_id),
                             FOREIGN KEY (product_category, product_id)
                             REFERENCES product(category, id)
                             ON UPDATE CASCADE ON DELETE RESTRICT,
                             INDEX (customer_id),
                             FOREIGN KEY (customer_id)
                             REFERENCES customer(id)) TYPE=INNODB;
```

Начиная с MySQL 3.23.50, анализатор выражений InnoDB позволяет использовать обратные кавычки вокруг имен таблиц и столбцов в конструкциях FOREIGN KEY... REFERENCES... Начиная с MySQL 4.0.5, анализатор выражений InnoDB принимает во внимание установку системной переменной `lower_case_table_names`.

### 6.2.5.2. Неявные изменения спецификаций столбцов

В некоторых случаях MySQL без предупреждений изменяет спецификации столбцов по сравнению с теми, что были заданы оператором CREATE TABLE или ALTER TABLE:

- Столбцы VARCHAR длиной менее четырех символов получают тип CHAR.
- Если любой из столбцов таблицы имеет переменную длину, вся строка в результате имеет переменную длину. Поэтому, если таблица содержит любой столбец с типом переменной длины (VARCHAR, TEXT или BLOB), все столбцы CHAR длиной более трех символов становятся VARCHAR. Это не влияет на то, как столбцы будут использоваться. В MySQL тип VARCHAR – это просто другой способ хранения символов. MySQL выполняет это преобразование, потому что это позволяет сэкономить место на диске и ускоряет операции работы с таблицей.
- Начиная с MySQL 4.1.0, столбцы типа CHAR и VARCHAR длиной более 255 символов преобразуются к минимальному типу TEXT, который может вместить заданное количество символов. Например, VARCHAR (500) преобразуется в TEXT, а VARCHAR (200000) – в MEDIUMTEXT. Это средство обеспечения совместимости.
- Размер отображения типа TIMESTAMP исключен из MySQL 4.1 из-за изменений, которые претерпел тип столбцов TIMESTAMP в этой версии. До MySQL 4.1 размер отображения TIMESTAMP должен был быть в пределах от 2 до 14. Если указать размер отображения 0 или более 14, он принимался равным 14. Нечетные величины размера от 1 до 13 округлялись до ближайшего большего четного.
- Невозможно присвоить литеральное значение NULL столбцу TIMESTAMP. Присвоение NULL устанавливает его в текущую дату и время. Поскольку столбцы TIMESTAMP ведут себя подобным образом, атрибуты NULL и NOT NULL неприменимы к ним в привычном смысле и потому игнорируются. DESCRIBE имя\_таблицы всегда сообщает, что столбцы TIMESTAMP могут принимать значения NULL.
- Столбцы, входящие в определение первичного ключа становятся NOT NULL независимо от того, задано ли это явным образом.

- Начиная с MySQL 3.23.51, завершающие пробелы автоматически удаляются из значений-членов ENUM и SET при создании таблицы.
- MySQL отображает некоторые типы столбцов, используемые другими поставщиками SQL-серверов, в типы MySQL. См. раздел 4.7.
- Если вы включите конструкцию USING, чтобы специфицировать тип индекса, который не допустим для данного механизма хранения, но при этом существует другой тип индексов, который может быть использован без влияния на исполнение запросов, то механизм хранения использует этот тип.

Чтобы увидеть, изменил ли MySQL тип столбцов по сравнению с заданным вами, воспользуйтесь оператором DESCRIBE или SHOW CREATE TABLE после создания или изменения таблиц.

Если вы сжимаете таблицу с помощью myisampack, также могут произойти и другие изменения типов столбцов.

## 6.2.6. Синтаксис DROP DATABASE

`DROP DATABASE [IF EXISTS] имя_базы_данных`

DROP DATABASE уничтожает все таблицы в базе данных и удаляет базу данных. Будьте *очень* внимательны при обращении с этим оператором! Для использования DROP DATABASE необходимо иметь привилегию DROP в базе данных.

В MySQL 3.22 и более поздних версиях можно использовать ключевые слова IF EXISTS, чтобы предотвратить ошибку, связанную с попыткой удаления несуществующей базы данных.

Если вы применяете DROP DATABASE к базе, доступ к которой осуществляется через символические ссылки, удаляется как сама база, так и символические ссылки.

Начиная с MySQL 4.1.2, DROP DATABASE возвращает количество удаленных таблиц. Это соответствует количеству удаленных файлов .frm.

Оператор DROP DATABASE удаляет из заданного каталога базы данных те файлы и подкаталоги, которые MySQL может создать при нормальной операции:

- Все файлы со следующими расширениями:

.BAK	.DAT	.HSH	.ISD
.ISM	.ISM	.MRG	.MYD
.MYI	.db	.frm	

- Все подкаталоги с именами, включающими шестнадцатеричные числа 00-ff. Это подкаталоги, используемые для RAID-таблиц.
- Файл db.opt, если он есть.

Если в каталоге базы данных останутся какие-то другие файлы, кроме перечисленных, MySQL не сможет удалить каталог базы. В этом случае вы должны вручную удалить все оставшиеся там файлы и подкаталоги и повторить оператор DROP DATABASE снова.

## 6.2.7. Синтаксис DROP INDEX

`DROP INDEX имя_индекса ON имя_таблицы`

DROP INDEX уничтожает индекс с именем *имя\_индекса*, принадлежащий таблице *имя\_таблицы*. В MySQL 3.22 и выше DROP INDEX отображается на ALTER TABLE, чтобы удалить индекс. См. раздел 6.2.2. До MySQL 3.22 DROP INDEX ничего не делал.

## 6.2.8. Синтаксис DROP TABLE

```
DROP {TEMPORARY} TABLE [IF EXISTS]
    имя_таблицы [, имя_таблицы] ...
    [RESTRICT | CASCADE]
```

DROP TABLE удаляет одну или более таблиц. Вы должны иметь привилегию DROP для каждой из этих таблиц. Этим оператором удаляются все данные таблицы, а также ее определение, поэтому *будьте осторожны* с ним!

В MySQL 3.22 и более поздних версиях вы можете использовать ключевые слова IF EXISTS, чтобы предотвратить ошибку, вызванную попыткой удалить несуществующую таблицу. Начиная с MySQL 4.1, генерируется предупреждение о каждой несуществующей таблице, если указано IF EXISTS. См. раздел 6.5.3.20.

Для облегчения переносимости допускаются слова RESTRICT и CASCADE. Однако на сегодняшний день они ничего не делают.

### На заметку!

DROP TABLE автоматически завершает текущую активную транзакцию, если только вы не используете MySQL 4.1 и выше, а также ключевое слово TEMPORARY.

Ключевое слово TEMPORARY игнорируется в MySQL 4.0. В версии MySQL 4.1 оно дает следующий эффект:

- Оператор удаляет любые временные таблицы.
- Оператор не завершает текущую транзакцию.
- Никакие права доступа не проверяются (временные таблицы видимы только клиенту, который их создал, поэтому никакие проверки не требуются).

Применение слова TEMPORARY – хороший способ гарантировать, что вы не удалите случайно никаких постоянных (не временных) таблиц.

## 6.2.9. Синтаксис RENAME TABLE

```
RENAME TABLE имя_таблицы TO новое_имя_таблицы
    [, имя_таблицы2 TO новое_имя_таблицы2] ...
```

Оператор переименовывает одну или несколько таблиц. Был добавлен в MySQL 3.23.23.

Операция переименования является атомарной, а это означает, что ни один другой поток сервера не может получить доступ к таблице во время выполнения переименования. Например, если у вас есть существующая таблица old\_table, вы можете создать новую пустую таблицу new\_table, которая имеет ту же структуру, а затем заменить старую таблицу на новую пустую следующим образом:

```
CREATE TABLE new_table (...);
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

Если оператор переименовывает более одной таблицы, операции переименования выполняются слева направо. Если вы хотите, чтобы две таблицы обменялись именами, это можно сделать примерно так (предполагается, что таблица tmp\_table не существует):

```
RENAME TABLE old_table TO tmp_table,
    new_table TO old_table,
    tmp_table TO new_table;
```



До тех пор пока две базы данных находятся в одной файловой системе, вы также можете переименовать таблицу, чтобы переместить ее из одной базы в другую:

```
RENAME TABLE текущая_база_данных.имя_таблицы  
TO другая_база_данных.имя_таблицы;
```

Когда выполняется RENAME, нельзя иметь никаких заблокированных таблиц или активных транзакций. Вы должны также обладать привилегиями ALTER и DROP на исходной таблице, а также привилегиями CREATE и INSERT на новой таблице.

## 6.3. Служебные операторы MySQL

### 6.3.1. Синтаксис DESCRIBE

(получить информацию о столбцах)

```
{DESCRIBE | DESC} имя_таблицы [имя_столбца [wild]]
```

DESCRIBE представляет информацию о столбцах таблицы. Это сокращение для SHOW COLUMNS FROM.

См. раздел 6.5.3.4.

имя\_столбца может быть именем столбца или строкой, содержащей символы SQL-шаблонов '%' и '\_', чтобы получить информацию только о столбцах, имена которых соответствуют шаблону. Нет необходимости заключать строку в кавычки, если только она не содержит пробелов или других специальных символов.

Если типы столбцов отличаются от ожидаемых, которые указывались в операторе CREATE TABLE, помните, что MySQL иногда подменяет типы столбцов. См. раздел 6.2.5.2.

Оператор DESCRIBE введен для совместимости с Oracle.

Операторы SHOW CREATE TABLE и SHOW TABLE STATUS также предоставляют информацию о таблицах. См. раздел 6.5.3.

### 6.3.2. Синтаксис USE

```
USE имя_базы_данных
```

Оператор USE имя\_базы\_данных сообщает MySQL о том, что базу с именем имя\_базы\_данных нужно использовать по умолчанию (как текущую) в последующих операциях. База данных остается базой по умолчанию до завершения сеанса или до момента, когда будет выполнен другой оператор USE:

```
mysql> USE db1;  
mysql> SELECT COUNT(*) FROM mytable; # выборка из db1.mytable  
mysql> USE db2;  
mysql> SELECT COUNT(*) FROM mytable; # выборка из db2.mytable
```

Назначение отдельной базы данных как используемой по умолчанию не запрещает доступ к другим базам. Следующий пример обращается к таблице author в базе db1 и к таблице editor в базе db2:

```
mysql> USE db1;  
mysql> SELECT author_name, editor_name FROM author, db2.editor  
-> WHERE author.editor_id = db2.editor.editor_id;
```

Оператор USE введен для обеспечения совместимости с Sybase.

## 6.4. Операторы управления транзакциями и блокировкой MySQL

### 6.4.1. Синтаксис START TRANSACTION, COMMIT и ROLLBACK

По умолчанию MySQL работает в режиме автоматического завершения транзакций (autocommit). Это означает, что как только выполняется оператор обновления данных, который модифицирует таблицу, MySQL тут же сохраняет изменения на диске.

Если вы работаете с таблицами, безопасными в отношении транзакций (такими как InnoDB и BDB), то режим автоматического завершения транзакций можно отключить следующим оператором:

```
SET AUTOCOMMIT=0;
```

После отключения режима автоматического завершения транзакций вы должны использовать оператор COMMIT, чтобы сохранять изменения на диске, либо ROLLBACK, чтобы отменять изменения, выполненные с момента начала транзакции.

Если вы хотите отключить режим автоматического завершения транзакций только для отдельной последовательности операторов, можете воспользоваться оператором START TRANSACTION:

```
START TRANSACTION;  
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;  
UPDATE table2 SET summary=@A WHERE type=1;  
COMMIT;
```

После START TRANSACTION режим автоматического завершения транзакций остается выключенным до явного завершения транзакции с помощью COMMIT или отката посредством ROLLBACK. Затем режим автоматического завершения возвращается в свое предыдущее состояние.

Вместо START TRANSACTION для обозначения начала транзакции могут использоваться BEGIN и BEGIN WORK. Оператор START TRANSACTION появился в версии MySQL 4.0.11. Это стандартный синтаксис SQL и рекомендуемый способ начинать транзакции. BEGIN и BEGIN WORK доступны, начиная с MySQL 3.23.17 и MySQL 3.23.19, соответственно.

Следует отметить, что если вы не используете таблицы, безопасные к транзакциям, все изменения сохраняются немедленно, независимо от режима автоматического завершения транзакций.

Если вы дадите ROLLBACK после обновления таблицы, безопасной к транзакциям, в пределах транзакции, выдается предупреждение ER\_WARNING\_NOT\_COMPLETE\_ROLLBACK. Изменения в таблицах, безопасных к транзакциям, будут отменены, а в небезопасных — останутся.

Если вы используете START TRANSACTION или SET AUTOCOMMIT=0, то должны для резервных копий использовать бинарный журнал MySQL, вместо старого журнала обновлений. Транзакции сохраняются в бинарном журнале одним куском, до COMMIT. Транзакции, отмененные оператором ROLLBACK, в журнал не заносятся. (Существует одно исключение: модификации нетранзакционных таблиц не могут быть отменены. Если отмененная транзакция включала в себя модификацию данных нетранзакционных таблиц, такая транзакция целиком записывается в бинарный журнал с оператором ROLLBACK

в конце, чтобы гарантировать, что модификации этих таблиц будут реплицированы. Это верно, начиная с версии MySQL 4.0.15.)

Вы можете изменить уровень изоляции транзакций оператором `SET TRANSACTION ISOLATION LEVEL`. См. раздел 6.4.6.

## 6.4.2. Операторы, которые нельзя откатить

Для некоторых операторов нельзя выполнить откат с помощью `ROLLBACK`. В их число входят операторы языка определения данных (Data Definition Language – DDL), которые создают и уничтожают базы данных, а также создают, удаляют и изменяют таблицы.

Вы должны проектировать свои транзакции таким образом, чтобы они не включали в себя эти операторы. Если ввести такой оператор в начале транзакции, которая не может быть откатана, и затем следующий оператор позже завершится аварийно, полный эффект транзакции не может быть отменен оператором `ROLLBACK`.

## 6.4.3. Операторы, вызывающие неявный COMMIT

Следующие операторы неявно завершают транзакцию (как если бы перед их выполнением был выдан `COMMIT`):

<code>ALTER TABLE</code>	<code>BEGIN</code>	<code>CREATE INDEX</code>
<code>DROP DATABASE</code>	<code>DROP INDEX</code>	<code>DROP TABLE</code>
<code>LOAD MASTER DATA</code>	<code>LOCK TABLES</code>	<code>RENAME TABLE</code>
<code>SET AUTOCOMMIT=1</code>	<code>START TRANSACTION</code>	<code>TRUNCATE TABLE</code>

`UNLOCK TABLES` также завершает транзакцию, если какие-либо таблицы были блокированы. До MySQL 4.0.13 `CREATE TABLE` завершал транзакцию, если была бы включена бинарная регистрация.

Транзакции не могут быть вложенными. Это следствие того, что неявный `COMMIT` выполняется для любой текущей транзакции, когда выполняется оператор `START TRANSACTION` или его синонимы.

## 6.4.4. Синтаксис `SAVEPOINT` и `ROLLBACK TO SAVEPOINT`

`SAVEPOINT` *идентификатор*  
`ROLLBACK TO SAVEPOINT` *идентификатор*

Начиная с версий MySQL 4.0.14 и 4.1.1, InnoDB поддерживает SQL-операторы `SAVEPOINT` и `ROLLBACK TO SAVEPOINT`.

Оператор `SAVEPOINT` устанавливает именованную точку начала транзакции с именем *идентификатор*. Если текущая транзакция уже имеет точку сохранения с таким же именем, старая точка удаляется и устанавливается новая.

Оператор `ROLLBACK TO SAVEPOINT` откатывает транзакцию к именованной точке сохранения. Модификации строк, которые выполнялись текущей транзакцией после этой точки, отменяются откатом, однако InnoDB не снимает блокировок строк, которые были установлены в памяти после точки сохранения. (Отметим, что для вновь вставленных строк информация о блокировке опирается на идентификатор транзакции, сохраненный в строке, блокировка не хранится в памяти отдельно. В этом случае блокировка строки снимается при отмене.) Точки сохранения, установленные в более поздние моменты, чем именованная точка, удаляются.

Если оператор возвращает следующую ошибку, это означает, что названная точка сохранения не существует:

```
ERROR 1181: Got error 153 during ROLLBACK
```

Все точки сохранения транзакций удаляются, если выполняется оператор COMMIT или ROLLBACK без указания имени точки сохранения.

## 6.4.5. Синтаксис LOCK TABLES и UNLOCK TABLES

LOCK TABLES

```
имя_таблицы [AS псевдоним] {READ [LOCAL] | [LOW_PRIORITY] WRITE}  
[, имя_таблицы [AS псевдоним] {READ [LOCAL] | [LOW_PRIORITY] WRITE}] ...
```

UNLOCK TABLES

LOCK TABLES блокирует таблицы для текущего потока сервера. UNLOCK TABLES снимает любые блокировки, удерживаемые текущим потоком. Все таблицы, заблокированные в текущем потоке, неявно разблокируются, когда поток выполняет другой оператор LOCK TABLES либо когда закрывается соединение с сервером.

### На заметку!

LOCK TABLES не является оператором, безопасным в отношении транзакций, и неявно завершает транзакцию перед попыткой заблокировать таблицы.

Начиная с версии MySQL 4.0.2, для того, чтобы выполнять LOCK TABLES, необходимо иметь привилегию LOCK TABLES и привилегию SELECT для соответствующих таблиц. В MySQL 3.23 необходимо иметь привилегии SELECT, INSERT, DELETE и UPDATE для этих таблиц.

Основная причина применения LOCK TABLES — эмуляция транзакций или повышение скорости обновления таблиц. Ниже это объясняется более подробно.

Если поток устанавливает блокировку по чтению (READ) на таблице, то этот поток (и все остальные) может только читать данные из таблицы. Если поток устанавливает блокировку записи (WRITE) таблицы, то лишь этот поток может читать и писать в таблицу. Доступ остальных нитей к таблице блокируется.

Разница между READ LOCAL и READ состоит в том, что READ LOCAL позволяет неконфликтующим операторам INSERT (параллельным вставкам) выполняться, пока блокировка удерживается. Однако это не может быть выполнено, если вы пытаетесь манипулировать файлами базы данных извне MySQL в то время, пока удерживается блокировка.

В случае применения LOCK TABLES необходимо блокировать все таблицы, которые используются в запросах. Если одна и та же таблица используется несколько раз в запросе (через псевдонимы), вы должны получить блокировку на каждый псевдоним. Пока блокировка, полученная от LOCK TABLES, активна, вы не можете получить доступ ни к каким таблицам, которые не были заблокированы этим оператором.

Если ваш запрос обращается к таблице через псевдоним, вы должны блокировать таблицу, используя тот же псевдоним. Блокировка таблицы не будет работать, если не указан псевдоним:

```
mysql> LOCK TABLE t READ;  
mysql> SELECT * FROM t AS myalias;  
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

И наоборот, если таблица блокирована по псевдониму, вы должны обращаться к ней, используя этот псевдоним:

```
mysql> LOCK TABLE t AS myalias READ;
mysql> SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> SELECT * FROM t AS myalias;
```

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 поток получает блокировку по чтению, а затем другой поток запрашивает блокировку по записи, то последующие запросы на блокировку по чтению будут ожидать, пока не будет установлена и снята блокировка по записи. Вы можете использовать блокировки по записи с пониженным приоритетом (LOW\_PRIORITY WRITE), чтобы позволить другим потокам устанавливать блокировки по чтению, пока текущий поток ожидает возможности поставить блокировку по записи. Вы должны устанавливать блокировки LOW\_PRIORITY WRITE только тогда, когда уверены, что в процессе работы сервера будет время, когда ни один из потоков не будет удерживать блокировки по чтению.

LOCK TABLES работает следующим образом:

1. В определенном внутреннем порядке сортируются все таблицы, подлежащие блокировке. С точки зрения пользователя, этот порядок не определен.
2. Если таблица блокируется и по чтению и по записи, устанавливается блокировка записи перед блокировкой чтения.
3. Блокируется по одной таблице за раз до тех пор, пока поток не получит все блокировки.

Эта политика гарантирует, что при этом не случится взаимных блокировок (deadlocks). Существуют, однако, и другие обстоятельства в отношении этой политики, которые следует принимать во внимание.

Если вы используете блокировку LOW\_PRIORITY WRITE для таблицы, это означает только, что MySQL будет ожидать момента, когда не будет ни одного потока, который желает установить блокировку чтения. Когда потоку удастся установить блокировку записи одной таблицы, и он ожидает возможности заблокировать следующую таблицу в списке, все остальные потоки будут приостановлены до тех пор, пока блокировка записи не будет снята. Если это представляет серьезную проблему для ваших приложений, вы должны рассмотреть возможность преобразования некоторых ваших таблиц в нетранзакционную форму.

Можно безопасно использовать KILL для прерывания потока, который ожидает блокировки таблицы. См. раздел 6.5.4.3.

Заметьте, что вы *не должны* блокировать никаких таблиц из тех, в которых выполняете INSERT DELAYED, потому что в этом случае INSERT выполняется отдельным потоком сервера.

Обычно вам не нужно блокировать таблицы, поскольку все отдельные операторы INSERT атомарны – то есть никакой другой поток не может вмешаться в исполняемый в данный момент SQL-оператор. Однако существуют случаи, когда блокировать таблицы все же необходимо:

- Если вы собираетесь выполнять множество операций над набором таблиц MyISAM, то гораздо быстрее получится, если их предварительно заблокировать. Блокиров-

ка таблиц MyISAM ускоряет вставку, обновление или удаление в них. Отрицательная сторона этого состоит в том, что ни один поток не может обновлять заблокированную по чтению таблицу (включая тот, что установил блокировку), и ни один поток не может получить никакого доступа к таблице, заблокированной по записи, кроме потока, установившего блокировку.

Причина того, что некоторые операции MyISAM работают быстрее на заблокированных таблицах, связана с тем, что MySQL не сбрасывает на диск индексный кэш для этих таблиц до тех пор, пока не будет вызван `UNLOCK TABLES`. Обычно индексные кэши сбрасываются после каждого SQL-оператора.

- Если вы используете механизм хранения MySQL, которые не поддерживаете транзакций, вы должны выдавать `LOCK TABLES`, если хотите гарантировать, что между `SELECT` и `UPDATE` не будут выданы другие операторы. Приведенный ниже пример требует `LOCK TABLES` для безопасного выполнения:

```
mysql> LOCK TABLES trans READ, customer WRITE;
mysql> SELECT SUM(value) FROM trans WHERE customer_id=идентификатор;
mysql> UPDATE customer
    -> SET total_value=сумма_из_предыдущего_оператора
    -> WHERE customer_id=идентификатор;
mysql> UNLOCK TABLES;
```

Без `LOCK TABLES` существует возможность того, что другой поток может вставить новую строку в таблицу `trans` между выполнением ваших операторов `SELECT` и `UPDATE`.

Вы можете избежать применения `LOCK TABLES` во многих случаях, применяя относительные обновления (`UPDATE customer SET value=value+new_value`), либо функцию `LAST_INSERT_ID()`. См. раздел 1.8.5.3.

Избежать блокировки таблиц можно также в некоторых случаях, используя функции пользовательского уровня `GET_LOCK()` и `RELEASE_LOCK()`. Эти блокировки сохраняются в хэш-таблице на сервере и для скорости реализуются вызовами `pthread_mutex_lock()` и `pthread_mutex_unlock()`. См. раздел 5.8.4.

Вы можете заблокировать по чтению все таблицы во всех базах данных оператором `FLUSH TABLES WITH READ LOCK`. См. раздел 6.5.4.2. Это очень удобный способ получить резервные копии, если вы работаете с файловой системой вроде Veritas, которая может делать снимки во времени.

#### На заметку!

Если вы используете `ALTER TABLE` на заблокированной таблице, она может разблокироваться. См. раздел A.3.1.

## 6.4.6. Синтаксис SET TRANSACTION

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }
```

Этот оператор устанавливает уровень изоляции следующей транзакции, глобально либо только для текущего сеанса.

Поведение `SET TRANSACTION` по умолчанию заключается в том, что он устанавливает уровень изоляции для следующей (еще не стартовавшей) транзакции. Если вы используете ключевое слово `GLOBAL`, оператор устанавливает глобальный уровень изоляции

транзакций по умолчанию для всех новых соединений, которые будут установлены с этого момента. Существующие соединения не затрагиваются. Для выполнения этого оператора нужно иметь привилегию SUPER. Применение ключевого слова SESSION устанавливает уровень изоляции по умолчанию всех будущих транзакций только для текущего сеанса.

Описание уровней изоляции транзакций InnoDB приведено в книге *MySQL. Руководство администратора* (М. : Издательский дом "Вильямс", 2005, ISBN 5-8459-0805-1). InnoDB поддерживает эти уровни, начиная с версии MySQL 4.0.5. Уровень изоляции по умолчанию — REPEATABLE READ.

Вы можете также установить начальный глобальный уровень изоляции для сервера mysqld, запустив его с опцией `--transaction-isolation`.

## 6.5. Операторы администрирования базы данных

### 6.5.1. Операторы управления учетными записями

#### 6.5.1.1. Синтаксис DROP USER

`DROP USER пользователь`

Оператор `DROP USER` удаляет учетную запись пользователя MySQL, который не имеет никаких привилегий. Он служит для удаления учетной записи из таблицы `user`. Учетная запись именуется в таком же формате, как для `GRANT` или `REVOKE`, например, `'jeffrey'@'localhost'`. Части имени пользователя и имени хоста соответствуют значению столбцов `User` и `Host` записи о пользователе в таблице `user`.

Чтобы удалить пользовательскую учетную запись, вы должны использовать следующую процедуру, выполняя шаги в указанном порядке:

1. Воспользоваться `SHOW GRANTS` для определения, какими привилегиями обладает учетная запись. См. раздел 6.5.3.10.
2. С помощью `REVOKE` лишить пользователя привилегий, показанных `SHOW GRANTS`. См. раздел 6.5.1.2.
3. Удалить учетную запись с помощью оператора `DROP USER`, или удалив запись о пользователе из таблицы `user`.

Оператор `DROP USER` был добавлен в MySQL 4.1.1. До этой версии необходимо было лишить пользователя привилегий, как описано, а потом только удалить запись из таблицы `user` и сбросить таблицы привилегий, как показано ниже:

```
mysql> DELETE FROM mysql.user
-> WHERE User='имя_пользователя' and Host='имя_хоста';
mysql> FLUSH PRIVILEGES;
```

#### 6.5.1.2. Синтаксис GRANT и REVOKE

```
GRANT тип_привилегии [{(список_столбцов)}] [, тип_привилегии
  {(список_столбцов)}] ...
ON {имя_таблицы} | * | *.* | имя_базы_данных.*
TO user [IDENTIFIED BY [PASSWORD] 'пароль']
  [, пользователь [IDENTIFIED BY [PASSWORD] 'пароль']] ...
```

```

[REQUIRE
  NONE |
  {{SSL| X509}}
  [CIPHER 'шифр' [AND]]
  [ISSUER 'выдающий' [AND]]
  [SUBJECT 'субъект']]
[WITH [GRANT OPTION | MAX_QUERIES_PER_HOUR количество |
      MAX_UPDATES_PER_HOUR количество |
      MAX_CONNECTIONS_PER_HOUR количество]]
REVOKE тип_привилегии [{(список_столбцов)}] [, тип_привилегии
  [{(список_столбцов)}] ...
ON {имя_таблицы | * | *.* } имя_базы_данных.*}
FROM пользователь [, пользователь] ...
REVOKE ALL PRIVILEGES, GRANT OPTION FROM пользователь [, пользователь] ...

```

Операторы GRANT и REVOKE позволяют администратору системы создавать учетные записи пользователей MySQL, выдавать им привилегии и отбирать их. GRANT и REVOKE реализованы в версии MySQL 3.22.11 и выше. В более ранних версиях MySQL эти операторы не делали ничего.

Информация об учетных записях хранится в таблицах базы данных mysql. Эта база и система управления доступом подробно описываются в книге *MySQL. Руководство администратора*.

Привилегии могут быть выданы на четырех уровнях:

- **Глобальный уровень.** Глобальные привилегии касаются всех баз данных на сервере. Эти привилегии сохраняются в таблице mysql.user. GRANT ALL ON \*.\* и REVOKE ALL ON \*.\* выдают и отбирают только глобальные привилегии.
- **Уровень базы данных.** Привилегии уровня базы данных касаются всех таблиц в базе данных. Эти привилегии хранятся в таблицах mysql.db и mysql.host. GRANT ALL ON имя\_базы\_данных.\* и REVOKE ALL ON имя\_базы\_данных.\* выдают и отбирают только привилегии уровня базы данных.
- **Уровень таблицы.** Привилегии уровня таблицы касаются всех столбцов в данной таблице. Эти привилегии хранятся в таблице mysql.tables\_priv. GRANT ALL ON имя\_базы\_данных.имя\_таблицы и REVOKE ALL ON имя\_базы\_данных.имя\_таблицы выдают и отбирают только привилегии уровня таблицы.
- **Уровень столбца.** Привилегии уровня столбца касаются отдельных столбцов в таблице. Эти привилегии хранятся в таблице mysql.columns\_priv. При использовании REVOKE вы должны указать те же столбцы, которые присутствовали в GRANT.

Чтобы облегчить лишение всех привилегий, в MySQL 4.1.2 добавлен следующий синтаксис, который удаляет все привилегии уровня баз данных, таблиц и столбцов для указанных пользователей:

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM пользователь [, пользователь] ...
```

До версии MySQL 4.1.2 все привилегии нельзя было удалить сразу. Необходимо было выполнить два оператора:

```

REVOKE ALL PRIVILEGES FROM пользователь [, пользователь] ...
REVOKE GRANT OPTION FROM пользователь [, пользователь] ...

```

Для операторов GRANT и REVOKE аргумент тип\_привилегии может принимать значения, перечисленные в табл. 6.1.



Таблица 6.1. Привилегии, используемые в операторах GRANT и REVOKE

Привилегия	Назначение
ALL [PRIVILEGES]	Устанавливает простые привилегии кроме GRANT OPTION.
ALTER	Позволяет использовать ALTER TABLE.
CREATE	Позволяет использовать CREATE TABLE.
CREATE TEMPORARY TABLES	Позволяет использовать CREATE TEMPORARY TABLE.
DELETE	Позволяет использовать DELETE.
DROP	Позволяет использовать DROP TABLE.
EXECUTE	Позволяет запускать хранимые процедуры (MySQL 5.0).
FILE	Позволяет использовать SELECT...INTO OUTFILE и LOAD DATA INFILE.
INDEX	Позволяет использовать CREATE INDEX и DROP INDEX.
INSERT	Позволяет использовать INSERT.
LOCK TABLES	Позволяет использовать LOCK TABLES в таблицах, для которых имеется привилегия SELECT.
PROCESS	Позволяет использовать SHOW FULL PROCESSLIST.
REFERENCES	Пока не реализовано.
RELOAD	Позволяет использовать FLUSH.
REPLICATION CLIENT	Позволяет запрашивать, является ли сервер главным или подчиненным в репликации.
REPLICATION SLAVE	Необходимо для подчиненных серверов в репликации (для чтения бинарных журналов главного сервера).
SELECT	Позволяет использовать SELECT.
SHOW DATABASES	SHOW DATABASES показывает все базы данных.
SHUTDOWN	Позволяет применять mysqladmin shutdown.
SUPER	Позволяет выполнять операторы CHANGE MASTER, KILL, PURGE MASTER LOGS и SET GLOBAL, команду mysqladmin debug, разрешает подключаться (один раз), даже если достигнуто число соединений max_connections.
UPDATE	Позволяет использовать UPDATE.
USAGE	Синоним для "отсутствия привилегий".
GRANT OPTION	Позволяет передавать привилегии другим пользователям.

USAGE можно применять для создания пользователя, лишенного всех привилегий.

Новые привилегии CREATE TEMPORARY TABLES, EXECUTE, LOCK TABLES, REPLICATION..., SHOW DATABASES и SUPER введены в MySQL 4.0.2. Чтобы их использовать после обновления сервера до версии 4.0.2, необходимо запустить сценарий `mysql_fix_privilege_tables`.

В старых версиях MySQL, где не было привилегии SUPER, вместо нее можно было использовать PROCESS.

Вы можете выдавать глобальные привилегии, применяя синтаксис `ON *.*`, либо привилегии уровня базы данных, используя `ON имя_базы_данных.*`. Если вы указываете `ON *`, и у вас есть текущая база данных по умолчанию, привилегии будут выданы на текущую базу данных. (**Внимание!** Если использовать синтаксис `ON *`, не имея текущей базы, то привилегии будут выданы глобально.)

Привилегии `EXECUTION`, `FILE`, `PROCESS`, `RELOAD`, `REPLICATION CLIENT`, `REPLICATION SLAVE`, `SHOW DATABASES`, `SHUTDOWN` и `SUPER` являются административными и могут быть выданы только глобально (с использованием синтаксиса `ON *.*`).

Другие привилегии могут быть выданы глобально либо на более низком уровне.

Для таблиц можно установить только следующие типы привилегий: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `DROP`, `GRANT OPTION`, `INDEX` и `ALTER`.

Для столбцов (когда используется конструкция `список_столбцов`) можно установить только следующие типы привилегий: `SELECT`, `INSERT` и `UPDATE`.

`GRANT ALL` выдает привилегии, существующие только на текущем уровне. Например, если вы используете `GRANT ALL ON имя_базы_данных.*`, что является оператором уровня базы данных, то никакие из глобальных привилегий наподобие `FILE` не выдаются.

MySQL позволяет назначать привилегии уровня базы данных, даже если эта база не существует. Это облегчает подготовку к эксплуатации базы данных. Однако в настоящий момент MySQL не позволяет назначать привилегии уровня таблицы для несуществующих таблиц.

MySQL не удаляет никаких привилегий автоматически, даже если вы уничтожаете таблицу или базу данных.

### На заметку!

Шаблонные символы `'%'` и `'_'` допускаются в спецификациях имен баз данных для операторов `GRANT` глобального уровня и уровня базы данных. Это означает, например, что если вы хотите использовать символ `'_'` как часть имени базы, то должны предварять его обратной косой чертой (`'\_'`) в аргументах оператора `GRANT`, чтобы предотвратить нежелательный доступ к базам с именами, соответствующими шаблону, например, `GRANT...ON 'foo\_bar'.* TO...`

Для того чтобы выдавать права доступа пользователям с различных хостов, MySQL поддерживает спецификацию значения *пользователь* в форме `имя_пользователя@имя_хоста`. Если вам нужно указать строку `имя_пользователя`, включающую специальные символы (такие как `'-'`), либо строку `имя_хоста`, включающую специальные символы или символы шаблонов (подобные `'%'`), можете поместить в кавычки `имя_пользователя` и `имя_хоста` (например, `'test-user'@'test-hostname'`). Имена пользователя и хоста берутся в кавычки по отдельности.

Вы можете использовать шаблонные символы в имени хоста. Например, `имя_пользователя@%.loc.gov` касается пользователей `имя_пользователя` на любых хостах домена `loc.gov`, а `имя_пользователя@'144.155.166.%'` касается пользователей `имя_пользователя` с любого хоста подсети класса C 144.155.166.

Простая форма `имя_пользователя` — синоним для `имя_пользователя@'%'`.

MySQL не поддерживает символы шаблона в именах пользователей. Анонимные пользователи заводятся вставкой строк с `User=''` в таблицу `mysql.user`, или созданием пользователя с пустым именем, с помощью оператора `GRANT`:

```
mysql> GRANT ALL ON test.* TO ''@'localhost' ...
```

**Внимание!**

Если вы разрешаете анонимному пользователю подключаться к серверу MySQL, то также должны выдать привилегии всем локальным пользователям *имя\_пользователя@localhost*. В противном случае учетная запись анонимного пользователя для локального хоста в таблице `mysql.user` будет задействована, когда именованные пользователи попытаются подключиться к серверу MySQL с локальной машины! (Учетная запись анонимного пользователя создается во время установки MySQL.)

Вы можете определить, касается ли это вас, выполнив следующий запрос:

```
mysql> SELECT Host, User FROM mysql.user WHERE User='';
```

Если во избежание описанной проблемы вы хотите удалить учетную запись анонимного локального пользователя, примените следующие операторы:

```
mysql> DELETE FROM mysql.user WHERE Host='localhost' AND User='';  
mysql> FLUSH PRIVILEGES;
```

На данный момент GRANT поддерживает имена хостов, таблиц, баз данных и столбцов до 60 символов длиной. Имя пользователя может быть не длиннее 16 символов.

Привилегии на доступ к таблицам и столбцам формируются логическим ИЛИ из привилегий, выданных на каждом из четырех уровней. Например, если в таблице `mysql.user` указано, что пользователь имеет глобальную привилегию SELECT, то эта привилегия не может быть отвергнута на уровне базы данных, таблицы или столбца.

Привилегии на столбцы могут быть вычислены следующим образом:

*глобальные привилегии*

*ИЛИ (привилегии базы данных И привилегии хоста)*

*ИЛИ привилегии таблицы*

*ИЛИ привилегии столбца*

В большинстве случаев привилегии выдаются пользователю только на одном из уровней, поэтому в реальной жизни это все не так сложно.

Если выдаются привилегии для комбинации “имя пользователя/имя хоста”, которой нет в таблице `mysql.user`, эта комбинация будет добавлена туда и останется там до тех пор, пока не будет удалена с помощью оператора DELETE.

Другими словами, GRANT может создавать новых пользователей, но REVOKE не может удалять их. Вы должны делать это явно, применяя DROP USER или DELETE.

В MySQL 3.22.12 и последующих версиях, если создается новый пользователь или если вы имеете глобальные привилегии на выдачу других привилегий, то пользовательский пароль устанавливается конструкцией IDENTIFIED BY, если таковая указана. Если у пользователя уже был пароль, он заменяется новым.

**Внимание!**

Если вы создаете нового пользователя без конструкции IDENTIFIED BY, то пользователь не имеет пароля. Это небезопасно.

Пароли также могут устанавливаться оператором SET PASSWORD. См. раздел 6.5.1.3.

Если вы не хотите отправлять пароль в виде открытого текста, можно после ключевого слова PASSWORD давать зашифрованный пароль, который возвращает SQL-функция PASSWORD() или функция C API `make_scrambled_password()`.

Если выдается привилегия на доступ к базе данных, запись в `mysql.user` заводится при необходимости. Если все привилегии на доступ к базе отбираются оператором `REVOKE`, эта запись удаляется.

Если у пользователя нет привилегий на доступ к таблице, то эта таблица не отображается в ответ на запрос списка таблиц базы (например, оператором `SHOW TABLES`). Если пользователь не имеет привилегий на доступ к базе данных, имя базы не отображается в ответ на запрос `SHOW DATABASES`, если только у пользователя нет специальной привилегии `SHOW DATABASES`.

Конструкция `WITH GRANT OPTION` дает пользователю возможность самому выдавать привилегии заданного уровня другим пользователям. Вы должны быть осторожны с тем, кому выдаете привилегии с `GRANT OPTION`, поскольку два пользователя с разными привилегиями могут объединить свои привилегии!

Вы не можете выдать другому пользователю привилегию, которой не имеете сами. `GRANT OPTION` позволяет выдавать только те привилегии, которые вам принадлежат.

Имейте в виду, что когда вы дадите пользователю привилегию `GRANT OPTION` на определенном уровне привилегий, то все привилегии, принадлежащие ему на этом уровне (и те, которые он получит в будущем!), он сможет выдавать другим. Предположим, что вы дадите пользователю привилегию `INSERT` в базе данных. Если затем вы дадите ему привилегию `SELECT` на эту же базу с опцией `GRANT OPTION`, то он сможет выдавать другим не только привилегию `SELECT`, но и `INSERT`. Если вы дадите ему позже еще и привилегию `UPDATE` на ту же базу данных, то он сможет раздавать `INSERT`, `SELECT` и `UPDATE`.

Вы не должны давать привилегий `ALTER` обычному пользователю. Если вы это сделаете, он сможет потом разрушить систему, просто переименовав таблицы!

Опции `MAX_QUERIES_PER_HOUR количество`, `MAX_UPDATES_PER_HOUR количество` и `MAX_CONNECTIONS_PER_HOUR количество` — новые для MySQL 4.0.2. Они ограничивают количество запросов, обновлений и регистраций, которые пользователь может выполнить в течение часа. Если значение `количество` равно 0 (по умолчанию), это означает, что ограничений для данного пользователя не накладывается.

### На заметку!

Чтобы указать любую из этих опций для пользователя, не затрагивая его существующих привилегий, применяйте `GRANT USAGE ON *.* ... WITH MAX ...`.

MySQL может проверять атрибуты сертификата X509 в дополнение к обычной аутентификации, основанной на имени пользователя и пароле. Чтобы специфицировать SSL-опции пользовательской учетной записи MySQL, применяйте конструкцию `REQUIRE` оператора `GRANT`. Существуют различные возможности установить ограничения на типы подключений пользователя:

- Если для учетной записи не установлены требования SSL или X509, то нешифрованные подключения разрешаются, если указаны правильно имя и пароль. Однако шифрованные подключения также могут быть установлены пользователем, если клиент имеет правильный сертификат и файлы ключей.
- Опция `REQUIRE SSL` сообщает серверу, что учетной записи разрешаются только шифрованные SSL-подключения. Отметим, что эта опция может быть опущена, если есть какие-либо записи контроля доступа, разрешающие не SSL-подключения.

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'  
-> IDENTIFIED BY 'goodsecret' REQUIRE SSL;
```

- REQUIRE X509 означает, что клиент должен иметь действительный сертификат, но какой именно, кем и кому выданный – не важно. Единственное требование, чтобы можно было сверить его сигнатуру одним из сертификатов CA.

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'  
-> IDENTIFIED BY 'goodsecret' REQUIRE X509;
```

- REQUIRE ISSUER '*выдающий*' накладывает ограничение на попытки подключений, которое состоит в том, что клиент должен представить действительный сертификат X509, выданный CA '*выдающий*'. Если клиент представляет действительный сертификат, но выданный кем-то другим, сервер отвергает подключение. Применение сертификата X509 всегда подразумевает шифрование, поэтому опция SSL необязательна.

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'  
-> IDENTIFIED BY 'goodsecret'  
-> REQUIRE ISSUER '/C=FI/ST=Some-State/L=Helsinki/  
O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com';
```

Отметим, что значение ISSUER должно быть представлено как одна строка.

- REQUIRE SUBJECT '*субъект*' накладывает ограничение на попытки подключений, которое состоит в том, что клиент должен предъявить действительный сертификат X509 с указанием субъекта '*субъект*' в нем. Если клиент представляет действительный сертификат, но с другим субъектом в нем, сервер отвергает подключение.

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'  
-> IDENTIFIED BY 'goodsecret'  
-> REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/  
O=MySQL demo client certificate/  
CN=Tonu Samuel/Email=tonu@example.com';
```

Отметим, что значение SUBJECT должно быть представлено как одна строка.

- REQUIRE CIPHER '*шифр*' необходимо 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, вы можете затребовать некоторый конкретный метод шифрования, чтобы разрешить соединение.

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'  
-> IDENTIFIED BY 'goodsecret'  
-> REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

Опции SUBJECT, ISSUER и CIPHER могут комбинироваться в конструкции REQUIRE:

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'  
-> IDENTIFIED BY 'goodsecret'  
-> REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/  
O=MySQL demo client certificate/  
CN=Tonu Samuel/Email=tonu@example.com'  
-> AND ISSUER '/C=FI/ST=Some-State/L=Helsinki/  
O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com'  
-> AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

Следует отметить, что значения SUBJECT и ISSUER должны быть представлены как одна строка.

Начиная с MySQL 4.0.4, допускается необязательное слово AND между опциями REQUIRE.

Последовательность опций не имеет значения, но ни одна не должна повторяться.

При запуске `mysqld` все привилегии считываются в память. Привилегии на базы данных, таблицы и столбцы вступают в действие сразу, а привилегии уровня пользователя — при следующем его подключении. О модификациях в таблицах привилегий, выполняемых операторами GRANT и REVOKE, сервер извещается немедленно. Если вы модифицируете таблицы привилегий вручную (применяя INSERT, UPDATE и так далее), вы должны выполнить оператор FLUSH PRIVILEGES или запустить `mysqladmin flush-privileges`, чтобы сообщить серверу о необходимости перезагрузки таблиц привилегий в память.

Заметьте, что если вы используете привилегии на таблицы и столбцы всего для одного пользователя, сервер проверяет эти привилегии для всех пользователей и это несколько замедляет работу MySQL. Аналогично, если вы ограничиваете количество запросов, обновлений или подключений для любых пользователей, сервер должен вести мониторинг этих значений.

Наибольшие различия между стандартом SQL и версией GRANT от MySQL:

- В MySQL привилегии ассоциируются с комбинацией имя пользователя/имя хоста, а не только с одним именем пользователя.
- Стандарт SQL не предусматривает привилегий глобального уровня и уровня базы данных, как и типов привилегий, поддерживаемых MySQL.
- MySQL не поддерживает стандартных SQL-привилегий TRIGGER и UNDER.
- Стандартные привилегии SQL организованы в иерархической манере. Если вы удаляете пользователя, все его привилегии изымаются. В MySQL выданные привилегии не отнимаются автоматически, вы должны это делать сами.
- В стандартном SQL, когда удаляется таблица, все привилегии на нее отнимаются. В стандартном SQL, когда отнимается привилегия, все привилегии, выданные на основе данной привилегии, также отнимаются. В MySQL привилегии могут быть отняты только явным вызовом оператора REVOKE, либо манипуляциями в таблицах привилегий.
- В MySQL, если вы имеете привилегию INSERT только на некоторые столбцы таблицы, вы можете выполнять оператор INSERT в этой таблице. Столбцам, для которых у вас нет привилегии INSERT, будут присвоены значения по умолчанию. Стандарт SQL требует, чтобы вы имели привилегию INSERT на все столбцы.

### 6.5.1.3. Синтаксис SET PASSWORD

```
SET PASSWORD = PASSWORD('пароль')  
SET PASSWORD FOR пользователь = PASSWORD('пароль')
```

Оператор SET PASSWORD назначает пароль существующей пользовательской учетной записи MySQL.

Первый синтаксис устанавливает пароль текущему пользователю. Любой клиент, подключенный к серверу с использованием неанонимной учетной записи, может изменить свой пароль.

Второй синтаксис присваивает пароль указанной учетной записи на текущем хосте сервера. Только клиенты, имеющие доступ к базе данных mysql, могут это делать. Значение *пользователь* должно быть задано в формате *имя\_пользователя@имя\_хоста*, где

имя\_пользователя и имя\_хоста — точно такие же, которые указаны в столбцах User и Host таблицы mysql.user. Например, если у вас есть строка со значениями User и Host, равными, соответственно, 'bob' и '%.loc.gov', вы должны написать оператор следующим образом:

```
mysql> SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('newpass');
```

Это эквивалентно такому варианту:

```
mysql> UPDATE mysql.user SET Password=PASSWORD('newpass')
-> WHERE User='bob' AND Host='%.loc.gov';
mysql> FLUSH PRIVILEGES;
```

## 6.5.2. Операторы обслуживания таблиц

### 6.5.2.1. Синтаксис ANALYZE TABLE

```
ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE имя_таблицы [, имя_таблицы] ...
```

Этот оператор анализирует и сохраняет распределение ключей для таблицы. Во время анализа таблица блокируется по чтению. Это работает для таблиц MyISAM и BDB, а также (начиная с MySQL 4.0.13) InnoDB. Для таблиц MyISAM данный оператор эквивалентен myisamchk -a.

MySQL использует сохраненные данные о распределении ключей, чтобы принимать решения относительно порядка объединения таблиц, когда в запросе требуется объединение по неконстантным значениям.

Оператор возвращает таблицу со следующими столбцами:

Столбец	Значение
Table	Имя таблицы
Op	Всегда Analyze
Msg_type	Тип: status, error, info или warning
Msg_text	Сообщение

Вы можете проверить сохраненное распределение ключей с помощью оператора SHOW INDEX. См. раздел 6.5.3.7.

Если таблица не изменялась с последнего выполнения ANALYZE TABLE, ее анализ выполняться не будет.

До MySQL 4.1.1 оператор ANALYZE TABLE не записывался в бинарный журнал. Начиная с MySQL 4.1.1, он записывается, если только не было указано необязательное ключевое слово NO\_WRITE\_TO\_BINLOG (или его псевдоним LOCAL).

### 6.5.2.2. Синтаксис BACKUP TABLE

```
BACKUP TABLE имя_таблицы [, имя_таблицы] ...
TO '/путь/к/каталогу/резервных_копий'
```

#### На заметку!

Этот оператор устарел. Мы работаем над его улучшенной заменой, которая позволит выполнять горячее резервное копирование. А пока вместо этого можно пользоваться сценарием mysqlhotcopy.

BACKUP TABLE копирует в резервный каталог минимальный набор файлов таблицы, необходимый для ее последующего восстановления, после сброса всех буферизованных

изменений на диск. Этот оператор работает только с таблицами MyISAM. Он копирует файлы определения таблиц `.frm` и файлы данных `.MYD`. Индексные файлы `.MYI` могут быть перестроены на основе первых двух. Каталог должен быть указан с полным путем.

При резервном копировании каждая таблица последовательно блокируется по чтению. Если вы хотите скопировать несколько таблиц в виде снимка (предотвращая изменения в них во время резервирования), вы должны сначала выполнить оператор `LOCK TABLES`, чтобы установить блокировку по чтению для всех таблиц группы.

Оператор возвращает таблицу со следующими столбцами:

Столбец	Значение
Table	Имя таблицы
Op	Всегда Backup
Msg_type	Тип: status, error, info или warning
Msg_text	Сообщение

Оператор `BACKUP TABLE` доступен в MySQL 3.23.25 и последующих версиях.

### 6.5.2.3. Синтаксис CHECK TABLE

```
CHECK TABLE имя_таблицы [, имя_таблицы] ... [опция] ...
опция = {QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
```

`CHECK TABLE` работает только с таблицами MyISAM и InnoDB. Для таблиц MyISAM это то же самое, что и запуск `myisamchk --medium-check имя_таблицы`.

Если никакая опция не указана, принимается `MEDIUM`.

Проверяет таблицу или группу таблиц на наличие ошибок. Для таблиц MyISAM обновляется статистическая информация о ключах. Оператор возвращает таблицу со следующими столбцами:

Столбец	Значение
Table	Имя таблицы
Op	Всегда Check
Msg_type	Тип: status, error, info или warning
Msg_text	Сообщение

Следует отметить, что оператор может выдать множество строк информации о каждой проверяемой таблице. Последняя строка будет содержать значение `Msg_type`, равное `status` и `Msg_text` обычно должно быть `OK`. Если вы не получаете `OK` или же `Table is already up to date` (Таблица уже обновлена), обычно вы должны будете запустить восстановление таблицы. Сообщение `Table is already up to date` означает, что механизм хранения сообщает о том, что необходимости проверять таблицу не было.

Существуют следующие типы проверок:

Тип	Значение
QUICK	Не сканировать строки в поисках некорректных связей.
FAST	Проверять только те таблицы, которые не были корректно закрыты.
CHANGED	Проверять только те таблицы, которые изменились с момента последней проверки или не были корректно закрыты.



Тип	Значение
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 таблицы, но требует много времени.

Если не указана ни одна из опций QUICK, MEDIUM или EXTENDED, то по умолчанию для таблиц динамического формата MyISAM выбирается MEDIUM. Типом проверки для статических таблиц MyISAM также будет MEDIUM, если только не указано CHANGED или FAST. В этом случае по умолчанию выбирается QUICK. Для CHANGED и FAST проверка строк опускается, потому что строки очень редко оказываются поврежденными.

Вы можете комбинировать опции проверки, как показано в следующем примере, выполняющем быструю проверку таблицы на предмет того, была ли она закрыта правильно:

```
CHECK TABLE test_table FAST QUICK;
```

#### На заметку!

В некоторых случаях CHECK TABLE изменяет таблицу! Это происходит, если таблица помечена, как "поврежденная" или "неправильно закрытая", но CHECK TABLE не находит в ней никаких проблем. В этом случае CHECK TABLE помечает ее как исправную.

Если таблица повреждена, вероятнее всего, что причина проблем связана с индексацией, а не с данными. Все типы проверок включают полную проверку индексов и поэтому в состоянии обнаружить большинство ошибок.

Если вы только хотите проверить таблицу, о которой предполагаете, что с ней все в порядке, вы не должны указывать никаких опций проверки или опцию QUICK. Последняя должна применяться, когда нужно спешить, и риск того, что QUICK не найдет ошибок в файле данных, невелик. (В большинстве случаев при нормальной работе MySQL должен найти любые ошибки в файле данных. Если это происходит, таблица помечается как "поврежденная" и не может использоваться до тех пор, пока не будет восстановлена.)

FAST и CHANGED наиболее хорошо подходят для применения в сценариях (например, выполняемых автоматически демоном cron), если вы хотите проверять таблицы в соответствии с каким-нибудь графиком. В большинстве случаев опция FAST более предпочтительна, нежели CHANGED. (Единственный случай, когда FAST не предпочтительна, это если вы предполагаете, что нашли ошибку в коде MyISAM.)

EXTENDED предназначена для применения только после того, как выполнялась обычная проверка, но происходят странные ошибки при попытках MySQL обновить строку или найти ее по ключу. (Это очень маловероятно, если нормальная проверка завершилась успешно.)

Ниже описаны некоторые проблемы, о которых сообщает CHECK TABLE, и которые не могут быть исправлены автоматически.

- Found row where the auto\_increment column has the value 0 (Найдена строка, в которой столбец auto\_increment имеет значение 0).

Это означает, что в таблице содержится строка, в которой значение столбца AUTO\_INCREMENT равно 0. (Существует возможность создать строку со значением столбца AUTO\_INCREMENT, равным 0, с помощью оператора UPDATE.)

Это не является ошибкой само по себе, но может вызвать проблемы, если вы решите выгрузить дампы такой таблицы, а затем восстановить ее из дампа либо выполнить ALTER TABLE. В этом случае столбец AUTO\_INCREMENT изменит значение в соответствии с правилами, действующими относительно таких столбцов, что может привести к проблеме дублирования ключей.

Чтобы избавиться от упомянутого предупреждения, просто выполните оператор UPDATE и присвойте столбцу значение, отличное от 0.

#### 6.5.2.4. Синтаксис CHECKSUM TABLE

CHECKSUM TABLE имя\_таблицы [, имя\_таблицы] ... [ QUICK | EXTENDED ]

Сообщает контрольную сумму таблицы.

Если указана опция QUICK, сообщается актуальная контрольная сумма таблицы, если она доступна, в противном случае выдается NULL. Это делается очень быстро. Актуальная контрольная сумма включается опцией таблицы CHECKSUM=1, которая в настоящее время поддерживается только для таблиц MyISAM. См. раздел 6.2.5.

При выборе режима EXTENDED выполняется чтение всей таблицы – строка за строкой, и вычисляется контрольная сумма. Для больших таблиц это может происходить очень долго.

По умолчанию, если не указано ни QUICK, ни EXTENDED, MySQL возвращает актуальную контрольную сумму таблицы, если механизм хранения ее поддерживает, либо выполняет сканирование таблицы в противном случае.

Оператор реализован в версии MySQL 4.1.1.

#### 6.5.2.5. Синтаксис OPTIMIZE TABLE

OPTIMIZE [LOCAL | NO\_WRITE\_TO\_BINLOG] TABLE имя\_таблицы [, имя\_таблицы] ...

OPTIMIZE TABLE необходимо использовать, если удалялась большая часть строк таблицы либо выполнялось много изменений в таблице с переменной длиной строки (таблицы с столбцами типов VARCHAR, BLOB или TEXT). Удаленные строки помещаются в связанный список и последующие операции INSERT повторно используют старые позиции записей. Вы можете применять OPTIMIZE TABLE для восстановления неиспользованного пространства и дефрагментации файла данных.

В большинстве случаев вообще нет необходимости вызывать OPTIMIZE TABLE. Даже если вносится множество изменений в строки переменной длины, маловероятно, что вам понадобится делать это чаще раза в неделю или в месяц, да и то – только для некоторых таблиц.

В настоящий момент OPTIMIZE TABLE работает только с таблицами MyISAM и BDB. Причем в случае таблиц BDB оператор OPTIMIZE TABLE отображается на ANALYZE TABLE. См. раздел 6.5.2.1.

Можно заставить работать OPTIMIZE TABLE с другими типами таблиц, запустив сервер mysqld с опцией --skip-new или --safe-mode, но в этом случае OPTIMIZE TABLE просто отображается на ALTER TABLE.

OPTIMIZE TABLE работает так:

1. Если в таблице есть удаленные или разделенные строки, восстанавливает таблицу.
2. Если страницы индекса не отсортированы, сортирует их.
3. Если статистическая информация устарела (и восстановление не может быть выполнено простой сортировкой индекса), обновляет ее.

Отметим, что MySQL блокирует таблицу на время выполнения `OPTIMIZE TABLE`.

До MySQL 4.1.1 оператор `OPTIMIZE TABLE` не записывался в бинарный журнал. Начиная с MySQL 4.1.1, он записывается, если только не было указано необязательное ключевое слово `NO_WRITE_TO_BINLOG` (или его псевдоним `LOCAL`).

### 6.5.2.6. Синтаксис REPAIR TABLE

```
REPAIR [LOCAL | NO_WRITE_TO_BINLOG] TABLE
      имя_таблицы [, имя_таблицы] ... [QUICK] [EXTENDED] [USE_FRM]
```

`REPAIR TABLE` осуществляет восстановление (ремонт) поврежденных таблиц. По умолчанию, он имеет тот же эффект, что и `myisamchk --recover имя_таблицы`.

`REPAIR TABLE` работает только с таблицами MyISAM. Обычно вообще нет необходимости вызывать `REPAIR TABLE`. Однако если неприятность все же возникла, то весьма вероятно, что `REPAIR TABLE` сможет восстановить все ваши данные в таблицах MyISAM. Если ваши таблицы часто получают повреждения, вам нужно найти причину этого, чтобы избежать необходимости часто запускать `REPAIR TABLE`.

Оператор возвращает таблицу со следующими столбцами:

Столбец	Значение
Table	Имя таблицы
Op	Всегда Repair
Msg_type	Тип: status, error, info или warning
Msg_text	Сообщение

Оператор `REPAIR TABLE` может выдать множество строк с информацией о каждой восстанавливаемой таблице. Последняя строка будет содержать значение `Msg_type`, равное `status`, и `Msg_text` обычно должно быть `OK`. Если вы не получаете `OK`, то вам нужно попробовать восстановить таблицу командой `myisamchk --safe-recover`, потому что `REPAIR TABLE` пока не реализует всех возможностей `myisamchk`. Мы планируем в будущем сделать этот оператор более гибким.

Если указано `QUICK`, `REPAIR TABLE` пытается восстановить только дерево индекса. Этот тип восстановления аналогичен `myisamchk --recover --quick`.

Если применяется опция `EXTENDED`, MySQL создает индексы строку за строкой, вместо того, чтобы создавать их сортировкой по одному за раз. (До версии MySQL 4.1 это может быть лучше, чем сортировка по ключам фиксированной длины, особенно если у вас длинный ключ типа `CHAR`, который хорошо пакуется.) Этот тип восстановления аналогичен `myisamchk --safe-recover`.

Начиная с MySQL 4.0.2, в `REPAIR TABLE` добавлен режим `USE_FRM`. Его следует применять, если отсутствует файл индексов `.MYI`, или если его заголовок поврежден. В этом режиме MySQL пересоздает файл `.MYI`, используя информацию из файла `.frm`. Этот способ восстановления невозможно осуществить средствами `myisamchk`.

#### Внимание!

Если сервер аварийно завершается во время выполнения оператора `REPAIR TABLE`, важно после его перезапуска немедленно снова запустить `REPAIR TABLE`, прежде чем выполнять какие-то манипуляции с таблицей. (Всегда полезно начинать с создания резервной копии.) В худшем случае вы можете получить новый чистый индексный файл, без информации о файле данных, и следующая операция, которую вы попытаетесь выполнить, может перезаписать файл данных. Это маловероятно, но все же возможно.

До MySQL 4.1.1 операторы REPAIR TABLE не записывались в бинарный журнал. Начиная с MySQL 4.1.1, они записываются, если только не было указано необязательное ключевое слово NO\_WRITE\_TO\_BINLOG (или его псевдоним LOCAL).

### 6.5.2.7. Синтаксис RESTORE TABLE

```
RESTORE TABLE имя_таблицы [, имя_таблицы] ...
FROM '/путь/к/каталогу/резервных_копий'
```

Этот оператор восстанавливает таблицу из резервной копии (не путать с “восстановлением” оператором REPAIR TABLE, который, по сути, выполняет ремонт поврежденной таблицы – *прим. перев.*), которая была создана оператором BACKUP TABLE. Существующие таблицы перезаписаны не будут, если вы попытаетесь восстановить поверх существующей таблицы, то получите сообщение об ошибке. Точно так же, как и BACKUP TABLE, в настоящее время RESTORE TABLE работает только с таблицами MyISAM. Каталог должен быть указан с полным путем.

Резервная копия каждой таблицы состоит из его файла формата .frm и файла данных .MYD. Операция восстановления из резервной копии возвращает в базу эти файлы, а затем строит по ним индексный файл .MYI. Восстановление из копии требует больше времени, чем резервное копирование, поскольку нужно перестраивать индексы. Чем больше у таблицы индексов, тем дольше это происходит.

Оператор возвращает таблицу со следующими столбцами:

Столбец	Значение
Table	Имя таблицы
Op	Всегда Restore
Msg_type	Тип: status, error, info или warning
Msg_text	Сообщение

### 6.5.3. Синтаксис SET и SHOW

SET позволяет устанавливать переменные и опции.

SHOW имеет множество различных форм, которые представляют информацию о базах данных, таблицах, столбцах, а также информацию о состоянии сервера.

В данном разделе описаны следующие синтаксические формы:

```
SHOW [FULL] COLUMNS FROM имя_таблицы [FROM имя_базы_данных] [LIKE 'шаблон']
SHOW CREATE DATABASE имя_базы_данных
SHOW CREATE TABLE имя_таблицы
SHOW DATABASES [LIKE 'шаблон']
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [смещение,] row_count]
SHOW GRANTS FOR пользователь
SHOW INDEX FROM имя_таблицы [FROM имя_базы_данных]
SHOW INNODB STATUS
SHOW [BDB] LOGS
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW STATUS [LIKE 'шаблон']
SHOW TABLE STATUS [FROM имя_базы_данных] [LIKE 'шаблон']
SHOW [OPEN] TABLES [FROM имя_базы_данных] [LIKE 'шаблон']
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'шаблон']
SHOW WARNINGS [LIMIT [смещение,] row_count]
```

Если синтаксис данного оператора `SHOW` включает часть `LIKE 'шаблон'`, то 'шаблон' — это строка, которая может содержать символы SQL-шаблонов `'%'` и `'_'`. Шаблон (pattern) применяется для ограничения вывода только соответствующими значениями.

Отметим, что существуют и другие формы этих операторов, описанные в других местах:

- Оператор `SET PASSWORD` — для присвоения пароля пользовательской учетной записи, описан в разделе 6.5.1.3.
- Оператор `SHOW` имеет формы, предоставляющие информацию о главных и подчиненных серверах в репликации:

```
SHOW BINLOG EVENTS
SHOW MASTER LOGS
SHOW MASTER STATUS
SHOW SLAVE HOSTS
SHOW SLAVE STATUS
```

Эти формы `SHOW` описаны в разделе 6.6.

### 6.5.3.1. Синтаксис SET

`SET` присваивание\_переменной [, присваивание\_переменной] ...

присваивание\_переменной:

```
имя_пользовательской_переменной = выражение
| [GLOBAL|SESSION] имя_системной_переменной = выражение
| @@[global.|session.]имя_системной_переменной = выражение
```

Оператор `SET` устанавливает значение различным типам переменных, которые влияют на работу сервера и клиента. Он может применяться для присвоения значений пользовательским и системным переменным.

В MySQL 4.0.3 были добавлены опции `GLOBAL` и `SESSION`, а также была обеспечена возможность наиболее важным системным переменным изменяться динамически в процессе работы.

В более старых версиях MySQL использовалась опция `SET OPTION` вместо `SET`, но на данный момент она устарела. Просто опускайте слово `OPTION`.

Следующие примеры показывают различный синтаксис, который используется для установки переменных.

Пользовательская переменная записывается как `@имя_переменной` и может быть установлена следующим образом:

```
SET @имя_переменной = выражение;
```

Более подробная информация о пользовательских переменных представлена в разделе 2.3.

К системным переменным в операторе `SET` можно обращаться как к `имя_переменной`. Имени может предшествовать необязательное слово `GLOBAL` или `@@global.` для явного указания, что это глобальная переменная, либо `SESSION` или `@@session.` — что это переменная сеанса. `LOCAL` и `@@local.` — синонимы `SESSION` и `@@session.`

Если не указан ни один модификатор, `SET` устанавливает переменные сеанса.

Синтаксис `@@имя_переменной` для системных переменных поддерживается MySQL с целью обеспечения совместимости с некоторыми другими системами баз данных.

При установке нескольких системных переменных в одном операторе, последняя опция GLOBAL или SESSION применяется для переменных, для которых модификатор не указан.

```
SET sort_buffer_size=10000;
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=1000000;
SET @@sort_buffer_size=1000000;
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000000;
```

Если вы устанавливаете системную переменную, используя SESSION (по умолчанию), ее значение остается в силе до конца сеанса либо до того момента, когда ей будет присвоено другое значение. Если вы устанавливаете системную переменную, указав слово GLOBAL, что требует привилегии SUPER, ее значение запоминается и используется в новых соединениях до тех пор, пока сервер не будет перезапущен. Если вы хотите сделать установку переменной неизменяемой, вы должны поместить ее в файл опций.

Чтобы предотвратить неправильное использование, MySQL генерирует ошибку, если вы применяете SET GLOBAL с переменной, которую можно использовать только с SET SESSION, либо если вы не указываете GLOBAL, устанавливая значение глобальной переменной.

Если вы хотите установить переменную сеанса в значение GLOBAL или значение GLOBAL в заранее скомпилированное в MySQL значение по умолчанию, можете устанавливать их в DEFAULT. Например, следующие два оператора идентичны и устанавливают переменную сеанса max\_join\_size в глобальное значение:

```
SET max_join_size=DEFAULT;
SET @@session.max_join_size=@@global.max_join_size;
```

Список большинства системных переменных можно получить оператором SHOW VARIABLES. См. раздел 6.5.3.19. Чтобы получить отдельную переменную по имени, или список переменных с именами, соответствующими шаблону, воспользуйтесь конструкцией LIKE:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW GLOBAL VARIABLES LIKE 'max_join_size';
```

Вы также можете получить значение специфической переменной с помощью синтаксиса @@[global.|local.]имя\_переменной в операторе SELECT:

```
SELECT @@max_join_size, @@global.max_join_size;
```

Когда вы запрашиваете переменную в операторе SELECT @@имя\_переменной (то есть, без указания global., session., local.), MySQL возвращает значение SESSION, если оно существует, и GLOBAL – в противном случае.

В следующем списке представлены переменные, которые имеют нестандартный синтаксис либо не приводятся в списке системных переменных в книге *MySQL. Руководство администратора* (М.: Издательский дом "Вильямс", 2005, ISBN 5-8459-0805-1). Несмотря на то что эти переменные не отображаются через SHOW VARIABLES, вы можете получить их значения с помощью оператора SELECT (за исключением CHARACTER SET или NAMES). Например:

```
mysql> SELECT @@AUTOCOMMIT;
+-----+
| @@autocommit |
+-----+
|             1 |
+-----+
```

- **AUTOCOMMIT** = {0 | 1}. Устанавливает режим автоматического завершения транзакций. Если установлено значение 1, все изменения в данных таблиц вступают в силу немедленно. Если установлено значение 0, вы обязаны завершить транзакцию явно оператором **COMMIT** либо отменить ее с помощью **ROLLBACK**. Если вы переключаете **AUTOCOMMIT** с 0 на 1, MySQL выполняет автоматический **COMMIT** для всех активных транзакций. Другой способ начать транзакцию – это использовать оператор **START TRANSACTION** или **BEGIN**. См. раздел 6.4.1.
- **BIG\_TABLES** = {0 | 1}. Если установлена в 1, все временные таблицы сохраняются на диске вместо того, чтобы сохраняться в памяти. Это немного медленнее, однако при выполнении оператора **SELECT**, который требует создания большой временной таблицы, не возникает ошибка *The table имя\_таблицы is full* (Таблица *имя\_таблицы* переполнена). Значением по умолчанию для новых соединений является 0 (размещать временные таблицы в памяти). Начиная с MySQL 4.0, вам, как правило, не придется устанавливать эту переменную, поскольку MySQL сам в случае необходимости преобразует временные таблицы в памяти в таблицы на диске. Ранее эта переменная называлась **SQL\_BIG\_TABLES**.
- **CHARACTER SET** (*имя\_набора\_символов* | **DEFAULT**). Это передает все строки от клиента и обратно с заданным отображением. До версии MySQL 4.1 единственным допустимым значением для *имя\_набора\_символов* было **cp1551\_koi8**, однако вы можете добавить новые варианты отображения, редактируя файл `sql/convert.cc` исходного дистрибутива MySQL. Начиная с MySQL 4.1, **SET CHARACTER SET** устанавливает три системных переменных: **character\_set\_client** и **character\_set\_results** устанавливаются в заданный набор символов, а **character\_set\_connection** – в значение **character\_set\_database**.

Отображение по умолчанию может быть восстановлено указанием **DEFAULT**.

Следует отметить, что синтаксис **SET CHARACTER SET** отличается от принятого для установки других большинства других опций.
- **FOREIGN\_KEY\_CHECKS** = {0 | 1}. Если установлена в 1 (по умолчанию), ограничения внешних ключей для InnoDB контролируются сервером. Если установлено в 0, игнорируются. Отключение проверок внешних ключей может оказаться удобным для перезагрузки таблиц InnoDB в порядке, отличающемся от того, который требуют их отношения родительский/дочерний. Эта переменная была добавлена в MySQL 3.23.52.
- **IDENTITY** = значение. Эта переменная – синоним **LAST\_INSERT\_ID**. Введена с целью достижения совместимости с другими базами данных. Начиная с MySQL 3.23.25, вы можете получить ее значение с помощью **SELECT @IDENTITY**. Начиная с MySQL 4.0.3, вы также можете установить ее значение через **SET IDENTITY**.
- **INSERT\_ID** = значение. Устанавливает значение, которое будет использовано следующим оператором **INSERT** или **ALTER TABLE** для вставки в столбец **AUTO\_INCREMENT**. В основном это используется с бинарным журналом.
- **LAST\_INSERT\_ID** = значение. Устанавливает значение, которое будет возвращено **LAST\_INSERT\_ID()** в операторах, обновляющих таблицу. Установка этой переменной не обновляет значения, возвращаемого функцией **C API mysql\_insert\_id()**.
- **NAMES** ('*имя\_набора\_символов*' | **DEFAULT**)

SET NAMES устанавливает три системные переменные сеанса: `character_set_client`, `character_set_connection` и `character_set_result` в указанный набор символов. Отображение по умолчанию может быть восстановлено, если указать DEFAULT. Обратите внимание, что синтаксис SET NAMES отличается от принятого для установки других большинства других опций. Этот оператор появился в MySQL 4.1.0.

- `SQL_AUTO_IS_NULL = {0 | 1}`. Если установлена в 1 (по умолчанию), вы можете найти последнюю вставленную в таблицу строку, которая содержит столбец `AUTO_INCREMENT`, используя следующую конструкцию:

`WHERE столбец_auto_increment IS NULL`

Это поведение характерно для некоторых ODBC-программ, таких как Access. `SQL_AUTO_IS_NULL` была добавлена в MySQL 3.23.52.

- `SQL_BIG_SELECTS = {0 | 1}`. Если установлена в 0, MySQL прерывает выполнение операторов SELECT, которые предположительно будут выполняться долго (го есть операторов, для которых оптимизатор ожидает, что количество проверяемых строк превысит значение `max_join_size`). Это удобно, когда применяются нерациональные условия в конструкции WHERE. Значение по умолчанию этой переменной для новых соединений равно 1, что позволяет выполнять любые операторы SELECT. Если вы устанавливаете значение системной переменной `max_join_size` в значение, отличное от DEFAULT, `SQL_BIG_SELECTS` принимает значение 0.
- `SQL_BUFFER_RESULT = {0 | 1}`  
`SQL_BUFFER_RESULT` принуждает операторы SELECT помещать результаты во временные таблицы. Это помогает MySQL освобождать табличные блокировки пораньше и может быть выгодно в случаях, когда требуется много времени для отправки результатов клиентам. Переменная была добавлена в MySQL 3.23.13.
- `SQL_LOG_BIN = {0 | 1}`. Если установлена в 0, никакой бинарной регистрации для клиента не выполняется. Клиент должен иметь привилегию SUPER для установки этой опции. Переменная была добавлена в MySQL 3.23.16.
- `SQL_LOG_OFF = {0 | 1}`. Если установлена в 1, регистрация запросов клиента в общем журнале не ведется. Клиент должен иметь привилегию SUPER для установки этой опции.
- `SQL_LOG_UPDATE = {0 | 1}`. Если установлена в 0, регистрация обновлений клиента в журнале не ведется. Клиент должен иметь привилегию SUPER для установки этой опции. Переменная была добавлена в MySQL 3.23.5. Начиная с MySQL 5.0.0, считается устаревшей и отображается на `SQL_LOG_BIN`.
- `SQL_QUOTE_SHOW_CREATE = {0 | 1}`. Если установлена в 1, SHOW CREATE TABLE выводит имена таблиц и столбцов в кавычках. Если установлена в 0, кавычки отключаются. По умолчанию эта опция включена, чтобы репликация работала с таблицами и столбцами в кавычках. Переменная появилась в MySQL 3.23.26. См. раздел 6.5.3.6.
- `SQL_SAFE_UPDATES = {0 | 1}`. Если установлена в 1, MySQL прерывает операторы UPDATE и DELETE, у которых не используются ключевые значения в конструкции WHERE. Это позволяет перехватывать операторы UPDATE и DELETE, у которых неправильно используются ключи и которые могут непреднамеренно изменить или удалить большое количество строк. Переменная была добавлена в MySQL 3.22.32.



- `SQL_SELECT_LIMIT` = {значение | DEFAULT}. Максимальное количество строк, которые может вернуть оператор `SELECT`. Значение по умолчанию для новых подключений – “неограниченное” (unlimited). Если вы изменяете это ограничение, значение по умолчанию может быть восстановлено присвоением `SQL_SELECT_LIMIT` значения `DEFAULT`.
- `SQL_WARNINGS` = {0 | 1}. Эта переменная устанавливает режим вывода предупреждений однострочными операторами `INSERT`. По умолчанию имеет значение 0. Установите ее равной 1, чтобы генерировать строки предупреждений. Переменная была добавлена в MySQL 3.22.11.
- `TIMESTAMP` = {значение\_timestamp | DEFAULT}. Устанавливает текущее время для клиента. Применяется для получения оригинальной временной метки при использовании бинарного журнала для восстановления строк. значение\_timestamp должно быть временной меткой в формате Unix, а не временной меткой MySQL.
- `UNIQUE_CHECKS` = {0 | 1}. Если установлена в 1 (как по умолчанию), выполняется проверка уникальности вторичных индексов в таблицах InnoDB. Если установлена в 0, никаких проверок уникальности не выполняется. Переменная появилась в MySQL 3.23.52.

### 6.5.3.2. Синтаксис SHOW CHARACTER SET

`SHOW CHARACTER SET [LIKE 'шаблон']`

Оператор `SHOW CHARACTER SET` выводит список всех допустимых символьных наборов. Он принимает необязательную конструкцию `LIKE` для указания шаблона имен наборов символов. Например:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+-----+-----+-----+-----+
| Charset | Description | Default collation | Maxlen |
+-----+-----+-----+-----+
| latin1  | ISO 8859-1 West European | latin1_swedish_ci | 1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci | 1 |
| latin5  | ISO 8859-9 Turkish | latin5_turkish_ci | 1 |
| latin7  | ISO 8859-13 Baltic | latin7_general_ci | 1 |
+-----+-----+-----+-----+
```

Столбец `Maxlen` показывает максимальное число байт, необходимых для хранения одного символа. `SHOW CHARACTER SET` доступен, начиная с версии MySQL 4.1.0.

### 6.5.3.3. Синтаксис SHOW COLLATION

`SHOW COLLATION [LIKE 'шаблон']`

Вывод `SHOW COLLATION` включает все допустимые порядки сопоставления наборов символов. Принимает необязательную конструкцию `LIKE` для указания шаблона наименований порядка сопоставления. Например:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-----+-----+-----+-----+-----+-----+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+-----+
| latin1_german1_ci | latin1 | 5 | | | 0 |
```

latin1_swedish_ci	latin1	8	Yes	Yes	0
latin1_danish_ci	latin1	15			0
latin1_german2_ci	latin1	31		Yes	2
latin1_bin	latin1	47		Yes	0
latin1_general_ci	latin1	48			0
latin1_general_cs	latin1	49			0
latin1_spanish_ci	latin1	94			0
-----	-----	-----	-----	-----	-----

Столбец Default показывает, является ли данный порядок сопоставления порядком по умолчанию для данного набора символов. Compiled отражает, является ли данный набор символом заранее скомпилированным на сервере. Sortlen имеет отношение к объему памяти, необходимому для сортировки строк, представленных в данном наборе символов.

SHOW COLLATION доступен, начиная с версии MySQL 4.1.0.

#### 6.5.3.4. Синтаксис SHOW COLUMNS

SHOW [FULL] COLUMNS FROM имя\_таблицы [FROM имя\_базы\_данных] [LIKE 'шаблон']

SHOW COLUMNS выводит список столбцов заданной таблицы. Если типы столбцов отличаются от тех, что были заданы оператором CREATE TABLE, имейте в виду, что MySQL иногда изменяет типы столбцов при создании или изменении структуры таблицы. Условия, при которых это происходит, описаны в разделе 6.2.5.2.

Ключевое слово FULL может использоваться, начиная с MySQL 3.23.32 и выше. Оно включает в вывод информацию о привилегиях, которые вы имеете для доступа к данным столбцам. Начиная с MySQL 4.1, слово FULL также включает вывод всех комментариев о столбцах.

Вы можете использовать имя\_базы\_данных.имя\_таблицы как альтернативу синтаксису имя\_таблицы FROM имя\_базы\_данных. Следующие два оператора эквивалентны:

```
mysql> SHOW COLUMNS FROM mytable FROM mydb;
mysql> SHOW COLUMNS FROM mydb.mytable;
```

SHOW FIELDS — это синоним SHOW COLUMNS. Вы также можете вывести список столбцов с помощью команды mysqlshow имя\_базы\_данных имя\_таблицы.

Оператор DESCRIBE выводит информацию, подобную SHOW COLUMNS. См. раздел 6.3.1.

#### 6.5.3.5. Синтаксис SHOW CREATE DATABASE

SHOW CREATE DATABASE имя\_базы\_данных

Выводит оператор CREATE DATABASE, который создаст указанную базу данных. Добавлен в MySQL 4.1.

```
mysql> SHOW CREATE DATABASE test\G
***** 1. row *****
Database: test
Create Database: CREATE DATABASE `test`
/*!40100 DEFAULT CHARACTER SET latin1 */
```

#### 6.5.3.6. Синтаксис SHOW CREATE TABLE

SHOW CREATE TABLE имя\_таблицы

Выводит оператор `CREATE TABLE`, который создаст указанную таблицу. Добавлен в MySQL 3.23.20.

```
mysql> SHOW CREATE TABLE t\G
***** 1. row *****
      Table: t
Create Table: CREATE TABLE t (
  id INT(11) default NULL auto_increment,
  s char(60) default NULL,
  PRIMARY KEY (id)
) TYPE=MyISAM
```

`SHOW CREATE TABLE` заключает в кавычки имена таблицы и столбцов в соответствии со значением опции `SQL_QUOTE_SHOW_CREATE`. См. раздел 6.5.3.1.

### 6.5.3.7. Синтаксис SHOW DATABASES

```
SHOW DATABASES [LIKE 'шаблон']
```

`SHOW DATABASES` выводит список баз данных на хосте сервера MySQL. Вы также можете получить этот список с помощью команды `mysqlshow`. Начиная с MySQL 4.0.2, вы увидите только те базы данных, для доступа к которым имеете какие-либо привилегии, если только у вас нет глобальной привилегии `SHOW DATABASES`.

Если сервер запущен с опцией `--skip-show-database`, вы не сможете использовать этот оператор вообще, если не имеете привилегии `SHOW DATABASES`.

### 6.5.3.8. Синтаксис SHOW ENGINES

```
SHOW [STORAGE] ENGINES
```

`SHOW ENGINES` выводит информацию о состоянии механизмов хранения. Это особенно удобно для проверки того, какие механизмы хранения поддерживаются сервером, или для того, чтобы увидеть, какой механизм используется по умолчанию. Оператор реализован в MySQL 4.1.2. Существует устаревший синоним — `SHOW TABLE TYPES`.

```
mysql> SHOW ENGINES\G
***** 1. row *****
      Type: MyISAM
Support: DEFAULT
Comment: Default type from 3.23 with great performance
***** 2. row *****
      Type: HEAP
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
***** 3. row *****
      Type: MEMORY
Support: YES
Comment: Alias for HEAP
***** 4. row *****
      Type: MERGE
Support: YES
Comment: Collection of identical MyISAM tables
***** 5. row *****
```

```

Type: MRG_MYISAM
Support: YES
Comment: Alias for MERGE
***** 6. row *****
Type: ISAM
Support: NO
Comment: Obsolete table type; Is replaced by MyISAM
***** 7. row *****
Type: MRG_ISAM
Support: NO
Comment: Obsolete table type; Is replaced by MRG_MYISAM
***** 8. row *****
Type: InnoDB
Support: YES
Comment: Supports transactions, row-level locking and foreign keys
***** 9. row *****
Type: INNODB
Support: YES
Comment: Alias for INNODB
***** 10. row *****
Type: BDB
Support: YES
Comment: Supports transactions and page-level locking
***** 11. row *****
Type: BERKELEYDB
Support: YES
Comment: Alias for BDB

```

Значение `Support` показывает, поддерживается ли данный механизм сервером, и какой из них является механизмом по умолчанию. Например, если сервер запущен с опцией `--default-table-type=InnoDB`, то значением `Support` для `InnoDB` будет `DEFAULT`.

### 6.5.3.9. Синтаксис SHOW ERRORS

```

SHOW ERRORS [LIMIT [смещение,] количество_строк]
SHOW COUNT(*) ERRORS

```

Этот оператор похож на `SHOW WARNINGS`, с тем отличием, что вместо отображения ошибок, предупреждений и примечаний, он выводит только ошибки. `SHOW ERRORS` доступен, начиная с версии MySQL 4.1.0.

Конструкция `LIMIT` имеет тот же синтаксис, что и в операторе `SELECT`. См. раздел 6.1.7.

Оператор `SHOW COUNT(*) ERRORS` отображает количество ошибок. Вы также можете получить это количество из переменной `error_count`:

```

SHOW COUNT(*) ERRORS;
SELECT @@error_count;

```

Более подробную информацию можно получить в разделе 6.5.3.20.

### 6.5.3.10. Синтаксис SHOW GRANTS

```

SHOW GRANTS FOR пользователь

```

Этот оператор выводит операторы GRANT, которые должны быть выполнены для дублирования набора привилегий пользователя MySQL.

```
mysql> SHOW GRANTS FOR 'root'@'localhost';
+-----+
| Grants for root@localhost |
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+-----+
```

Начиная с версии MySQL 4.1.2, чтобы получить список привилегий текущего сеанса, можно воспользоваться любым из перечисленных ниже операторов:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

До MySQL 4.1.2 узнать, какой пользователь был аутентифицирован в текущем сеансе, можно было с помощью функции CURRENT\_USER() (новая в MySQL 4.0.6). Затем полученное значение нужно было указать в операторе SHOW GRANTS. См. раздел 5.8.3.

SHOW GRANTS доступен, начиная с MySQL 3.23.4.

### 6.5.3.11. Синтаксис SHOW INDEX

SHOW INDEX FROM *имя\_таблицы* [FROM *имя\_базы\_данных*]

SHOW INDEX возвращает информацию об индексах таблицы в формате, подобном тому, что выдает вызов SQLStatistics в ODBC.

SHOW INDEX возвращает следующие поля:

- Table. Имя таблицы.
- Non\_unique. 0, если индекс не может иметь дублированных ключей, 1 – если может.
- Key\_name. Имя индекса.
- Seq\_in\_index. Порядок столбца в ключе индекса, начиная с 1.
- Column\_name. Имя столбца.
- Collation. Как столбец сортируется в индексе. В MySQL может иметь значения 'A' (ascending – по возрастанию) или NULL (не сортировано).
- Cardinality. Количество уникальных значений в индексе. Это обновляется оператором ANALYZE TABLE или командой myisamchk -a. Cardinality рассчитывается на базе статистики, хранится в виде целого числа, поэтому нет необходимости в большой точности для маленьких таблиц.
- Sub\_part. Количество проиндексированных символов столбца, если ключ построен на части столбца. Если вся столбец является ключом индекса, принимает значение NULL.
- Packed. Показывает, упакован ли индекс. Если нет – NULL.
- Null. Содержит YES, если столбец допускает значение NULL.
- Index\_type. Использованный метод индексации (BTREE, FULLTEXT, HASH, RTREE).
- Comment. Различные замечания. До MySQL 4.0.2, когда был добавлен столбец Index\_type, Comment показывает, является ли индекс FULLTEXT.

Столбцы Packed и Comment появились в версии MySQL 3.23.0. Столбцы Null и Index\_type были добавлены в MySQL 4.0.2.

В качестве альтернативы синтаксиса `имя_таблицы FROM имя_базы_данных` можно использовать `имя_базы_данных.имя_таблицы`. Следующие два оператора эквивалентны:

```
mysql> SHOW INDEX FROM mytable FROM mydb;
mysql> SHOW INDEX FROM mydb.mytable;
```

SHOW KEYS – синоним для SHOW INDEX. Список индексов можно также получить по команде `mysqlshow -k имя_базы_данных имя_таблицы`.

### 6.5.3.12. Синтаксис SHOW INNODB STATUS

SHOW INNODB STATUS

Этот оператор показывает подробную информацию о состоянии механизма хранения InnoDB.

### 6.5.3.13. Синтаксис SHOW LOGS

SHOW [BDB] LOGS

SHOW LOGS выводит информацию о существующих журнальных файлах. Оператор был реализован в MySQL 3.23.29. В настоящий момент он отображает только информацию о журнальных файлах Berkley DB, поэтому псевдонимом для него (доступным с MySQL 4.1.1) является SHOW BDB LOGS.

SHOW LOGS возвращает следующие поля:

- File. Полный путь к файлу журнала.
- Type. Тип файла журнала (BDB для журналов Berkley DB).
- Status. Состояние файла журнала (FREE, если файл может быть удален, или IN USE, если файл необходим подсистеме обработки транзакций).

### 6.5.3.14. Синтаксис SHOW PRIVILEGES

SHOW PRIVILEGES

SHOW PRIVILEGES выводит список системных привилегий, поддерживаемых сервером MySQL. Этот оператор реализован в MySQL 4.1.0.

```
mysql> SHOW PRIVILEGES\G
***** 1. row *****
Privilege: Select
Context: Tables
Comment: To retrieve rows from table
***** 2. row *****
Privilege: Insert
Context: Tables
Comment: To insert data into tables
***** 3. row *****
Privilege: Update
Context: Tables
Comment: To update existing rows
***** 4. row *****
Privilege: Delete
```

```

Context: Tables
Comment: To delete existing rows
***** 5. row *****
Privilege: Index
Context: Tables
Comment: To create or drop indexes
***** 6. row *****
Privilege: Alter
Context: Tables
Comment: To alter the table
***** 7. row *****
Privilege: Create
Context: Databases, Tables, Indexes
Comment: To create new databases and tables
***** 8. row *****
Privilege: Drop
Context: Databases, Tables
Comment: To drop databases and tables
***** 9. row *****
Privilege: Grant
Context: Databases, Tables
Comment: To give to other users those privileges you possess
***** 10. row *****
Privilege: References
Context: Databases, Tables
Comment: To have references on tables
***** 11. row *****
Privilege: Reload
Context: Server Admin
Comment: To reload or refresh tables, logs and privileges
***** 12. row *****
Privilege: Shutdown
Context: Server Admin
Comment: To shutdown the server
***** 13. row *****
Privilege: Process
Context: Server Admin
Comment: To view the plain text of currently executing queries
***** 14. row *****
Privilege: File
Context: File access on server
Comment: To read and write files on the server

```

### 6.5.3.15. Синтаксис SHOW PROCESSLIST

SHOW [FULL] PROCESSLIST

SHOW PROCESSLIST выводит список работающих потоков сервера. Эту же информацию можно получить командой `mysqladmin processlist`. Если у вас есть привилегия SUPER, вы можете видеть все потоки, в противном случае – только свои собственные (то есть потоки, ассоциированные с учетной записью пользователя MySQL, под которой вы

вошли). См. раздел 6.5.4.3. Если не указано ключевое слово `FULL`, выводятся только первые 100 символов каждого запроса.

Начиная с MySQL 4.0.12, этот оператор сообщает имена хостов для подключений TCP/IP в формате `имя_хоста:порт_клиента`, чтобы можно было понять, какой клиент что делает.

Этот оператор очень удобен, если вы получаете сообщение об ошибке наподобие “слишком много подключений” и хотите разобраться в том, что происходит. MySQL резервирует дополнительное подключение для учетной записи с привилегией `SUPER`, чтобы всегда давать возможность администратору подключиться и проверить систему (предполагается, что вы не даете эту привилегию всем пользователям).

Ниже описаны некоторые состояния, которые обычно можно увидеть в выводе `SHOW PROCESSLIST`:

- **Checking table.** Поток выполняет (автоматическую) проверку таблицы.
- **Closing table.** Означает, что поток сбрасывает измененные данные таблицы на диск и закрывает используемые таблицы. Это должно быть быстрой операцией. Если же нет, вам стоит проверить, не переполнен ли диск и не слишком ли интенсивно он используется.
- **Connect Out.** Подчиненный сервер репликации подключается к главному.
- **Copying to tmp table on disk.** Временный результирующий набор оказался больше `tmp_table_size` и поток переносит временную таблицу из памяти на диск для экономии памяти.
- **Creating tmp file.** Поток создает временный файл для сохранения части результата запроса.
- **Deleting from main table.** Поток выполняет первую часть операции многотабличного удаления, удаляя данные из первой таблицы.
- **Deleting from reference table.** Поток выполняет вторую часть операции многотабличного удаления, удаляя соответствующие строки из других таблиц.
- **Flushing tables.** Поток выполняет `FLUSH TABLES` и ожидает, когда все потоки закроют свои таблицы.
- **Killed.** Кто-то послал команду прерывания потока, и он должен быть прерван при следующей проверке флага удаления потока. Этот флаг проверяется на каждом шаге основного цикла MySQL, но иногда это может потребовать некоторого времени на прерывание потока. Если поток заблокирован каким-то другим потоком, прерывание произойдет немедленно после освобождения блокировки.
- **Sending data.** Поток обрабатывает строки для оператора `SELECT` и отправляет их клиенту.
- **Sorting for group.** Поток выполняет сортировку для удовлетворения условию `GROUP BY`.
- **Sorting for order.** Поток выполняет сортировку для удовлетворения условию `ORDER BY`.
- **Opening tables.** Поток пытается открыть таблицу. Это должно быть очень быстрой операцией, если только ничто не мешает открытию. Например, операторы



ALTER TABLE или LOCK TABLE могут предотвратить открытие таблиц до своего завершения.

- Removing duplicates. Запрос использовал SELECT DISTINCT таким образом, что MySQL не может оптимизировать операцию DISTINCT на ранней стадии. Из-за этого MySQL потребовалась дополнительная стадия для удаления всех дублированных строк из результирующего набора перед отправкой его клиенту.
- Reopen table. Поток получил блокировку таблицы, но после этого был извещен о том, что структура таблицы изменилась. Он освобождает блокировку, закрывает таблицу, а затем вновь открывает ее.
- Repair by sorting. Код восстановления использует сортировку для создания индексов.
- Repair with keycache. Код восстановления использует создание ключей одного за другим по кэшу ключей. Это происходит намного медленнее, чем Repair by sorting.
- Searching rows for update. Поток выполняет первую фазу поиска всех подлежащих обновлению строк. Это должно быть сделано, если UPDATE изменяет значения ключа индекса, который используется для поиска изменяемых строк.
- Sleeping. Поток ожидает нового оператора от клиента.
- System lock. Поток ожидает получения внешней системной блокировки таблицы. Если вы не используете сразу несколько серверов mysqld на одной машине, которые осуществляют доступ к одним и тем же таблицам, можете отключить внешние блокировки, указав опцию --skip-external-locking.
- Upgrading lock. Обработчик отложенных вставок INSERT DELAYED пытается получить блокировку таблицы для вставки строк.
- Updating. Поток ищет строки для обновления и обновляет их.
- User lock. Поток ожидает возврата GET\_LOCK().
- Waiting for tables. Поток получил извещение о том, что структура таблицы изменилась и нуждается в повторном открытии таблицы, чтобы получить новую структуру. Однако чтобы быть готовым повторно открыть таблицу, поток вынужден ожидать, когда все остальные потоки закроют таблицу.

Такое извещение поступает, когда другой поток использует FLUSH TABLES, либо в обработке находится один из следующих операторов: FLUSH TABLES *имя\_таблицы*, ALTER TABLE, RENAME TABLE, REPAIR TABLE, ANALYZE TABLE или OPTIMIZE TABLE.

- Waiting for handler insert. Обработчик отложенных вставок INSERT DELAYED обработал все отложенные вставки и ожидает новых.

Большинство состояний соответствуют очень быстрым операциям. Если поток останавливается в одном из этих состояний на много секунд, это может говорить о проблеме, которую необходимо исследовать.

Существует ряд других состояний, не перечисленных в предыдущем списке, однако многие из них применяются только при поиске ошибок в коде сервера.

### 6.5.3.16. Синтаксис SHOW STATUS

SHOW STATUS [LIKE '*шаблон*']

SHOW STATUS выводит информацию о состоянии сервера. Эту же информацию можно получить командой `mysqladmin extended-status`.

Частичный вывод представлен ниже. На вашем сервере список переменных и их значений может отличаться. Смысл каждой переменной подробно описан в книге *MySQL. Руководство администратора* (М. : Издательский дом "Вильямс", 2005, ISBN 5-8459-0805-1).

```
mysql> SHOW STATUS;
```

Variable_name	Value
Aborted_clients	0
Aborted_connects	0
Bytes_received	155372598
Bytes_sent	1176560426
Connections	30023
Created_tmp_disk_tables	0
Created_tmp_tables	8340
Created_tmp_files	60
...	
Open_tables	1
Open_files	2
Open_streams	0
Opened_tables	44600
Questions	2026873
...	
Table_locks_immediate	1920382
Table_locks_waited	0
Threads_cached	0
Threads_created	30022
Threads_connected	1
Threads_running	1
Uptime	80380

С конструкцией LIKE оператор выводит только те переменные, имена которых соответствуют шаблону:

```
mysql> SHOW STATUS LIKE 'Key%';
```

Variable_name	Value
Key_blocks_used	14955
Key_read_requests	96854827
Key_reads	162040
Key_write_requests	7589728
Key_writes	3813196

### 6.5.3.17. Синтаксис SHOW TABLE STATUS

```
SHOW TABLE STATUS [FROM имя_базы_данных] [LIKE 'шаблон']
```

SHOW TABLE STATUS (новый оператор в MySQL 3.23) работает подобно SHOW TABLE, но выводит большой объем информации о каждой таблице. Вы можете получить этот список также с помощью команды `mysqlshow --status имя_базы_данных`.

SHOW TABLE STATUS возвращает следующие поля:

- Name. Имя таблицы.
- Type. Тип таблицы.
- Row\_format. Формат хранения строки (Fixed, Dynamic или Compressed).
- Rows. Количество строк.
- Avg\_row\_length. Средняя длина строки.
- 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 это общее число байт данных, которые можно поместить в таблицу, учитывая размер используемого указателя данных.
- Index\_length. Длина индексного файла.
- Data\_free. Количество выделенных, но не используемых байт.
- Auto\_increment. Следующее значение AUTO\_INCREMENT.
- Create\_time. Время создания таблицы.
- Update\_time. Время обновления файла данных.
- Check\_time. Когда таблица проверялась в последний раз.
- Collation. Набор символов и порядок сопоставления таблицы (новое в 4.1.1).
- Checksum. Актуальная контрольная сумма (если есть) (новое в 4.1.1).
- Create\_options. Дополнительные опции, использованные в CREATE TABLE.
- Comment. Комментарий, введенный при создании таблицы (или некоторая информация о том, почему MySQL не может получить доступ к данным в таблице).

В комментариях к таблице InnoDB сообщает о свободном месте в табличном пространстве, которому принадлежит таблица. Для таблиц, расположенных в разделяемом табличном пространстве (shared tablespace), это свободное место в разделяемом табличном пространстве. Если вы используете множественные табличные пространства, и таблица имеет свое собственное табличное пространство, отображается свободное пространство только этой таблицы.

Для таблиц типа MEMORY (HEAP) значение Data\_length, Max\_data\_length и Index\_length принимают примерное значение объема выделенной памяти. Алгоритм распределения памяти резервирует память большими кусками, чтобы уменьшить количество необходимых операций.

### 6.5.3.18. Синтаксис SHOW TABLES

SHOW [OPEN] TABLES [FROM имя\_базы\_данных] [LIKE 'шаблон']

SHOW TABLES выводит список всех постоянных (не временных) таблиц заданной базы данных. Этот же список можно получить командой `mysqlshow имя_базы_данных`.

**На заметку!**

Если у вас нет привилегий в таблице, эта таблица не будет присутствовать в списке.

SHOW TABLES выводит список таблиц, которые в данный момент открыты в табличном кэше. Поле Comment в списке говорит о том, сколько раз таблица была cached и in\_use.

Слово OPEN может использоваться, начиная с MySQL 3.23.33 и в последующих версиях.

**6.5.3.19. Синтаксис SHOW VARIABLES**

SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'шаблон']

SHOW VARIABLES выводит значения некоторых системных переменных MySQL. Эта информация также может быть получена командой mysqladmin variables.

Опции GLOBAL и SESSION появились в MySQL 4.0.3. Указав GLOBAL, вы получите значения, которые будут использоваться новыми подключениями к MySQL. Задав SESSION, вы получите значения, действующие в текущем сеансе. Если не указывать ни одну из этих опций, по умолчанию принимается SESSION. LOCAL — это синоним для SESSION.

Если значения по умолчанию вам не подходят, можете установить значения большинства из этих переменных, используя опции командной строки при запуске mysqld, или же во время выполнения с помощью оператора SET. См. раздел 6.5.3.1.

Частичный вывод представлен ниже. Список переменных и их значения могут отличаться на вашем сервере. Описание каждой переменной можно найти в книге *MySQL. Руководство администратора* (М.: Издательский дом "Вильямс", 2005, ISBN 5-8459-0805-1).

```
mysql> SHOW VARIABLES;
```

Variable_name	Value
back_log	50
basedir	/usr/local/mysql
bdb_cache_size	8388572
bdb_log_buffer_size	32768
bdb_home	/usr/local/mysql
...	
max_connections	100
max_connect_errors	10
max_delayed_threads	20
max_error_count	64
max_heap_table_size	16777216
max_join_size	4294967295
max_relay_log_size	0
max_sort_length	1024
...	
timezone	EEST
tmp_table_size	33554432
tmpdir	/tmp:/mnt/hd2/tmp/
version	4.0.4-beta
wait_timeout	28800

С конструкцией LIKE оператор выводит только те переменные, имена которых соответствуют шаблону:

```
mysql> SHOW VARIABLES LIKE 'have%';
```

Variable_name	Value
have_bdb	YES
have_innodb	YES
have_isam	YES
have_raid	NO
have_symlink	DISABLED
have_openssl	YES
have_query_cache	YES

### 6.5.3.20. Синтаксис SHOW WARNINGS

```
SHOW WARNINGS [LIMIT [смещение,] количество_строк]
SHOW COUNT(*) WARNINGS
```

SHOW WARNINGS выводит ошибки, предупреждения и другие замечания, которые появляются в результате выполнения последнего оператора, который генерирует сообщения, или же ничего, если последний оператор, использующий таблицу, не сгенерировал никаких сообщений. Этот оператор реализован в MySQL 4.1.0. Родственный оператор — SHOW ERRORS — показывает только ошибки. См. раздел 6.5.3.9.

Список сообщений очищается для каждого нового оператора, использованного в таблице.

Оператор SHOW COUNT(\*) WARNINGS отображает общее количество ошибок, предупреждений и замечаний. Вы можете прочитать это значение в переменной warning\_count:

```
SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;
```

Значение warning\_count может больше, чем количество сообщений, отображенное SHOW WARNINGS, если значение системной переменной max\_error\_count установлено достаточно малым, то есть так, что не все сообщения сохраняются. Приведенный ниже пример показывает, как это может случиться.

Конструкция LIMIT имеет тот же синтаксис, что и в операторе SELECT. См. раздел 6.1.7.

Сервер MySQL отправляет обратно общее количество ошибок, предупреждений и замечаний, полученных в результате последнего оператора. Если вы используете программный интерфейс C API, это значение может быть получено вызовом mysql\_warning\_count().

Отметим, что подсистема предупреждений появилась в MySQL 4.1.0, когда многие операторы еще не генерировали предупреждений. В версии MySQL 4.1.1 ситуация с предупреждениями, генерируемыми такими операторами, как LOAD DATA INFILE, а также операторами DML (языка манипуляции данными), подобными INSERT, UPDATE, CREATE TABLE и ALTER TABLE (в других источниках принято считать, что CREATE TABLE и ALTER TABLE — это операторы DDL (Data Definition Language — язык определения данных), а не DML — прим. пер.), значительно улучшилась.

Следующий оператор DROP TABLE генерирует сообщение:

```
mysql> DROP TABLE IF EXISTS no_such_table;
mysql> SHOW WARNINGS;
```

```

+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Note | 1051 | Unknown table 'no_such_table' |
+-----+-----+-----+

```

Ниже представлен простой пример, показывающий синтаксис предупреждений CREATE TABLE и замену предупреждения после оператора INSERT:

```

mysql> CREATE TABLE t1 (a TINYINT NOT NULL, b CHAR(4)) TYPE=MyISAM;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1287
Message: 'TYPE=storage_engine' is deprecated, use
        'ENGINE=storage_engine' instead
1 row in set (0.00 sec)

mysql> INSERT INTO t1 VALUES(10, 'mysql'), (NULL, 'test'),
-> (300, 'open source');
Query OK, 3 rows affected, 4 warnings (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 4

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1265
Message: Data truncated for column 'b' at row 1
***** 2. row *****
Level: Warning
Code: 1263
Message: Data truncated, NULL supplied to NOT NULL column 'a' at row 2
***** 3. row *****
Level: Warning
Code: 1264
Message: Data truncated, out of range for column 'a' at row 3
***** 4. row *****
Level: Warning
Code: 1265
Message: Data truncated for column 'b' at row 3
4 rows in set (0.00 sec)

```

Максимальное количество ошибок, сообщений и замечаний, которые сохраняются между вызовами операторов, определены значением системной переменной `max_error_count`. По умолчанию оно равно 64. Чтобы изменить количество сообщений, которые нужно сохранять, просто измените значение этой переменной. В следующем примере оператор ALTER TABLE генерирует три предупреждающих сообщения, но сохраняется только одно, потому что `max_error_count` присвоено значение 1:

```

mysql> SHOW VARIABLES LIKE 'max_error_count';
+-----+-----+
| Variable_name | Value |
+-----+-----+

```

```

+-----+-----+
| max_error_count | 64      |
+-----+-----+
1 row in set (0.00 sec)

mysql> SET max_error_count=1;
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t1 MODIFY b CHAR;
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 3

mysql> SELECT @@warning_count;
+-----+
| @@warning_count |
+-----+
|                3 |
+-----+
1 row in set (0.01 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1263 | Data truncated for column 'b' at row 1 |
+-----+-----+-----+
1 row in set (0.00 sec)

```

Чтобы отключить сообщения вообще, присвойте 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.

## 6.5.4 Другие операторы администрирования

### 6.5.4.1. Синтаксис CACHE INDEX

CACHE INDEX

список\_индексов\_таблицы [, список\_индексов\_таблицы] ...  
IN имя\_кэша\_ключей

список\_индексов\_таблицы:

имя\_таблицы [[INDEX] (имя\_индекса [, имя\_индекса] ...)]

Оператор `CACHE INDEX` назначает индексам таблиц специальный кэш ключей. Это действительно только для таблиц `MyISAM`.

Следующий оператор назначает индексам таблиц `t1`, `t2` и `t3` ключевой кэш с именем `hot_cache`:

```

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       |
+-----+-----+-----+-----+

```

Синтаксис `CACHE INDEX` позволяет привязать к кэшу только отдельные индексы таблицы. Однако на самом деле текущая реализация привязывает все индексы таблицы к указанному кэшу, поэтому нет особого смысла специфицировать что-то еще, кроме имени таблицы.

Кэш ключей, на который ссылается оператор `CACHE INDEX`, может быть создан установкой его размера в параметре оператора `SET` либо в настройках параметров сервера. Например:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

Параметры кэша ключей доступны как члены структурированной системной переменной (см. раздел 2.4.1).

Кэш ключей должен существовать до привязки индексов к нему:

```
mysql> CACHE INDEX t1 in non_existent_cache;  
ERROR 1283 (HY000): Unknown key cache 'non_existent_cache'
```

По умолчанию индексы таблиц привязываются к главному (по умолчанию) кэшу ключей, который создается при запуске сервера. Когда кэш ключей разрушается, все назначенные ему индексы опять переназначаются на кэш по умолчанию.

Назначение индексов касается сервера глобально. Если один клиент назначает индекс определенному кэшу, этот кэш будет использоваться при выполнении всех запросов, независимо от того, какой клиент их инициирует.

`CACHE INDEX` появился в MySQL 4.1.1.

### 6.5.4.2. Синтаксис `FLUSH`

`FLUSH [LOCAL | NO_WRITE_TO_BINLOG] опция_очистки [, опция_очистки] ...`

Оператор `FLUSH` следует применять, когда вы хотите очистить внутренние кэши, которые использует MySQL. Для выполнения `FLUSH` необходимо иметь привилегию `RELOAD`.

Аргумент *опция\_очистки* может принимать любое из перечисленных ниже значений.

- **HOSTS.** Очищает таблицы кэша хостов. Вы должны сбрасывать таблицы хостов, если какой-нибудь из ваших хостов меняет IP-адрес, либо если вы получаете сообщение об ошибке `Host ... is blocked` (Хост ... заблокирован). Когда случается больше, чем `max_connect_errors` при попытке подключиться к серверу MySQL с данного хоста, MySQL предполагает, что с хостом что-то не в порядке и блокирует дальнейшие попытки его подключения. Сброс кэша хостов опять снимает эту блокировку и позволяет продолжить попытки подключения. Вы можете запустить сервер `mysqld` с опцией `--max_connect_errors=999999999`, чтобы избежать ошибок подобного рода.
- **DES\_KEY\_FILE.** Перезагружает DES-ключи из файла, заданного опцией `--des-key-file` во время запуска сервера.
- **LOGS.** Закрывает и заново открывает все журнальные файлы. Если вы указали имена журнальных файлов изменений или бинарного журнала без расширения, номер в расширении имени будет увеличен на единицу по сравнению с предыдущим файлом. Если же вы укажете имена с расширениями, MySQL просто закроет и откроет файл журнала обновлений или бинарного журнала. В среде Unix это то же самое, что послать сигнал `SIGHUP` серверу `mysqld`.



- **PRIVILEGES.** Перезагружает в память информацию о привилегиях из таблиц привилегий базы данных `mysql`.
- **QUERY CACHE.** Выполняет дефрагментацию кэша запросов с целью более эффективного использования памяти. В отличие от `RESET QUERY CACHE`, этот оператор не удаляет никаких запросов из кэша
- **STATUS.** Сбрасывает большинство переменных состояния в ноль. Это то, что стоит применять только при отладке запросов. См. раздел 1.7.1.3.
- **{TABLE | TABLES} [имя\_таблицы [, имя\_таблицы] ...]**  
Когда не названа таблица, закрывает все открытые таблицы и принуждает к закрытию все используемые таблицы. Также очищает кэш запросов. С одним или более именем таблицы сбрасывает только эти таблицы. `FLUSH TABLES` также удаляет все результаты запросов из кэша, как и оператор `RESET QUERY CACHE`.
- **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 `UNLOCK TABLES`. Это очень удобный способ получить резервную копию, если вы пользуетесь такой файловой системой, как Veritas, которая позволяет делать снимки по времени.
- **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. См. раздел 6.5.1.2.

До версии MySQL 4.1.1 операторы `FLUSH` не записывались в бинарный журнал. Начиная с MySQL 4.1.1, этот оператор записывается, если только не было указано необязательное ключевое слово `NO_WRITE_TO_BINLOG` (или его псевдоним `LOCAL`). Исключениями являются `FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE` и `FLUSH TABLES WITH READ LOCK`, которые не регистрируются в любом случае, поскольку они могут быть причинами проблем при репликации на подчиненный сервер.

Вы можете получить доступ к некоторым из этих операторов с помощью утилиты `mysqladmin`, используя команды `flushhosts`, `flush-logs`, `flush-privileges`, `flush-status` или `flush-tables`.

Взгляните также на оператор `RESET`, используемый при репликации (см. раздел 6.5.4.5).

### 6.5.4.3. Синтаксис KILL

`KILL [CONNECTION | QUERY] идентификатор_потока`

Каждое подключение к `mysqld` запускает отдельный поток. Вы можете просмотреть все выполняющиеся потоки с помощью `SHOW PROCESSLIST` и прервать любой из них с помощью оператора `KILL идентификатор_потока`.

Начиная с MySQL 5.0.0, `KILL` предусматривает необязательные модификаторы `CONNECTION` и `QUERY`:

- **KILL CONNECTION** — это то же самое, что `KILL` без модификаторов. Это прерывает соединение, ассоциированное с указанным потоком `идентификатор_потока`.
- **KILL QUERY** прерывает оператор, который в данный момент выполняется в соединении, но оставляет соединение не тронутым.

Если у вас есть привилегия `PROCESS`, вы можете видеть все потоки сервера. Имея привилегию `SUPER`, вы можете прерывать любой поток и любой оператор. В противном случае вы сможете видеть и прерывать только свои собственные потоки и операторы.

Для просмотра и прерывания потоков также используются `mysqladmin processlist` и `mysqladmin kill`.

### На заметку!

В настоящее время вы не можете применять `KILL` с библиотекой встроенного сервера MySQL, так как встроенный сервер запускается внутри потоков приложения хоста; он не создает своих собственных потоков.

Когда вы выполняете `KILL`, устанавливается специальный флаг прерывания потока. В большинстве случаев может потребоваться некоторое время для реальной остановки выполнения потока, поскольку флаги прерывания проверяются сервером только через определенные интервалы времени.

- В циклах `SELECT`, `ORDER BY` и `GROUP BY`, флаг проверяется после чтения блока строк. Если флаг прерывания установлен, выполнение оператора прекращается.
- Во время выполнения `ALTER TABLE` флаг прерывания проверяется перед чтением блока строк из исходной таблицы. Если флаг установлен, выполнение оператора прекращается и временная таблица удаляется.
- Во время выполнения `UPDATE` или `DELETE` флаг прерывания проверяется после чтения каждого блока и после каждой обновленной или удаленной строки. Если флаг установлен, выполнение оператора прекращается. Следует отметить, что если не применялись транзакции, изменения не откатываются!
- `GET_LOCK()` прерывается и возвращает `NULL`.
- Поток, обслуживающий `INSERT DELAYED`, немедленно сбрасывает все строки, которые находятся в памяти, и прерывается.
- Если поток удерживает блокировку таблицы, блокировка немедленно отменяется.
- Если поток ожидает освобождения дискового пространства, чтобы выполнить запись, процедура записи прерывается с сообщением об ошибке “переполнение диска”.

#### 6.5.4.4. Синтаксис `LOAD INDEX INTO CACHE`

```
LOAD INDEX INTO CACHE
    список_индексов_таблицы [, список_индексов_таблицы] ...
список_индексов_таблицы:
    имя_таблицы
    [[INDEX] (имя_индекса [, имя_индекса] ...)]
    [IGNORE LEAVES]
```

Оператор `LOAD INDEX INTO CACHE` предварительно загружает индекс таблицы в кэш ключей, с которым он связан явным вызовом оператора `CACHE INDEX`, либо в кэш ключей по умолчанию в противном случае. `LOAD INDEX INTO CACHE` применяется только в отношении таблиц `MyISAM`.

Модификатор `IGNORE LEAVES` загрузку в кэш только блоков нелистовых узлов индексного дерева.

Следующий оператор предварительно загружает узлы (индексные блоки) индексов таблиц `t1` и `t2`:

```
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

Этот оператор предварительно загружает все индексные блоки `t1` и нелистовые узлы `t2`.

Синтаксис `LOAD INDEX INTO CACHE` позволяет указать, что предварительно загружаться в кэш нужно только отдельные индексы таблицы. Однако текущая реализация загружает сразу все индексы таблицы в кэш, поэтому нет необходимости специфицировать что-то кроме имени таблицы.

`LOAD INDEX INTO CACHE` был добавлен в MySQL 4.1.1.

#### 6.5.4.5. Синтаксис RESET

```
RESET опция_сброса [, опция_сброса] ...
```

Оператор `RESET` используется для очистки состояния различных операций сервера. Он также выступает как более строгая версия оператора `FLUSH` (см. раздел 6.5.4.2).

Чтобы выполнить `RESET`, необходимо обладать привилегией `RELOAD`.

опция\_сброса может быть одной из следующих:

- **MASTER.** Удаляет все бинарные журналы, перечисленные в индексном файле, очищает индексный файл бинарного журнала и создает новый файл бинарного журнала.
- **QUERY CACHE.** Удаляет все результаты из кэша запросов.
- **SLAVE.** Заставляет подчиненный сервер “забыть” свою позицию репликации в бинарном журнале главного сервера. Ранее оператор назывался `FLUSH SLAVE`. См. раздел 6.6.2.

## 6.6. Операторы репликации

### 6.6.1. Операторы SQL для управления главными серверами

Репликацией можно управлять через SQL-интерфейс. В настоящем разделе описываются операторы для управления главными серверами репликации. Раздел 6.6.2 посвящен управлению подчиненными серверами.

#### 6.6.1.1. Синтаксис PURGE MASTER LOGS

```
PURGE {MASTER | BINARY} LOGS TO 'имя_журнала'
PURGE {MASTER | BINARY} LOGS BEFORE 'дата'
```

Удаляет все бинарные журналы, перечисленные в индексе журнала, которые строго предшествуют заданному имени или дате. При этом они также удаляются из списка в индексном файле журнала и, таким образом, заданный файл журнала становится первым.

Примеры:

```
PURGE MASTER LOGS TO 'mysql-bin.010';  
PURGE MASTER LOGS BEFORE '2003-04-02 22:46:26';
```

Вариант BEFORE доступен, начиная с версии MySQL 4.1. Аргумент даты может быть представлен в формате 'ГГГГ-ММ-ДД чч:мм:сс'. MASTER и BINARY являются синонимами, но BINARY появился только в версии MySQL 4.1.1.

Если есть активный подчиненный сервер, в данный момент читающий журнал, который вы пытаетесь удалить, этот оператор не делает ничего и завершается ошибкой. Однако если подчиненный сервер пребывает в бездействии, а вы пытаетесь удалить один из журналов, которые он хочет читать, подчиненный сервер не сможет продолжать репликацию, когда вернется к работе. Этот оператор безопасен, если подчиненные серверы заняты репликацией. Нет необходимости останавливать их.

Для очистки журналов выполните перечисленные ниже шаги.

1. На каждом подчиненном сервере с помощью SHOW SLAVE STATUS определите, какой журнал в настоящий момент читается.
2. Получите список журналов главного сервера с помощью оператора SHOW MASTER LOGS.
3. Определите самый ранний журнал среди подчиненных серверов. Это и будет целевой журнал. Если все журналы на подчиненных серверах являются актуальными, целевым будет последний журнал в списке.
4. Сделайте резервную копию всех журналов, которые собираетесь удалять. (Это не обязательно, но желательно.)
5. Очистите все журналы за исключением целевого.

### 6.6.1.2. Синтаксис RESET MASTER

RESET MASTER

Удаляет все бинарные журналы, перечисленные в индексном файле, очищает сам индексный файл и создает новый файл бинарного журнала.

До версии MySQL 3.23.26 этот оператор назывался FLUSH MASTER.

### 6.6.1.3. Синтаксис SET SQL\_LOG\_BIN

SET SQL\_LOG\_BIN = {0 | 1}

Включает и выключает бинарную регистрацию текущего соединения (SQL\_LOG\_BIN — это переменная сеанса), если клиент подключен, используя учетную запись с привилегией SUPER. Оператор будет отклонен с ошибкой, если клиент не имеет этой привилегии. (До MySQL 4.1.2 в этом случае оператор попросту игнорировался.)

### 6.6.1.4. Синтаксис SHOW BINLOG EVENTS

SHOW BINLOG EVENTS

[IN 'имя\_журнала'] [FROM позиция] [LIMIT [смещение,] количество\_строк]

Показывает события, занесенные в бинарный журнал. Если не указывать 'имя\_журнала', отображается первый бинарный журнал.

Конструкция LIMIT имеет тот же синтаксис, что и в операторе SELECT (см. раздел 6.1.7). Этот оператор доступен, начиная с версии MySQL 4.0.

### 6.6.1.5. Синтаксис SHOW MASTER LOGS

SHOW MASTER LOGS

Выводит список файлов бинарных журналов на главном сервере. Этот оператор применяется как часть процедуры, описанной в разделе 6.6.1.1, чтобы определить, какие журналы могут быть очищены.

### 6.6.1.6. Синтаксис SHOW MASTER STATUS

SHOW MASTER STATUS

Представляет информацию о состоянии файлов бинарных журналов главного сервера.

### 6.6.1.7. Синтаксис SHOW SLAVE HOSTS

SHOW SLAVE HOSTS

Выводит список подчиненных серверов, зарегистрированных в данный момент на главном. Любой подчиненный сервер, который не был запущен с опцией `--report-host=имя_подчиненного_сервера`, в списке показан не будет.

## 6.6.2. SQL-операторы для управления подчиненными серверами

Репликацией можно управлять через SQL-интерфейс. В настоящем разделе описываются операторы для управления подчиненными серверами репликации. Раздел 6.6.1 посвящен управлению главными серверами.

### 6.6.2.1. Синтаксис CHANGE MASTER TO

CHANGE MASTER TO *определение\_главного\_сервера*  
[, *определение\_главного\_сервера*] ...

*определение\_главного\_сервера*:

```
MASTER_HOST = 'имя_хоста'  
| MASTER_USER = 'имя_пользователя'  
| MASTER_PASSWORD = 'пароль'  
| MASTER_PORT = номер_порта  
| MASTER_CONNECT_RETRY = количество  
| MASTER_LOG_FILE = 'имя_журнала_главного_сервера'  
| MASTER_LOG_POS = позиция_в_журнале_главного_сервера  
| RELAY_LOG_FILE = 'имя_журнала_ретрансляций'  
| RELAY_LOG_POS = позиция_в_журнале_ретрансляций  
| MASTER_SSL = {0|1}  
| MASTER_SSL_CA = 'имя_файла_са'  
| MASTER_SSL_CAPATH = 'имя_каталога_са'  
| MASTER_SSL_CERT = 'имя_файла_сертификата'  
| MASTER_SSL_KEY = 'имя_файла_ключей'  
| MASTER_SSL_CIPHER = 'список_шифров'
```

Изменяет параметры, которые подчиненный сервер использует для подключения и взаимодействия с ведущим сервером.

MASTER\_USER, MASTER\_PASSWORD, MASTER\_SSL, MASTER\_SSL\_CA, MASTER\_SSL\_CAPATH, MASTER\_SSL\_CERT, MASTER\_SSL\_KEY и MASTER\_SSL\_CIPHER предоставляют информацию подчиненному серверу о том, как подключиться к главному.

Опции управления ретрансляцией (RELAY\_LOG\_FILE и RELAY\_LOG\_POS) доступны, начиная с MySQL 4.0.

Опции SSL (MASTER\_SSL, MASTER\_SSL\_CA, MASTER\_SSL\_CAPATH, MASTER\_SSL\_CERT, MASTER\_SSL\_KEY и MASTER\_SSL\_CIPHER) доступны, начиная с версии MySQL 4.1.1. Вы можете изменять эти опции даже на подчиненных серверах, которые были скомпилированы без поддержки SSL. Они сохраняются в файле master.info, но игнорируются до тех пор, пока вы не используете сервер с поддержкой SSL.

Если вы не указываете тот или иной параметр, сохраняется его старое значение, за исключением описанных ниже случаев. Например, если изменился пароль для подключения к главному серверу MySQL, вы должны выполнить следующие операторы, чтобы сообщить подчиненному серверу о новом пароле:

```
mysql> STOP SLAVE; -- если репликация была запущена
mysql> CHANGE MASTER TO MASTER_PASSWORD='new3cret';
mysql> START SLAVE; -- если нужно перезапустить репликацию
```

Параметры, которые не изменились, указывать не нужно (хост, порт, пользователь и так далее).

MASTER\_HOST и MASTER\_PORT — это имя хоста (или IP-адрес), на котором находится главный сервер, и номер его порта TCP/IP. Следует отметить, что если MASTER\_HOST является localhost, то, как и в других частях MySQL, порт может быть проигнорирован (если могут быть использованы сокет-файлы Unix, например).

Если вы указываете MASTER\_HOST или MASTER\_PORT, то подчиненный сервер предполагает, что главным сервером стал другой сервер, нежели раньше (даже если вы указываете тот же самый хост и порт). В этом случае старые значения имени бинарного журнала главного сервера и позиции в нем рассматриваются как недействительные. Поэтому, если вы не указываете в операторе MASTER\_LOG\_FILE и MASTER\_LOG\_POS, они принимаются такими: MASTER\_LOG\_FILE='' и MASTER\_LOG\_POS=4.

MASTER\_LOG\_FILE и MASTER\_LOG\_POS — это координаты, по которым поток ввода-вывода подчиненного сервера начнет чтение бинарного журнала главного сервера при следующем запуске. Если вы указываете оба эти параметра, то не можете указать RELAY\_LOG\_FILE или RELAY\_LOG\_POS. Если же не указан ни MASTER\_LOG\_FILE, ни MASTER\_LOG\_POS, то подчиненный сервер использует последние координаты *потока SQL подчиненного сервера*, которые были перед тем, как выполнялся CHANGE MASTER. Это гарантирует, что репликация не будет иметь никаких разрывов, даже если поток SQL подчиненного сервера позже сравнивался с потоком ввода-вывода. Такое безопасное поведение было представлено в MySQL 4.0.17 и MySQL 4.1.1. (До этих версий использовались последние координаты потока ввода-вывода, которые он имела перед тем, как выполнялся CHANGE MASTER. Это приводило к тому, что поток SQL мог потерять некоторые события с главного сервера, таким образом, прерывая репликацию.)

CHANGE MASTER удаляет все файлы журналов ретрансляций и начинает новый журнал, если только не были указаны RELAY\_LOG\_FILE или RELAY\_LOG\_POS. В этом случае журналы ретрансляций сохраняются. С MySQL 4.1.1 значение глобальной переменной relay\_log\_purge по умолчанию равно 0.

CHANGE MASTER обновляет содержимое файлов master.info и relay-log.info.

CHANGE MASTER применим для настройки подчиненного сервера, когда у вас есть снимок данных главного сервера и записано имя файла бинарного журнала и смещения в

нем, которые были действительны на момент получения снимка. После загрузки снимка данных на подчиненный сервер вы можете запустить на нем

```
CHANGE MASTER TO MASTER_LOG_FILE='имя_журнала_главного_сервера',  
MASTER_LOG_POS=позиция_в_журнале_главного_сервера.
```

Примеры:

```
mysql> CHANGE MASTER TO  
-> MASTER_HOST='master2.mycompany.com',  
-> MASTER_USER='replication',  
-> MASTER_PASSWORD='big3cret',  
-> MASTER_PORT=3306,  
-> MASTER_LOG_FILE='master2-bin.001',  
-> MASTER_LOG_POS=4,  
-> MASTER_CONNECT_RETRY=10;  
  
mysql> CHANGE MASTER TO  
-> RELAY_LOG_FILE='slave-relay-bin.006',  
-> RELAY_LOG_POS=4025;
```

Первый пример меняет главный сервер и координаты его бинарного журнала. Это применяется, когда вы хотите настроить подчиненный сервер для репликации данных с главного сервера.

Второй пример демонстрирует операцию, которая выполняется не так часто. Это делается, когда подчиненный сервер уже имеет журналы ретрансляций, которые вы хотите по каким-то причинам выполнить снова. Чтобы это сделать, нужно временно сделать главный сервер недоступным. Вам нужно просто применить `CHANGE MASTER TO` и запустить поток SQL (`START SLAVE SQL_THREAD`).

Вы даже можете использовать вторую операцию в среде без репликации с выделенным сервером, не являющимся подчиненным, для восстановления после аварии. Предположим, что ваш сервер потерпел крах, и вы восстанавливаете резервную копию. Вы хотите повторить операторы, содержащиеся в его собственном бинарном журнале (не в журнале ретрансляций, а в обычном бинарном журнале), имеющим имя, допустим, `myhost-bin.*`. Первое, что потребуется сделать, — создать резервную копию бинарных журналов в каком-то безопасном месте на случай, если вы неточно выполните следующую процедуру и сервер непреднамеренно повредит их. Если используется MySQL 4.1.1 или более новой версии, введите `SET GLOBAL relay_log_purge=0` для большей безопасности. Затем запустите сервер без опции `--log-bin` с новым (отличным от старого) идентификатором сервера (server ID), а также с опциями `--relay-log=myhost-bin` (чтобы представить серверу этот бинарный журнал как журнал ретрансляций) и `--skip-slave-start`. После того, как сервер стартует, выполните следующие операторы:

```
mysql> CHANGE MASTER TO  
-> RELAY_LOG_FILE='myhost-bin.153',  
-> RELAY_LOG_POS=410,  
-> MASTER_HOST='фигтивная_строка';  
mysql> START SLAVE SQL_THREAD;
```

Сервер прочтает и выполнит свой собственный бинарный журнал, таким образом реализовав восстановление после аварии. Как только восстановление завершится, выполните `STOP SLAVE`, остановите сервер, удалите файлы `master.info` и `relay-log.info` и перезапустите сервер с обычными опциями.

На данный момент указание `MASTER_HOST` (даже с фиктивным значением) необходимо, чтобы заставить сервер считать себя подчиненным. Выдать серверу новый, отличающийся от старого, идентификатор сервера также необходимо, иначе он увидит в журнале события, помеченные его собственным идентификатором, и, решив, что это попытка циклической репликации, пропустит их. В будущем мы планируем добавить опции, чтобы избавиться от этих небольших неудобств.

### 6.6.2.2. Синтаксис `LOAD DATA FROM MASTER`

#### `LOAD DATA FROM MASTER`

Берет снимок данных главного сервера и загружает его на подчиненный сервер. При этом обновляются значения `MASTER_LOG_FILE` и `MASTER_LOG_POS` таким образом, чтобы репликация началась с правильной позиции. Учитываются все правила, исключаящие базы данных, или таблицы, указанные опциями `--replicate--do--*` и `--replicate--ignore--*`. Опция `--replicate-rewrite-db` не принимается во внимание (поскольку кто-нибудь может установить неоднозначный режим отображения, такой как `--replicate-rewrite-db=db1->db3` и `--replicate-rewrite-db=db2->db3`, что запутает подчиненный сервер при загрузке таблиц главного сервера).

Применение этого оператора подчиняется следующим условиям:

- Он работает только с таблицами `MyISAM`.
- Требуется глобальной блокировки чтения на главном сервере в процессе получения снимка данных, что предотвращает обновления на главном сервере в процессе операции загрузки.

В будущем мы планируем обеспечить работу этого оператора с таблицами `InnoDB` и исключить необходимость глобальной блокировки чтения за счет применения неблокирующего онлайн-резервного копирования.

При загрузке больших таблиц может понадобиться увеличить значения `net_read_timeout` и `net_write_timeout` как на главном, так и на подчиненном серверах.

Отметим, что `LOAD DATA FROM MASTER` не выполняет копирования таблиц из базы данных `mysql`. Это позволяет иметь разных пользователей с различными привилегиями на главном и подчиненном серверах.

Оператор `LOAD DATA FROM MASTER` требует наличия специальной пользовательской учетной записи репликации, который используется для подключения к главному и имеет привилегии `RELOAD` и `SUPER`, а также привилегию `SELECT` на все таблицы главного сервера, которые нужно загружать. Все таблицы главного сервера, к которым у пользователя нет прав доступа по `SELECT`, игнорируются оператором `LOAD DATA FROM MASTER`. Это связано с тем, что главный сервер скрывает их от пользователя: `LOAD DATA FROM MASTER` вызывает `SHOW DATABASES`, чтобы получить список баз данных для загрузки, а `SHOW DATABASES` возвращает только те базы данных, в которых у пользователя есть какие-то привилегии. См. раздел 6.5.3.7. Со стороны подчиненного сервера пользователь, который выполняет `LOAD DATA FROM MASTER`, должен иметь права на удаление и создание баз данных и таблиц, которые подлежат копированию.

### 6.6.2.3. Синтаксис `LOAD TABLE имя_таблицы FROM MASTER`

#### `LOAD TABLE имя_таблицы FROM MASTER`



Переносит копию таблицы с главного на подчиненный сервер. Этот оператор реализован главным образом для отладки `LOAD DATA FROM MASTER`. Он требует, чтобы пользовательская учетная запись, которая подключается к главному серверу, имела там привилегии `RELOAD` и `SUPER`, а также `SELECT` на таблицу, которую нужно загружать.

Со стороны подчиненного сервера пользователь, запускающий `LOAD TABLE...FROM MASTER` должен иметь привилегии на уничтожение и создание таблицы.

Условия применения `LOAD DATA FROM MASTER` применимы и здесь. Например, `LOAD TABLE FROM MASTER` работает только с таблицами `MyISAM`. Замечания по установке таймаутов также актуальны.

#### 6.6.2.4. Синтаксис `MASTER_POS_WAIT()`

```
SELECT MASTER_POS_WAIT('имя_журнала_главного_сервера',  
                        позиция_в_журнале_главного_сервера)
```

Это функция, а не оператор. Применяется для обеспечения того, чтобы подчиненный сервер прочитал и исполнил все события из бинарного журнала главного сервера до указанной позиции. См. полное описание в разделе 5.8.4.

#### 6.6.2.5. Синтаксис `RESET SLAVE`

```
RESET SLAVE
```

Заставляет подчиненный сервер забыть позицию репликации в бинарном журнале главного сервера. Этот оператор предназначен для “чистого” старта. Он удаляет файлы `master.info` и `relay-log.info`, все старые журналы ретрансляций и стартует новый журнал ретрансляций.

##### На заметку!

Все журналы ретрансляций удаляются, даже если они еще не полностью обработаны потоком SQL подчиненного сервера. (Это весьма вероятно, особенно если была дана команда `STOP SLAVE` или подчиненный сервер сильно нагружен.)

Информация о подключении, сохраняемая в файле `master.info`, немедленно очищается с использованием любых значений, указанных через соответствующие опции запуска. Эта информация включает в себя: имя хоста главного сервера, номер порта, имя пользователя и пароль. Если поток SQL подчиненного сервера находился в процессе репликации временных таблиц, когда он был остановлен и выполнен оператор `RESET SLAVE`, реплицированные временные таблицы удаляются с подчиненного сервера.

До версии MySQL 3.23.26 этот оператор назывался `FLUSH SLAVE`.

#### 6.6.2.6. Синтаксис `SET GLOBAL SQL_SLAVE_SKIP_COUNTER`

```
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = n
```

Пропускает следующие *n* событий с главного сервера. Это применимо для восстановления после остановки репликации, вызванной некоторым оператором.

Этот оператор допустим, только когда поток подчиненного не работает. В противном случае генерируется ошибка.

До MySQL 4.0 в этом операторе пропускайте слово `GLOBAL`.

#### 6.6.2.7. Синтаксис `SHOW SLAVE STATUS`

```
SHOW SLAVE STATUS
```

Представляет информацию о состоянии существенных параметров потоков подчиненного сервера. Если выполнить этот оператор из среды клиентской программы `mysql`, можно указать ограничитель `\G` вместо точки с запятой и получить более читабельный вертикальный вывод:

```
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
```

В зависимости от установленной версии MySQL вы можете не увидеть некоторых полей из числа приведенных выше. В частности, некоторые поля присутствуют только в MySQL 4.1.1.

`SHOW SLAVE STATUS` возвращает следующие поля:

- **Slave\_IO\_State.** Копия поля `State` из вывода `SHOW PROCESSLIST` для потока ввода-вывода подчиненного сервера. Информировать о том, что поток пытается подключиться к главному серверу, ожидает событий с главного сервера, повторно подключается и так далее. Обращать внимание на это поле необходимо, потому что, например, поток может работать, но ему не удастся подключиться к главному серверу. Только здесь вы можете обнаружить проблему подключения. Состояние потока SQL не копируется, потому что оно проще. Если он работает, то проблем нет, а если нет, то вы найдете ошибку в поле `Last_Error`, которое описано ниже. Это поле появилось в версии MySQL 4.1.1.

- **Master\_Host.** Хост главного сервера.
- **Master\_User.** Имя пользователя, подключенного к главному серверу.
- **Master\_Port.** Номер порта на главном сервере.
- **Connect\_Retry.** Текущее значение опции `--master-connect-retry`.
- **Master\_Log\_File.** Имя файла бинарного журнала на главном сервере, из которого поток ввода-вывода осуществляет чтение.
- **Read\_Master\_Log\_Pos.** Позиция в бинарном журнале главного сервера, до которой дочитал поток ввода-вывода.
- **Relay\_Log\_File.** Имя файла журнала ретрансляций, из которого поток SQL осуществляет чтение и выполнение операторов.
- **Relay\_Log\_Pos.** Позиция в журнале ретрансляций, до которой поток SQL прочитал и выполнил операторы.
- **Relay\_Master\_Log\_File.** Имя файла бинарного журнала главного сервера, который содержит последний оператор, выполненный потоком SQL.
- **Slave\_IO\_Running.** Запущен ли поток ввода-вывода.
- **Slave\_SQL\_Running.** Запущен ли поток SQL.
- **Replicate\_Do\_DB,**  
**Replicate\_Ignore\_DB**

Список баз данных, которые указаны опциями `--replicate-do-db` и `--replicate-ignore-db`, если таковые были.

- **Replicate\_Do\_Table,**  
**Replicate\_Ignore\_Table,**  
**Replicate\_Wild\_Do\_Table,**  
**Replicate\_Wild\_Ignore\_Table**

Список таблиц, которые указаны опциями `--replicate-do-table`, `--replicate-ignore-table`, `--replicate-wild-do-table` и `--replicate-wild-ignore-table`, если таковые были.

Эти поля присутствуют, начиная с MySQL 4.1.1.

- **Last\_Errno, Last\_Error.** Номер ошибки и сообщение об ошибке, возвращенное последним запросом. Номер ошибки 0 и сообщение в виде пустой строки означают, что ошибок нет. Если **Last\_Error** не пустое, оно также появляется в журнале ошибок подчиненного сервера.

Например:

**Last\_Errno:** 1051

**Last\_Error:** error 'Unknown table 'z'' on query 'drop table z'

Это означает, что таблица **z** находилась на главном сервере и была там удалена, но она не существует на подчиненном сервере, поэтому оператор **DROP TABLE** не прошел. (Это может случиться, например, если при настройке репликации вы забыли скопировать таблицу на подчиненный сервер.)

- **Skip\_Counter.** Последнее использованное значение **SQL\_SLAVE\_SKIP\_COUNTER**.

- `Exec_Master_Log_Pos`. Позиция последнего исполненного потоком SQL оператора в бинарном журнале главного сервера (`Relay_Master_Log_File`). (`Relay_Master_Log_File`, `Exec_Master_Log_Pos`) в бинарном журнале главного сервера соответствуют (`Relay_Log_File`, `Relay_Log_Pos`) в журнале ретрансляций.
- `Relay_Log_Space`. Суммарный размер всех существующих журналов ретрансляций.
- `Until_Condition`,  
`Until_Log_File`,  
`Until_Log_Pos`

Значения, указанные в конструкции `UNTIL` оператора `START SLAVE`.

`Until_Condition` включает 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 ретрансляций.

`Until_Log_File` и `Until_Log_Pos` означают имя файла журнала и позицию в нем, определяющие точку, в которой поток SQL прекратит выполнение.

Эти поля присутствуют, начиная с MySQL 4.1.1.

- `Master_SSL_Allowed`,  
`Master_SSL_CA_File`,  
`Master_SSL_CA_Path`,  
`Master_SSL_Cert`,  
`Master_SSL_Cipher`,  
`Master_SSL_Key`

Эти поля показывают параметры SSL, используемые подчиненным сервером при подключении к главному серверу, если они есть.

`Master_SSL_Allowed` принимает следующие значения:

- `Yes`, если SSL-подключение к главному серверу разрешено.
- `No`, если SSL-подключение к главному серверу не разрешено.
- `Ignored`, SSL-подключение к главному серверу разрешено, но подчиненный сервер не поддерживает SSL.

Значения остальных полей, имеющих отношение к SSL, соответствуют значениям опций `--master-ca`, `--master-capath`, `--master-cert`, `--master-cipher` и `--master-key`.

Эти поля представлены, начиная с MySQL 4.1.1.

- `Seconds_Behind_Master`. Количество секунд, прошедших от временной метки последнего события с главного сервера, выполненного потоком SQL. Оно будет равно `NULL`, если еще не выполнялось ни одно из них, либо сразу после выдачи `CHANGE MASTER` или `RESET SLAVE`. Это поле может быть использовано для определения того, насколько “запаздывает” подчиненный сервер. Это работает, даже если часы на главном и подчиненном сервере не синхронизированы.

Эти поля представлены, начиная с MySQL 4.1.1.

### 6.6.2.8. Синтаксис START SLAVE

```
START SLAVE [тип_потока [, тип_потока] ... ]
START SLAVE [SQL_THREAD] UNTIL
    MASTER_LOG_FILE = 'имя_журнала', MASTER_LOG_POS = позиция_в_журнале
START SLAVE [SQL_THREAD] UNTIL
    RELAY_LOG_FILE = 'имя_журнала', RELAY_LOG_POS = позиция_в_журнале
тип_потока: IO_THREAD | SQL_THREAD
```

START SLAVE без опций запускает оба потока репликации на подчиненном сервере. Поток ввода-вывода читает запросы с главного сервера и помещает их в журнал ретрансляций. Поток SQL читает журнал ретрансляций и выполняет запросы. START SLAVE требует наличия привилегии SUPER.

Если START SLAVE удалось запустить потоки репликации на подчиненном сервере, он возвращает управление без ошибок. Однако даже в этом случае может случиться, что потоки подчиненного сервера стартуют, а позже остановятся (например, потому, что он не в состоянии подключиться к главному серверу или прочесть его бинарный журнал, либо из-за каких-то других проблем). START SLAVE не предупреждает об этом. Вы должны проверять журнал ошибок подчиненного сервера на предмет наличия сообщений об ошибках, сгенерированных потоками репликации подчиненного сервера, либо проверять, как он работает, с помощью SHOW SLAVE STATUS.

Начиная с MySQL 4.0.2, можно добавлять опции IO\_THREAD и SQL\_THREAD к оператору, чтобы указать, какой из потоков надо запустить.

Начиная с MySQL 4.1.1, можно указывать конструкцию UNTIL для указания того, что подчиненный сервер должен запуститься и работать до тех пор, пока SQL не достигнет заданной точки в бинарном журнале главного сервера или в журнале ретрансляций подчиненного сервера. Когда поток SQL достигает этой точки, он останавливается. Если в операторе указана опция SQL\_THREAD, он запускает только поток SQL. В противном случае запускаются оба потока подчиненного сервера. Если поток SQL уже работает, конструкция UNTIL игнорируется и выдается предупреждение.

С конструкцией UNTIL обязательно указывать и имя файла журнала, и позицию в нем. Не смешивайте опции журнала главного сервера и журнала ретрансляций подчиненного сервера.

Конструкция UNTIL сбрасывается последующим оператором STOP SLAVE, либо START SLAVE без конструкции UNTIL, либо перезапуском сервера.

Конструкция UNTIL может оказаться удобной для отладки репликации либо для выполнения репликации только до определенной точки, в которой нужно избежать репликации отдельного оператора. Например, если нежелательный оператор DROP TABLE был выполнен на главном сервере, вы можете применить UNTIL, чтобы сообщить подчиненному серверу, что репликацию нужно выполнять до этого оператора исключительно, но не далее. Для поиска нужного события в журнале пользуйтесь командой mysqlbinlog с бинарным журналом главного сервера или журналом ретрансляций подчиненного сервера либо оператором SHOW BINLOG EVENTS.

Если вы используете UNTIL для того, чтобы организовать репликацию по разделам, мы рекомендуем запускать подчиненный сервер с опцией --skip-slave-start, чтобы предотвратить автоматический запуск потока SQL при старте подчиненного сервера. Возможно, лучше указывать эту опцию в файле опций, чем в командной строке, чтобы неожиданный перезапуск сервера не привел к тому, что она будет забыта.

Оператор `SHOW SLAVE STATUS` включает поля вывода, которые отображают текущие установки `UNTIL`.

До версии MySQL 4.0.5 этот оператор назывался `SLAVE START`. В настоящее время такое написание принимается для обратной совместимости, но считается устаревшим.

#### 6.6.2.9. Синтаксис `STOP SLAVE`

```
STOP SLAVE [тип_потока [, тип_потока] ... ]
```

*тип\_потока:* `IO_THREAD` | `SQL_THREAD`

Останавливает потоки репликации подчиненного сервера. `STOP SLAVE` требует привилегии `SUPER`.

Как и `START SLAVE`, начиная с MySQL 4.0.2, этот оператор может использоваться с опциями `IO_THREAD` и `SQL_THREAD` для именованного потока или потоков, которые нужно остановить.

До MySQL 4.0.5 оператор назывался `SLAVE STOP`. В настоящее время `SLAVE STOP` принимается для обратной совместимости, но считается устаревшим.

# Пространственные расширения в MySQL

**В** MySQL 4.1 введены пространственные расширения, позволяющие генерировать, хранить и анализировать географические свойства. В настоящее время эти свойства доступны только для таблиц MyISAM.

В этой главе рассматриваются следующие вопросы:

- Основы пространственных расширений в геометрической модели OpenGIS.
- Форматы для представления пространственных данных.
- Использование пространственных данных в MySQL.
- Использование индексации для пространственных данных.
- Отличия MySQL от спецификации OpenGIS.

## 7.1. Введение

MySQL реализует пространственные расширения согласно спецификации Open GIS Consortium, Inc (OGC). OGC – это международный консорциум, в который входят более 250 компаний, агентств и университетов, занимающихся разработкой общедоступных концептуальных решений для всех типов приложений, которые работают с пространственными данными. У OGC есть свой Web-сайт, доступный по адресу <http://www.opengis.org/>.

В 1997 году консорциум Open GIS Consortium опубликовал документ под названием “Спецификации простых функций OpenGIS для SQL”, в котором были предложены несколько концептуальных методов расширения SQL RDBMS для поддержки пространственных данных. Его можно найти на Web-сайте консорциума Open GIS по адресу <http://www.opengis.org/docs/99-049.pdf>. Там же находится дополнительная информация по теме данной главы.

MySQL поддерживает подмножество среды **SQL with Geometry Types** (SQL и геометрические типы), предложенное OGC. Этот термин относится к среде SQL, которая была расширена набором геометрических типов. SQL-столбец с геометрическими значениями реализован как столбец, имеющий геометрический тип. В спецификации опи-

сывается набор геометрических типов SQL, а также доступные в этих типах функции, необходимые для создания и анализа геометрических значений.

Географический элемент – это любой в мире элемент, имеющий местоположение. Географическим элементом могут быть:

- Объект. Например, гора, водоем, город.
- Пространство. Например, область, соответствующая почтовому индексу, тропики.
- Определенное местонахождение. Например, перекресток как конкретное место пересечения двух улиц.

Также встречаются документы, в которых для ссылки на географические элементы используется термин “геопространственный элемент” (geospatial feature).

Геометрия (geometry) – еще одно слово, которым обозначают географический элемент. Изначально слово “геометрия” означало систему мер Земли. Другое значение происходит из картографии и относится к геометрическим элементам, применяемым картографами для составления карт мира.

В этой главе все эти термины – географический элемент, геопространственный элемент и геометрия – используются как синонимы. Чаще всего из этих терминов здесь встречается термин “геометрия”.

Давайте определим “геометрию” как *точку или совокупность точек, которые представляют любой объект в мире, имеющий определенное месторасположение*.

## 7.2. Геометрическая модель OpenGIS

Набор геометрических типов, предложенный OGC для среды SQL with Geometry Types основывается на геометрической модели OpenGIS (OpenGIS Geometry Model). В этой модели каждый геометрический объект имеет следующие общие свойства:

- Объект ассоциируется с пространственной системой координат, описывающей координаты пространства, в котором находится объект.
- Объект принадлежит к некоторому геометрическому классу.

### 7.2.1. Иерархия геометрических классов

Иерархия геометрических классов выглядит следующим образом:

- Geometry (без возможности создания экземпляров)
  - Point (с возможностью создания экземпляров)
  - Curve (без возможности создания экземпляров)
- LineString (с возможностью создания экземпляров)
  - Line
  - LinearRing
- Surface (без возможности создания экземпляров)
  - Polygon (с возможностью создания экземпляров)
- GeometryCollection (с возможностью создания экземпляров)
  - MultiPoint (с возможностью создания экземпляров)
  - MultiCurve (без возможности создания экземпляров)
  - MultiLineString (с возможностью создания экземпляров)



- MultiSurface (без возможности создания экземпляров)
  - MultiPolygon (с возможностью создания экземпляров)

Создавать объекты в классах без возможности создания экземпляров нельзя, их можно создавать только в классах с возможностью создания экземпляров. Все классы обладают свойствами, и классы с возможностью создания экземпляров также могут иметь утверждения (правила, определяющие допустимые экземпляры класса).

Geometry является базовым классом. Это абстрактный класс. Подклассы с возможностью создания экземпляров класса Geometry ограничиваются нуль-одно- и двухмерными геометрическими объектами, существующими в двухмерном координатном пространстве. Все геометрические классы с возможностью создания экземпляров определены так, что допустимые экземпляры геометрического класса являются топологически замкнутыми (то есть все определенные геометрии имеют границы).

Базовый класс Geometry включает подклассы для Point, Curve, Surface и GeometryCollection:

- Point представляет нульмерные объекты.
- Curve представляет одномерные объекты и включает подкласс LineString, который, в свою очередь, содержит подклассы Line и LinearRing.
- Surface предназначен для двухмерных объектов и включает подкласс Polygon.
- GeometryCollection содержит специализированные классы нуль-одно- и двухмерных коллекций — MultiPoint, MultiLineString и MultiPolygon, необходимые для моделирования геометрических форм, соответствующих коллекциям Points, LineStrings и Polygons (в указанном порядке). MultiCurve и MultiSurface введены как абстрактные суперклассы, обобщающие интерфейсы коллекций для обработки Curves и Surfaces.

Geometry, Curve, Surface, MultiCurve и MultiSurface обозначены как классы без возможности создания экземпляров. Они определяют общий набор методов для соответствующих им подклассов и служат для расширяемости.

Point, LineString, Polygon, GeometryCollection, MultiPoint, MultiString и MultiPolygon являются классами с возможностью создания экземпляров.

### 7.2.2. Класс Geometry

Класс Geometry является корневым в иерархии. Это класс без возможности создания экземпляров, имеющий, однако, ряд свойств, общих для всех геометрических значений, которые созданы любым из подклассов класса Geometry. Эти свойства описываются в представленном ниже списке. (Некоторые подклассы обладают собственными специфическими свойствами, которые будут рассмотрены позже.)

#### Геометрические свойства

Геометрические значения имеют следующие свойства:

- **Тип.** Каждая геометрическая форма принадлежит одному из классов с возможностью создания экземпляров в иерархии.
- **Идентификатор пространственной системы координат (Spatial Reference Identifier — SRID).** Это значение идентифицирует пространственную систему координат (Spatial Reference System), описывающую координатное пространство, в котором задан геометрический объект.

- **Координаты** в пространственной системе координат, представленные в виде чисел с двойной (8 байт) точностью. Все непустые геометрии включают, по крайней мере, одну пару координат (X,Y). Пустые геометрии координат не содержат. Координаты зависят от идентификатора пространственной системы координат SRID. Например, в различных системах координат расстояние между двумя объектами может отличаться, даже если объекты имеют одинаковые координаты, поскольку расстояние в плоской системе координат и расстояние в геоцентрической системе координат (рассматривающей координаты на поверхности Земли) – это разные понятия.
- **Внутреннее пространство, границы и внешнее пространство.** Каждая геометрическая форма занимает определенное место в пространстве. Внешнее пространство – это все пространство, не занимаемое геометрической формой. Внутреннее пространство – это пространство, занимаемое геометрической формой. Границы – это область стыка внешнего и внутреннего пространства.
- **Минимальный ограничивающий прямоугольник (Minimum Bounding Rectangle – MBR), или огибающая.** Это ограничивающая геометрическая форма, обусловленная минимальными и максимальными координатами (X,Y):  
({MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY})
- **Качество объекта: простой (simple) или непростой (non-simple).** Геометрические значения типов (LineString, MultiPoint, MultiString) являются либо простыми, либо непростыми. Каждый тип определяет свои собственные утверждения по присвоению качества “простой/непростой”.
- **Качество объекта: замкнутый (closed) или незамкнутый (not closed).** Геометрические значения типов (LineString, MultiString) являются либо замкнутыми, либо незамкнутыми. Каждый тип определяет свои собственные утверждения по присвоению качества “замкнутый/незамкнутый”.
- **Качество объекта: пустой (empty) или непустой (not empty).** Геометрический объект считается пустым, если у него нет ни одной точки. Границы, внешнее и внутреннее пространство пустой геометрии не определены, то есть, представлены значениями NULL. Пустая геометрическая форма всегда простая и значение ее области всегда равно нулю.
- **Измерение.** Измерение геометрического объекта может принимать следующие значения: -1, 0, 1, or 2:
  - -1 для пустой геометрии.
  - 0 для геометрии, не имеющей длины и не имеющей области.
  - 1 для геометрии с отличной от нуля длиной и равной нулю областью.
  - 2 для геометрии с ненулевой областью.

Объекты Point имеют нулевое измерение. Объекты LineString имеют измерение, равное 1. Объекты Polygon имеют измерение, равное 2. Измерение объектов MultiPoint, MultiString и MultiPolygon такое же, как и измерение элементов, из которых они состоят.

### 7.2.3. Класс Point

Point (точка) – это геометрия, представляющая собой единичное месторасположение в пространстве координат.

#### Примеры Point

- Представьте себе крупномасштабную карту мира с множеством городов: объект Point может обозначать каждый город.
- На карте города объект Point может обозначать автобусную остановку.

#### Свойства Point

- Значение координаты X.
- Значение координаты Y.
- Объект Point определен как геометрический объект с нулевым измерением.
- Значения границ объекта Point пустые.

### 7.2.4. Класс Curve

Curve (кривая) – это одномерная геометрия, обычно представляемая в виде последовательности точек. Отдельные подклассы класса Curve определяют тип интерполяции между точками. Класс Curve является классом без возможности создания экземпляров.

#### Свойства Curve

- Кривая Curve имеет координаты составляющих ее точек.
- Кривая Curve определена как одномерная геометрия.
- Кривая Curve является простой, если не проходит через одну и ту же точку дважды.
- Кривая Curve является замкнутой, если начальная точка кривой также является и ее конечной точкой.
- Значения границ замкнутой кривой Curve пустые.
- Границы замкнутой кривой Curve состоят из ее двух конечных точек.
- Простая и замкнутая кривая Curve представляет собой линейное кольцо, то есть объект LinearRing.

### 7.2.5. Класс LineString

LineString (ломаная линия) представляет собой кривую линию Curve с линейной интерполяцией между точками.

#### Примеры объекта LineString

- На карте мира объекты LineString могли бы обозначать реки.
- На карте города объекты LineString могли бы обозначать улицы.

#### Свойства объекта LineString

- Объект LineString имеет координаты сегментов, определяемые каждой последовательной парой точек.
- Объект LineString является линией Line, если он состоит только из двух точек.
- Объект LineString представляет собой линейное кольцо LinearRing, если он является одновременно замкнутым и простым.

## 7.2.6. Класс Surface

Surface (поверхность) – это двумерная геометрическая форма. Класс Surface является классом без возможности создания экземпляров с единственным подклассом с возможностью создания экземпляров – Polygon (многоугольник).

### Свойства объекта Surface

- Surface определен как двумерный геометрический объект.
- В спецификации OpenGIS простая поверхность определяется как геометрический объект, который состоит из одного “участка”, имеющего единственную внешнюю границу и ноль или более внутренних границ.
- Границы простого объекта Surface представляют собой множество замкнутых кривых, соответствующих его внешним и внутренним границам.

## 7.2.7. Класс Polygon

Polygon (многоугольник) – это плоская поверхность (Surface), представляющая собой многостороннюю геометрию. Объект Polygon определяется единственной внешней границей и нуль или большим количеством внутренних границ, где каждая внутренняя граница обозначает отверстие в Polygon.

### Примеры объекта Polygon

- На карте области объекты Polygon могли бы обозначать лесные участки, районы и так далее.

### Утверждения для Polygon

- Границы Polygon состоят из множества объектов LinearRing (линейное кольцо) (объектов LineString, которые являются одновременно простыми и замкнутыми), образующих его внешние и внутренние границы.
- Объект Polygon не имеет пересекающихся колец. Кольца в пределах объекта Polygon могут пересекаться в некоторой точке Point, но только по касательной.
- Объект Polygon не содержит линий, острых выступов или отверстий.
- Внутреннее пространство объекта Polygon представляет собой совокупность связанных точек.
- Объект Polygon может содержать отверстия. Внешние границы объекта Polygon с отверстиями не соприкасаются. Каждое отверстие определяет связанный компонент внешнего пространства.

Изложенные выше утверждения делают объект Polygon простым геометрическим объектом.

## 7.2.8. Класс GeometryCollection

GeometryCollection (коллекция геометрических форм) – это геометрия, представляющая собой коллекцию из одного или больше геометрических объектов любого класса.

Все элементы в GeometryCollection должны находиться в одной и той же пространственной системе координат. Других ограничений для элементов класса GeometryCollection нет, хотя в некоторых подклассах GeometryCollection, описывае-

мых в следующих разделах, принадлежность может быть ограничена по следующим признакам:

- Тип элемента (например, объект `MultiPoint` может состоять только из элементов `Point`).
- Измерение.
- Ограничения по степени перекрытия пространств между элементами.

## 7.2.9. Класс `MultiPoint`

`MultiPoint` (множество точек) представляет собой коллекцию геометрических объектов, состоящую из элементов `Point` (точка). Точки не упорядочены и не связаны между собой.

### Примеры объекта `MultiPoint`

- На карте мира объект `MultiPoint` мог бы обозначать цепочку небольших островов.
- На карте города с помощью объекта `MultiPoint` можно было бы представить входы в билетную кассу.

### Свойства объекта `MultiPoint`

- Объект `MultiPoint` – это геометрия с нулевым измерением.
- Объект `MultiPoint` является простым, если никакие два из его значений `Point` не равны (не имеют идентичных значений координат).
- Значения границ объекта `MultiPoint` пустые.

## 7.2.10. Класс `MultiCurve`

`MultiCurve` (множество кривых) представляет собой коллекцию геометрических объектов, состоящую из элементов `Curve` (кривая). Класс `MultiCurve` является классом без возможности создания экземпляров.

### Свойства объекта `MultiCurve`

- Объект `MultiCurve` – это одномерная геометрия.
- Объект `MultiCurve` является простым, если и только если все его элементы простые; два любых элемента могут пересекаться исключительно в точках, расположенных на границах обоих элементов.
- Значения границ объекта `MultiCurve` получаются путем применения “правила объединения по модулю 2”, (также известного как “правило ‘четный/нечетный’”): точка находится в пределах объекта `MultiCurve`, если она находится в пределах нечетного числа элементов `MultiCurve`.
- Объект `MultiCurve` считается замкнутым, если все составляющего его элементы являются замкнутыми.
- Значения границ замкнутого объекта `MultiCurve` всегда пустые.

## 7.2.11. Класс `MultiLineString`

`MultiLineString` (множество ломаных линий) – это коллекция геометрий `MultiCurve`, состоящая из элементов `LineString`.

### Примеры объекта MultiLineString

- На карте области с помощью объекта MultiLineString можно было бы представлять систему рек или систему скоростных шоссе.

## 7.2.12. Класс MultiSurface

MultiSurface (множество поверхностей) представляет собой коллекцию геометрий, состоящую из элементов Surface (поверхность). Класс MultiSurface является классом без возможности создания экземпляров и включает единственный подкласс с возможностью создания экземпляров – MultiPolygon.

### Утверждения для MultiSurface

- Внутренние пространства объектов MultiSurface никогда не пересекаются.
- Два элемента MultiSurface имеют границы, которые пересекаются самое большее в конечном числе точек.

## 7.2.13. Класс MultiPolygon

MultiPolygon (множество многоугольников) является объектом MultiSurface, который состоит из элементов Polygon.

### Примеры объекта MultiPolygon

- На карте области с помощью объекта MultiPolygon можно представить систему озер.

### Утверждения для MultiPolygon

- Объект MultiPolygon не может содержать двух элементов Polygon, внутренние пространства которых частично пересекаются.
- Объект MultiPolygon не может содержать двух элементов Polygon, которые пересекаются (пересечение также запрещается и предыдущим утверждением) или которые соприкасаются в бесконечном количестве точек.
- Объект MultiPolygon не может содержать острых выступов, обрезанных или оборванных линий. Объект MultiPolygon – стандартное замкнутое множество точек.
- Внутреннее пространство объекта MultiPolygon, содержащего более одного объекта Polygon, будет несвязанным. Число связанных компонентов во внутреннем пространстве объекта MultiPolygon равняется количеству значений Polygon в этом объекте MultiPolygon.

### Свойства объекта MultiPolygon

- Объект MultiPolygon является двумерной геометрией.
- Границы объекта MultiPolygon представляют собой множество замкнутых кривых (значения LineString), соответствующих границам его элементов Polygon.
- Каждый объект Curve (кривая) в пределах MultiPolygon находится в пределах только одного элемента Polygon (многоугольник).
- Каждый объект Curve в пределах элемента Polygon находится в пределах объекта MultiPolygon.

## 7.3. Поддерживаемые форматы пространственных данных

В этом разделе описываются стандартные форматы пространственных данных, используемые для представления геометрических объектов в запросах. Форматы пространственных данных бывают следующими:

- WKT – известный текстовый формат (Well-Known Text).
- WKB – известный двоичный формат (Well-Known Binary).

Внутри MySQL значения геометрических объектов сохраняются в формате, не являющемся идентичным ни формату WKT, ни формату WKB.

### 7.3.1. Формат WKT

Формат WKT разработан для обмена геометрическими данными в виде ASCII.

Ниже приведены примеры представления геометрических объектов в формате WKT:

- Объект Point.

```
POINT(15 20)
```

Обратите внимание, что координаты точки запятой не разделяются.

- Объект LineString, состоящий из четырех точек.

```
LINESTRING (0 0, 10 10, 20 25, 50 60)
```

Обратите внимание, что пары координат точек разделяются запятыми.

- Объект Polygon с одним внешним и одним внутренним кольцами:

```
Polygon ((0 0,10 0,10 10,0 10,0 0), (5 5,7 5,7 7,5 7, 5 5))
```

- Объект MultiPoint с тремя значениями Point.

```
MultiPoint (0 0, 20 20, 60 60)
```

- Объект MultiLineString с двумя значениями LineString.

```
MULTILINESTRING ((10 10, 20 20), (15 15, 30 15))
```

- Объект MultiPolygon с двумя значениями Polygon.

```
MultiPolygon (((0 0,10 0,10 10,0 10,0 0)), ((5 5,7 5,7 7,5 7, 5 5)))
```

- Объект GeometryCollection, состоящий из двух значений Point и одного значения LineString.

```
GEOMETRYCOLLECTION (POINT(10 10), POINT(30 30), LINESTRING (15 15, 20 20))
```

Грамматику Бэкуса-Наура, которая определяет формальные порождающие правила для записи WKT-значений можно найти в спецификации OGC, о которой шла речь в начале этой главы.

### 7.3.2. Формат WKB

Формат WKB, предназначенный для представления геометрических значений, определен в спецификации OpenGIS, а также в разделе “SQL/MM Part 3: Spatial” (“SQL/MM. Часть 3: Пространственные данные”) стандартов, установленных Международной организацией по стандартизации ISO.

Формат WKB используется для обмена геометрическими данными в виде двоичных потоков, представленных значениями BLOB, которые содержат геометрическую WKB-информацию.

В WKB используются однобайтовые целые числа без знака, 4-байтовые целые числа без знака и 8-байтовые числа с двойной точностью (формат IEEE 754). Один байт равен 8 битам.

Например, WKB-значение, соответствующее POINT(1 1), состоит из показанной ниже 21-байтовой последовательности (каждый из байтов представлен здесь двумя шестнадцатеричными цифрами):

```
0101000000000000000000F03F000000000000F03F
```

Последовательность можно поделить на такие компоненты:

Порядок байтов: 01

Тип WKB: 01000000

X: 000000000000F03F

Y: 000000000000F03F

Компоненты могут быть представлены следующим образом:

- Значение порядка байтов может быть либо 0, либо 1 для указания сохранения с прямым или обратным порядком байтов. Прямой и обратный порядки байтов также известны как форматы NDR (Network Data Representation – сетевое представление данных) или XDR (External Data Representation – внешнее представление данных) соответственно.
- Тип WKB – это код, обозначающий геометрический тип. Значения от 1 до 7 указывают на объекты Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon и GeometryCollection.
- Значение объекта Point имеет координаты X и Y, каждая из которых представлена в виде числа с двойной точностью.

WKB-значения для более сложных геометрических объектов представляются с помощью более сложных структур данных, что более подробно рассматривается в спецификации OpenGIS.

## 7.4. Создание базы данных MySQL для работы с пространственными данными

В этом разделе описываются типы данных, которые можно использовать для представления пространственных данных в MySQL, и функции, доступные для создания и извлечения пространственных значений.

### 7.4.1. Типы пространственных данных MySQL

В MySQL имеются типы данных, которые соответствуют классам OpenGIS. Некоторые из этих типов хранят единственное геометрическое значение:

- GEOMETRY
- POINT
- LINESTRING
- POLYGON



GEOMETRY может хранить геометрические значения любого типа. Остальные типы пространственных данных с единственным значением — POINT, LINESTRING и POLYGON — ограничивают свои значения определенным геометрическим типом.

Некоторые типы данных могут хранить коллекции значений:

- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

GEOMETRYCOLLECTION позволяет сохранять коллекцию объектов любого типа. Остальные типы коллекций — MULTIPOINT, MULTILINESTRING, MULTIPOLYGON и GEOMETRYCOLLECTION — ограничиваются коллекциями с определенным типом геометрических объектов.

## 7.4.2. Создание пространственных значений

В этом разделе описываются способы создания пространственных значений с помощью WKT- и WKB-функций, определенных в стандартах OpenGIS, а также с помощью специальных MySQL-функций.

### 7.4.2.1. Создание геометрических значений с помощью WKT-функций

MySQL предлагает целый ряд функций, которые в качестве входных параметров принимают WKT-значения и, по выбору, идентификатор пространственной системы координат (SRID). Они возвращают значение соответствующей геометрии.

Функция `GeomFromText()` в качестве первого аргумента принимает WKT-значение любого геометрического типа. Реализация также обеспечивает отдельные для каждого типа функции-конструкторы, необходимые для создания геометрических значений каждого типа геометрических объектов.

- `GeomCollFromText(wkt[,srid]), GeometryCollectionFromText(wkt [, srid])`  
Создает значение GEOMETRYCOLLECTION, используя его WKT-формат и идентификатор SRID.
- `GeomFromText(wkt[,srid]), GeometryFromText(wkt [, srid])`  
Создает геометрическое значение любого типа, используя его WKT-формат и идентификатор SRID.
- `LineFromText(wkt[,srid]), LineStringFromText(wkt [, srid])`  
Создает значение LINESTRING, используя его WKT-формат и идентификатор SRID.
- `MLineFromText(wkt[,srid]), MultiLineStringFromText(wkt [, srid])`  
Создает значение MULTILINESTRING, используя его WKT-формат и идентификатор SRID.
- `MPointFromText(wkt[,srid]), MultiPointFromText(wkt [, srid])`  
Создает значение MULTIPOINT, используя его WKT-формат и идентификатор SRID.
- `MPolyFromText(wkt[,srid]), MultiPolygonFromText(wkt [, srid])`  
Создает значение MULTIPOLYGON, используя его WKT-формат и идентификатор SRID.
- `PointFromText(wkt[,srid])`  
Создает значение POINT, используя его WKT-формат и идентификатор SRID.
- `PolyFromText(wkt[,srid]), PolygonFromText(wkt [, srid])`  
Создает значение POLYGON, используя его WKT-формат и идентификатор SRID.

В спецификации OpenGIS также описываются и необязательные функции для создания значений Polygon и MultiPolygon, основанных на WKT-представлении коллекции колец или значений замкнутых объектов LineString. Эти значения могут пересекаться. В MySQL не реализованы следующие функции:

- `BdMPolyFromText(wkt, srid)`

Создает значение MultiPolygon из значения MultiLineString в WKT-формате, которое содержит произвольную коллекцию значений замкнутых объектов LineString.

- `BdPolyFromText(wkt, srid)`

Создает значение Polygon из значения MultiLineString в WKT-формате, которое содержит произвольную коллекцию значений замкнутых объектов LineString.

#### 7.4.2.2. Создание геометрических значений с помощью WKB-функций

MySQL предлагает набор функций, которые в качестве входных параметров принимают значения BLOB в WKB-формате, и, по выбору, идентификатор пространственной системы координат (SRID). Они возвращают значение соответствующей геометрии.

Функция `GeomFromWKT()` в качестве первого параметра принимает WKB-значения любого типа геометрических объектов. Реализация также обеспечивает отдельные для каждого типа функции-конструкторы, предназначенные для создания геометрических значений каждого типа геометрических объектов.

- `GeomCollFromWKB(wkb[, srid]), GeometryCollectionFromWKB(wkt[, srid])`

Создает значение GEOMETRYCOLLECTION, используя его WKB-формат и идентификатор SRID.

- `GeomFromWKB(wkb[, srid]), GeometryFromWKB(wkt[, srid])`

Создает геометрическое значение геометрии любого типа, используя его WKB-формат и идентификатор SRID.

- `LineFromWKB(wkb[, srid]), LineStringFromWKB(wkb[, srid])`

Создает значение LINESTRING, используя его WKB-формат и идентификатор SRID.

- `MLineFromWKB(wkb[, srid]), MultiLineStringFromWKB(wkb[, srid])`

Создает значение MULTILINESTRING, используя его WKB-формат и идентификатор SRID.

- `MPointFromWKB(wkb[, srid]), MultiPointFromWKB(wkb[, srid])`

Создает значение MULTIPOINT, используя его WKB-формат и идентификатор SRID.

- `MPolyFromWKB(wkb[, srid]), MultiPolygonFromWKB(wkb[, srid])`

Создает значение MULTIPOLYGON, используя его WKB-формат и идентификатор SRID.

- `PointFromWKB(wkb[, srid])`

Создает значение POINT, используя его WKB-формат и идентификатор SRID.

- `PolyFromWKB(wkb[, srid]), PolygonFromWKB(wkb[, srid])`

Создает значение POLYGON, используя его WKB-формат и идентификатор SRID.

В спецификации OpenGIS также описываются и необязательные функции для создания значений Polygon и MultiPolygon, основанных на WKB-представлении коллекции колец или значений замкнутых объектов LineString. Эти значения могут пересекаться. В MySQL не реализованы следующие функции:

- `BdMPolyFromWKB(wkb, srid)`

Создает значение MultiPolygon из значения MultiLineString в WKB-формате, которое содержит произвольную коллекцию значений замкнутых объектов LineString.

- `BdPolyFromWKB(wkb, srid)`

Создает значение Polygon из значения MultiLineString в WKB-формате, которое содержит произвольную коллекцию значений замкнутых объектов LineString.

### 7.4.2.3. Создание геометрических значений с помощью специальных MySQL-функций

#### На заметку!

Функции, перечисленные в этом разделе, в MySQL не реализованы.

MySQL предлагает набор полезных функций для создания WKB-представлений геометрических значений. Функции, описанные в этом разделе, являются расширениями MySQL для спецификаций OpenGIS. Результаты данных функций – это значения BLOB, содержащие WKB-представление геометрических значений без идентификатора пространственной системы координат (SRID). Результаты этих функций могут использоваться в качестве первого аргумента для любой функции из семейства `GeomFromWKB()`.

- `GeometryCollection(g1, g2, ...)`

Создает WKB-значение GeometryCollection. Если какой-нибудь из параметров имеет неправильный WKB-формат геометрического объекта, возвращается значение NULL.

- `LineString(pt1, pt2, ...)`

Создает WKB-значение LineString из ряда WKB-аргументов Point. Если хотя бы один из аргументов не является WKB-значением Point, возвращается значение NULL. Если количество аргументов Point меньше двух, возвращаемым значением будет NULL.

- `MultiLineString(ls1, ls2, ...)`

Создает WKB-значение (объекта) MultiLineString, используя WKB-аргументы LineString. Если хотя бы один из аргументов не является WKB-значением LineString, возвращается значение NULL.

- `MultiPoint(pt1, pt2, ...)`

Создает WKB-значение MultiPoint, используя WKB-аргументы Point. Если хотя бы один из аргументов не является WKB-значением Point, возвращается значение NULL.

- `MultiPolygon(poly1, poly2, ...)`

Создает WKB-значение MultiPolygon из набора WKB-аргументов Polygon. Если хотя бы один из аргументов не является WKB-значением Polygon, возвращается значение NULL.

- `Point(x, y)`

Создает WKB-значение Point, используя координаты объекта Point.

- `Polygon(ls1, ls2, ...)`

Создает WKB-значение `Polygon` из ряда WKB- аргументов `LineString`. Если хотя бы один из аргументов не является WKB-значением объекта `LinearRing` (то есть незамкнутого простого объекта `LineString`), возвращается значение `NULL`.

### 7.4.3. Создание пространственных столбцов

MySQL предлагает стандартный путь создания пространственных столбцов для разных типов геометрий, например, с помощью операторов `CREATE TABLE` или `ALTER TABLE`. В настоящее время пространственные столбцы поддерживаются только для таблиц `MyISAM`.

- Для создания таблицы с пространственным столбцом используйте оператор `CREATE TABLE`:

```
mysql> CREATE TABLE geom (g GEOMETRY);
Query OK, 0 rows affected (0.02 sec)
```

- Чтобы добавить пространственный столбец в существующую таблицу или удалить его, используйте оператор `ALTER TABLE`:

```
mysql> ALTER TABLE geom ADD pt POINT;
Query OK, 0 rows affected (0.00 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE geom DROP pt;
Query OK, 0 rows affected (0.00 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

### 7.4.4. Заполнение пространственных столбцов

После того, как пространственные столбцы созданы, их можно заполнить пространственными данными.

Значения должны сохраняться во внутреннем формате геометрического объекта, но в такой формат их можно преобразовывать и из форматов WKT и WKB. Следующие примеры иллюстрируют способы вставки геометрических значений в таблицу путем преобразования WKT-значений во внутренний геометрический формат.

Преобразование можно выполнять непосредственно через оператор `INSERT`:

```
INSERT INTO geom VALUES (GeomFromText('POINT(1 1)'));
SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (GeomFromText(@g));
```

Или же можно преобразовать значения в нужный формат и перед выполнением оператора `INSERT`:

```
SET @g = GeomFromText('POINT(1 1)');
INSERT INTO geom VALUES (@g);
```

В следующих примерах выполняется вставка в таблицу более сложных геометрических значений:

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (GeomFromText(@g));
SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (GeomFromText(@g));
```

```
SET @g =
  'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomFromText(@g));
```

Во всех приведенных выше примерах для создания геометрических значений используется функция `GeomFromText()`. Также можно применять и специальные для каждого типа функции:

```
SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (PointFromText(@g));

SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (LineStringFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (PolygonFromText(@g));

SET @g =
  'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomCollFromText(@g));
```

Обратите внимание, что если клиентская прикладная программа требует использовать WKB-формат геометрических значений, именно она и будет отвечать за отправку на сервер корректно оформленных WKB-значений в запросах. Однако существует несколько способов удовлетворить такое требование. Например:

- Можно вставить значение `POINT (1 1)`, используя шестнадцатеричный литеральный/буквенный синтаксис:

```
mysql> INSERT INTO geom VALUES
-> (GeomFromWKB(0x010100000000000000000000F03F000000000000F03F));
```

- Приложение ODBC может посылать значения в WKB-формате, привязывая их к заполнителю с помощью аргумента типа `BLOB`:

```
INSERT INTO geom VALUES (GeomFromWKB(?))
```

Другие интерфейсы программирования могут поддерживать подобный механизм заполнителя.

- В программе C можно отменить двоичное значение с помощью `mysql_real_escape_string()` и включить результат в отсылаемую на сервер строку запроса.

## 7.4.5. Выборка пространственных данных

Геометрические значения, сохраненные в таблице, могут быть выбраны во внутреннем формате. Также их можно преобразовывать в формат WKT или WKB.

### 7.4.5.1. Выборка пространственных данных во внутреннем формате

Выборка геометрических значений с использованием внутреннего формата может пригодиться при переходах от таблицы к таблице:

```
CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
```

### 7.4.5.2. Выборка пространственных данных в WKT-формате

Функция `AsText()` преобразовывает значение геометрического объекта из внутреннего формата в WKT-строку.

```
mysql> SELECT AsText(g) FROM geom;
+-----+
| AsText(p1) |
+-----+
| POINT(1 1) |
| LINESTRING(0 0,1 1,2 2) |
+-----+
```

### 7.4.5.3. Выборка пространственных данных в WKB-формате

Функция `AsBinary()` преобразует геометрический объект из внутреннего формата в содержащий WKB-значение объект `BLOB`.

```
SELECT AsBinary(g) FROM geom;
```

## 7.5. Анализ пространственной информации

После заполнения пространственных столбцов значениями можно делать запросы и анализировать их. MySQL предлагает набор функций для выполнения различных операций над пространственными данными. Эти функции можно разделить на четыре основных категории согласно типу выполняемой ими операции:

- Функции, которые преобразуют геометрические объекты из одного формата в другой.
- Функции, которые обеспечивают доступ к качественным или количественным свойствам того или иного геометрического объекта.
- Функции, которые описывают взаимосвязь между двумя геометриями.
- Функции, которые создают новые геометрии из уже существующих.

Функции пространственного анализа могут использоваться в различных контекстах:

- В любой интерактивной SQL-программе, подобной `mysql` или `MySQLCC`.
- В прикладных программах, написанных на любом языке, который поддерживает клиентский API-интерфейс MySQL.

### 7.5.1. Функции преобразования формата геометрических объектов

MySQL поддерживает следующие функции для преобразования формата значений геометрических объектов с внутреннего в WKT- или WKB-формат и наоборот:

- `AsBinary(g)`. Преобразует значение во внутреннем геометрическом формате в WKB-формат и возвращает двоичный результат.
- `AsText(g)`. Преобразует значение во внутреннем геометрическом формате геометрии в WKT-формат и возвращает результат в виде строки.

```
mysql> SET @g = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(GeomFromText(@g));
+-----+
| AsText(GeomFromText(@g)) |
+-----+
| LINESTRING(1 1,2 2,3 3) |
+-----+
```

- `GeomFromText(wkt[,srid])`. Преобразует строковое значение из WKT-формата во внутренний геометрический формат и возвращает результат. Также поддерживается набор специальных для каждого типа функций, таких как `PointFromText()` и `LineFromText()`; см. раздел 7.4.2.1.
- `GeomFromWKB(wkb[,srid])`. Преобразует двоичное значение из WKB-формата во внутренний геометрический формат и возвращает результат. Также поддерживается набор специальных для каждого типа функций, таких как `PointFromWKB()` и `LineFromWKB()`; см. раздел 7.4.2.2.

## 7.5.2. Геометрические функции

Каждая функция, принадлежащая данной группе, принимает геометрическое значение в качестве аргумента и возвращает некоторое качественное или количественное свойство геометрии. Некоторые функции ограничивают тип аргументов. Такие функции возвращают значение `NULL`, если геометрический тип аргумента указан неверно. Например, функция `Area()` возвратит `NULL`, если тип объекта не соответствует ни `Polygon`, ни `MultiPolygon`.

### 7.5.2.1. Общие функции геометрических объектов

Функции, перечисленные в этом разделе, не ограничивают аргументы и принимают геометрическое значение любого типа.

- `Dimension(g)`. Возвращает унаследованное измерение геометрического значения `g`. Результатом может быть -1, 0, 1 или 2. (Описание смысла этих значений дается в разделе 7.2.2.)

```
mysql> SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+-----+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+-----+
| 1 |
+-----+
```

- `Envelope(g)`. Возвращается минимальный ограничивающий прямоугольник (MBR) для геометрического значения `g`. Результат возвращается в виде значения `Polygon`.

```
mysql> SELECT AsText(Envelope(GeomFromText('LineString(1 1,2 2)')));
+-----+
| AsText(Envelope(GeomFromText('LineString(1 1,2 2)')) |
+-----+
| POLYGON((1 1,2 1,2 2,1 2,1 1)) |
+-----+
```

Границы многоугольника определяются угловыми точками ограничивающего прямоугольника:

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

- `GeometryType(g)`. Возвращает в виде строки название геометрического типа, к которому относится экземпляр геометрии `g`. Название будет соответствовать названию одного из подклассов с возможностью создания экземпляров класса `Geometry`.

```
mysql> SELECT GeometryType(GeomFromText('POINT(1 1)'));
+-----+
| GeometryType(GeomFromText('POINT(1 1)')) |
+-----+
| POINT                                     |
+-----+
```

- **SRID(*g*)**. Возвращает целое число, обозначающее идентификатор пространственной системы координат (SRID) для геометрии *g*.

```
mysql> SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
+-----+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+-----+
| 101                                           |
+-----+
```

В спецификации OpenGIS также определены следующие функции, которые в MySQL пока не реализованы:

- **Boundary(*g*)**. Возвращает геометрический объект, который является замыканием комбинаторной границы геометрического значения *g*.
- **IsEmpty(*g*)**. Возвращает 1, если значение геометрии *g* является пустой геометрией, 0 – если не пустой, и -1 – если аргумент равен NULL. Если геометрический объект пустой, он представляет собой пустое множество точек.
- **IsSimple(*g*)**. В настоящее время эта функция является заполнителем и не должна использоваться. Ниже представлено ее описание на тот случай, если она все-таки будет применяться.

Функция возвращает 1, если значение геометрии *g* не содержит никаких аномальных геометрических точек, подобных самопересечению или самокасания. **IsSimple()** возвращает 0, если аргумент не является простым, и -1, если аргумент равен NULL.

В описании каждого геометрического класса с возможностью создания экземпляров, предложенном ранее в этой главе, перечислены определенные условия, при которых экземпляры того или иного класса будут классифицироваться как непростые.

### 7.5.2.2. Функции Point

**Point** состоит из координат *X* и *Y*, которые можно получить с помощью следующих функций:

- **X(*p*)**. Возвращает значение координаты *X* для точки *p* в виде числа с двойной точностью.

```
mysql> SELECT X(GeomFromText('Point(56.7 53.34)'));
+-----+
| X(GeomFromText('Point(56.7 53.34)')) |
+-----+
| 56.7 |
+-----+
```

- **Y(*p*)**. Возвращает значение координаты *Y* для точки *p* в виде числа с двойной точностью.



```
mysql> SELECT Y(GeomFromText('Point(56.7 53.34)'));
+-----+
| Y(GeomFromText('Point(56.7 53.34)')) |
+-----+
| 53.34 |
+-----+
```

### 7.5.2.3. Функции LineString

Объект LineString состоит из значений Point. Можно извлекать определенные точки объекта LineString, подсчитывать количество точек, из которых он состоит, или получать значение его длины.

- **EndPoint(*ls*)**. Возвращает точку Point, которая является конечной точкой LineString-значения *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(EndPoint(GeomFromText(@ls)));
+-----+
| AsText(EndPoint(GeomFromText(@ls))) |
+-----+
| POINT(3 3) |
+-----+
```

- **GLength(*ls*)**. Возвращает в виде числа с удвоенной точностью длину LineString-значения *ls* со связанной с ним пространственной ссылкой.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT GLength(GeomFromText(@ls));
+-----+
| GLength(GeomFromText(@ls)) |
+-----+
| 2.8284271247462 |
+-----+
```

- **IsClosed(*ls*)**. Возвращает 1, если LineString-значения *ls* является замкнутым (то есть значения StartPoint() и EnPoint() совпадают). Возвращает 0, если *ls* не является замкнутым, и -1, если *ls* соответствует NULL.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT IsClosed(GeomFromText(@ls));
+-----+
| IsClosed(GeomFromText(@ls)) |
+-----+
| 0 |
+-----+
```

- **NumPoints(*ls*)**. Возвращает число точек в LineString-объекте *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT NumPoints(GeomFromText(@ls));
+-----+
| NumPoints(GeomFromText(@ls)) |
+-----+
| 3 |
+-----+
```

- **PointN(*ls*,*n*)**. Возвращает *n*-ную точку в LineString-значении *ls*. Нумерация точек начинается с 1.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(PointN(GeomFromText(@ls),2));
+-----+
| AsText(PointN(GeomFromText(@ls),2)) |
+-----+
| POINT(2 2) |
+-----+
```

- **StartPoint(*ls*)**. Возвращает точку *Point*, которая является начальной точкой *LineString*-значения *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(StartPoint(GeomFromText(@ls)));
+-----+
| AsText(StartPoint(GeomFromText(@ls))) |
+-----+
| POINT(1 1) |
+-----+
```

В спецификации OpenGIS также определена следующая функция, которая MySQL пока не реализована:

- **IsRing(*ls*)**. Возвращает 1, если *LineString*-объект *ls* является замкнутым (то есть значения *StartPoint()* и *EndPoint()* совпадают) и простым (линия не проходит в одной и той же точке более одного раза). Возвращает 0, если *ls* – это не кольцо, и -1, если *ls* равно NULL.

#### 7.5.2.4. Функции **MultiLineString**

- **GLength(*m*ls)**. Возвращает в виде числа с удвоенной точностью значение длины *MultiLineString*-объекта *m*ls. Длина *m*ls равна сумме длин составляющих его элементов.

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT GLength(GeomFromText(@mls));
+-----+
| GLength(GeomFromText(@mls)) |
+-----+
| 4.2426406871193 |
+-----+
```

- **IsClosed(*m*ls)**. Возвращает 1, если *MultiLineString*-объект *m*ls является замкнутым (то есть значения *StartPoint()* и *EndPoint()* одинаковы для каждого объекта *LineString* в *m*ls). Возвращает 0, если *m*ls замкнутым не является, и -1, если значение *m*ls равно NULL.

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT IsClosed(GeomFromText(@mls));
+-----+
| IsClosed(GeomFromText(@mls)) |
+-----+
| 0 |
+-----+
```

### 7.5.2.5. Функции Polygon

- **Area(*poly*)**. Возвращает в виде числа с удвоенной точностью значение области Polygon-объекта *poly*, в соответствии с его размерами в пространственной системе координат.

```
mysql> SET @poly = 'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1))';
mysql> SELECT Area(GeomFromText(@poly));
+-----+
| Area(GeomFromText(@poly)) |
+-----+
| 4 |
+-----+
```

- **ExteriorRing(*poly*)**. Возвращает внешнее кольцо Polygon-объекта *poly* в виде LineString.

```
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) |
+-----+
```

- **InteriorRingN(*poly*,*n*)**. Возвращает *n*-ное внутреннее кольцо для Polygon-значения *poly* в виде LineString. Нумерация колец начинается с 1.

```
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) |
+-----+
```

- **NumInteriorRings(*poly*)**. Возвращает количество внутренних колец в Polygon-объекте *poly*.

```
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 |
+-----+
```

### 7.5.2.6. Функции MultiPolygon

- **Area(*mpoly*)**. Возвращает в виде числа двойной точности область MultiPolygon-объекта *mpoly*, в соответствии с его размерами в пространственной системе координат.

```
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 |
+-----+

```

В спецификации OpenGIS также определены следующие функции, которые в MySQL пока не реализованы:

- `Centroid(mpoly)`. Возвращает математический центр тяжести для MultiPolygon-объекта *mpoly* в виде Point. Не гарантируется, что результат будет находиться в пределах MultiPolygon.
- `PointOnSurface(mpoly)`. Возвращает значение Point, которое гарантированно будет находиться в пределах MultiPolygon-объекта *mpoly*.

### 7.5.2.7. Функции GeometryCollection

- `GeometryN(gc, n)`. Возвращает *n*-ый геометрический объект в GeometryCollection-объекте *gc*. Нумерация геометрических объектов начинается с 1.

```

mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT AsText(GeometryN(GeomFromText(@gc),1));

```

```

+-----+
| AsText(GeometryN(GeomFromText(@gc),1)) |
+-----+
| POINT(1 1) |
+-----+

```

- `NumGeometries(gc)`. Возвращает количество геометрических объектов в GeometryCollection-значении *gc*.

```

mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2,3 3))';
mysql> SELECT NumGeometries(GeomFromText(@gc));

```

```

+-----+
| NumGeometries(GeomFromText(@gc)) |
+-----+
|                               2 |
+-----+

```

## 7.5.3. Функции для создания новых геометрий из существующих

### 7.5.3.1. Геометрические функции для создания новых геометрий

В разделе 7.5.2 уже обсуждались некоторые функции, с помощью которых можно создавать новые геометрии из существующих:

- `Envelope(g)`
- `StartPoint(ls)`
- `EndPoint(ls)`
- `PointN(ls, n)`
- `ExteriorRing(poly)`
- `InteriorRingN(poly, n)`
- `GeometryN(gc, n)`

### 7.5.3.2. Пространственные операторы

OpenGIS предлагает ряд других функций для создания геометрий. В этих функциях используются пространственные операции.

В MySQL данные функции пока не реализованы, возможно, они появятся в будущих версиях.

- `Buffer(g, d)`. Возвращает геометрию, представляющую все точки, расстояние которых от значения геометрии *g* меньше или равно расстоянию *d*.
- `ConvexHull(g)`. Возвращает геометрию, которая представляет собой выпуклую оболочку геометрии *g*.
- `Difference(g1, g2)`. Возвращает геометрию, которая представляет разность множества точек геометрии *g1* и *g2*.
- `Intersection(g1, g2)`. Возвращает геометрию, которая представляет пересечение множества точек геометрий *g1* и *g2*.
- `SymDifference(g1, g2)`. Возвращает геометрию, которая представляет симметрическую разность множества точек геометрий *g1* и *g2*.
- `Union(g1, g2)`. Возвращает геометрию, которая представляет объединение множества точек геометрий *g1* и *g2*.

### 7.5.4. Функции для проверки пространственных отношений между геометрическими объектами

Функции, описанные в этих разделах, принимают две геометрии в качестве входных параметров и возвращают значение качественного или количественного отношения между ними.

### 7.5.5. Отношение минимальных ограничивающих прямоугольников

MySQL предлагает некоторые функции, с помощью которых можно проверять отношение между минимальными ограничивающими прямоугольниками (Minimum Bounding Rectangle – MBR) двух геометрий *g1* и *g2*. Они включают:

- `MBRContains(g1, g2)`. Возвращает значение 1 или 0, указывающее, содержит ли минимальный ограничивающий прямоугольник геометрии *g1* минимальный ограничивающий прямоугольник геометрии *g2*.

```
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 |
+-----+
```

- `MBRDisjoint(g1, g2)`. Возвращает значение 1 или 0, указывающее, разделены ли (не пересекаются) минимальные ограничивающие прямоугольники (MBR) двух геометрий *g1* и *g2*.

- **MBREqual(*g1*,*g2*)**. Возвращает значение 1 или 0, указывающее, являются ли минимальные ограничивающие прямоугольники двух геометрий *g1* и *g2* одинаковыми.
- **MBRIntersects(*g1*,*g2*)**. Возвращает значение 1 или 0, указывающее, пересекаются ли минимальные ограничивающие прямоугольники двух геометрий *g1* и *g2*.
- **MBROverlaps(*g1*,*g2*)**. Возвращает значение 1 или 0, указывающее, перекрывают ли друг друга минимальные ограничивающие прямоугольники двух геометрий *g1* и *g2*.
- **MBRTouches(*g1*,*g2*)**. Возвращает значение 1 или 0, указывающее, соприкасаются ли минимальные ограничивающие прямоугольники двух геометрий *g1* и *g2*.
- **MBRWithin(*g1*,*g2*)**. Возвращает значение 1 или 0, указывающее, находится ли минимальный ограничивающий прямоугольник геометрии *g1* внутри минимального ограничивающего прямоугольника геометрии *g2*.

```
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 |
+-----+-----+
```

### 7.5.6. Функции для проверки пространственных отношений между геометриями

В спецификации OpenGIS определены следующие функции. В настоящее время MySQL не реализует их согласно спецификации. Те из них, которые введены в MySQL, возвращают тот же результат, что и соответствующие функции для MBR. Сюда относятся представленные в следующем ниже списке функции, за исключением `Distance()` и `Related()`.

Вполне возможно, что эти функции будут введены в последующих версиях, включая полную поддержку для пространственного анализа, а не только поддержку для MBR-функций.

Функции работают с двумя значениями геометрий *g1* и *g2*.

- **Contains(*g1*,*g2*)**. Возвращает значение 1 или 0, указывающее, включает в себя геометрия *g1* геометрию *g2* полностью.
- **Crosses(*g1*,*g2*)**. Возвращает значение 1, если *g1* пространственно пересекает *g2*. Возвращает значение NULL, если *g1* является Polygon или MultiPolygon, или если *g2* является Point или MultiPoint. В противном случае возвращается значение 0. Выражение “пространственно пересекает” обозначает пространственное отношение между двумя данными геометриями, характеризующееся следующими свойствами:
  - Эти две геометрии пересекаются.
  - В результате их пересечения образуется геометрия, значение измерения которой на один меньше максимального значения измерения обеих данных геометрий.
  - Их пересечение не равно ни одной из двух данных геометрий.
- **Disjoint(*g1*,*g2*)**. Возвращает значение 1 или 0, указывающее, отделена ли пространственно геометрия *g1* от геометрии *g2* (то есть, пересекает ли она ее или нет).

- `Distance(g1, g2)`. Возвращает в виде числа двойной точности значение наименьшего расстояния между любыми двумя точками в двух геометриях.
- `Equals(g1, g2)`. Возвращает значение 1 или 0, указывающее, является ли *g1* пространственно равной *g2*.
- `Intersects(g1, g2)`. Возвращает значение 1 или 0, указывающее, пересекает ли пространственно геометрия *g1* геометрию *g2*.
- `Overlaps(g1, g2)`. Возвращает значение 1 или 0, указывающее, перекрывает ли пространственно геометрия *g1* геометрию *g2*. Термин “пространственно перекрывает” используется, если две геометрии пересекаются и в результате их пересечения образуется геометрия такого же измерения, но не равная ни одной из двух данных геометрий.
- `Related(g1, g2, pattern_matrix)`. Возвращает значение 1 или 0, указывающее, существует ли пространственное соотношение, заданное в *pattern\_matrix*, между *g1* и *g2*. Возвращает значение -1, если аргументы равны NULL. Матрица образов *pattern\_matrix* представляет собой строку. Требования для нее будут определены после реализации функции.
- `Touches(g1, g2)`. Возвращает значение 1 или 0, указывающее, соприкасается ли пространственно *g1* с *g2*. Две геометрии пространственно соприкасаются, если их внутренние пространства не пересекаются, но граница одной из геометрий пересекает либо границу, либо внутреннее пространство другой геометрии.
- `Within(g1, g2)`. Возвращает значение 1 или 0, указывающее, находится ли *g1* пространственно в пределах *g2*.

## 7.6. Оптимизация пространственного анализа

Операции поиска в непространственных базах данных можно оптимизировать, используя индексы. Такой способ подходит также и для пространственных баз данных. Благодаря большому разнообразию уже разработанных методов многомерной индексации, появилась возможность оптимизировать пространственный поиск.

Наиболее типичными из них являются:

- Запросы точки, которые выполняют поиск всех объектов, содержащих данную точку.
- Запросы области, которые выполняют поиск всех объектов, находящихся в данной области.

Для индексации пространственных столбцов MySQL использует **R-деревья с квадратичным разбиением**. Построение пространственного индекса выполняется с помощью MBR геометрического объекта. Для большинства геометрий MBR – это минимальный окружающий их прямоугольник. Для горизонтальной или вертикальной ломанной кривой MBR – это прямоугольник, вырожденный в ломанную кривую, а для точки – прямоугольник, вырожденный в точку.

### 7.6.1. Создание пространственных индексов

В MySQL можно создавать пространственные индексы, используя синтаксис, подобный синтаксису для создания стандартных индексов, но расширяемый с помощью ключевого слова **SPATIAL**. Индексируемые на данный момент пространственные столбцы

должны быть объявлены как NOT NULL. Следующие примеры иллюстрируют способы создания пространственных индексов.

- С помощью CREATE TABLE.

```
mysql> CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g));
```

- С помощью ALTER TABLE.

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
```

- С помощью CREATE INDEX.

```
mysql> CREATE SPATIAL INDEX sp_index ON geom (g);
```

Для удаления пространственных индексов используйте ALTER TABLE или DROP INDEX.

- С помощью ALTER TABLE.

```
mysql> ALTER TABLE geom DROP INDEX g;
```

- С помощью DROP INDEX.

```
mysql> DROP INDEX sp_index ON geom;
```

Для примера предположим, что таблица geom содержит более 32 000 геометрий, которые сохранены в столбце g типа GEOMETRY. Таблица также включает столбец fid типа AUTO\_INCREMENT, предназначенный для хранения идентификаторов объектов.

```
mysql> DESCRIBE geom;
```

Field	Type	Null	Key	Default	Extra
fid	int(11)		PRI	NULL	auto_increment
g	geometry				

2 rows in set (0.00 sec)

```
mysql> SELECT COUNT(*) FROM geom;
```

count(*)	32376
----------	-------

1 row in set (0.00 sec)

Чтобы добавить пространственный индекс по столбцу g, воспользуйтесь следующим оператором:

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
```

Query OK, 32376 rows affected (4.05 sec)

Records: 32376 Duplicates: 0 Warnings: 0

## 7.6.2. Использование пространственного индекса

Оптимизатор проверяет, могут ли доступные пространственные индексы быть включены в поиск запросов, использующих функцию, такую как MBRContains() или MBRWithin() в конструкции WHERE. Например, нам нужно найти все находящиеся в данном прямоугольнике объекты:

```
mysql> SELECT fid,AsText(g) FROM geom WHERE
```



```
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000
    16000,30000 16000,30000 15000))'),g);
```

fid	AsText(g)
21	LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30333.8 15828.8)
22	LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8,30334 15871.4)
23	LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4,30334 15914.2)
24	LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4,30273.4 15823)
25	LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882.4,30274.8 15866.2)
26	LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4,30275 15918.2)
249	LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946.8,30320.4 15938.4)
1	LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136.4,30240 15127.2)
2	LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136,30210.4 15121)
3	LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,30169 15113)
4	LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30157 15111.6)
5	LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4,30194.2 15075.2)
6	LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,30244.6 15077)
7	LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8,30201.2 15049.4)
10	LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6,30189.6 15019)
11	LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2,30151.2 15009.8)
13	LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,30114.6 15067.8)
154	LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30278 15134)
155	LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30259 15083.4)
157	LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4,30128.8 15001)

20 rows in set (0.00 sec)

Теперь воспользуемся EXPLAIN для проверки способа выполнения этого запроса (столбец вывода id был удален, поэтому выходные данные лучше размещаются на странице):

```
mysql> EXPLAIN SELECT fid,AsText(g) FROM geom WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000
    16000,30000 16000,30000 15000))'),g);
```

select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
SIMPLE	geom	range	g	g	32	NULL	50	Using where

1 row in set (0.00 sec)

Теперь проверим, что произойдет при отсутствии пространственного индекса:

```
mysql> EXPLAIN SELECT fid,AsText(g) FROM g IGNORE INDEX (g) WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000
    16000,30000 16000,30000 15000))'),g);
```

select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
SIMPLE	geom	ALL	NULL	NULL	NULL	NULL	32376	Using where

1 row in set (0.00 sec)

Выполним оператор SELECT, игнорируя существующий пространственный индекс:

```
mysql> SELECT fid,AsText(g) FROM geom IGNORE INDEX (g) WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000
    16000,30000 16000,30000 15000))'),g);
```

fid	AsText(g)
1	LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136.4,30240 15127.2)
2	LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136,30210.4 15121)
3	LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,30169 15113)
4	LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30157 15111.6)
5	LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4,30194.2 15075.2)
6	LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,30244.6 15077)
7	LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8,30201.2 15049.4)
10	LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6,30189.6 15019)
11	LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2,30151.2 15009.8)
13	LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,30114.6 15067.8)
21	LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30333.8 15828.8)
22	LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8,30334 15871.4)
23	LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4,30334 15914.2)
24	LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4,30273.4 15823)
25	LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882.4,30274.8 15866.2)
26	LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4,30275 15918.2)
154	LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30278 15134)
155	LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30259 15083.4)
157	LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4,30128.8 15001)
249	LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946.8,30320.4 15938.4)

20 rows in set (0.46 sec)

Когда индекс не используется, время выполнения данного запроса увеличивается с 0.00 секунд до 0.46 секунды.

В следующих версиях пространственные индексы также смогут использоваться и для оптимизации других функций (см. раздел 7.5.4).

## 7.7. Соответствие и совместимость MySQL

### 7.7.1. Функции геоинформационных систем, которые пока не реализованы

#### ■ Дополнительные виды метаданных

В спецификации OpenGIS предложено еще несколько дополнительных видов метаданных. Например, системное представление с именем `GEOMETRY_COLUMNS` включает описание столбцов геометрий: одна строка для каждого столбца геометрических объектов в базе данных.

#### ■ OpenGIS-функция `Length()` для `LineString` и `MultiLineString` в настоящее время должна вызываться в MySQL как `GLength()`.

Проблема заключается в том, что уже имеется SQL-функция `Length()`, которая вычисляет длину строковых значений, и порой просто невозможно определить, в каком контексте эта функция вызывается – в текстовом или пространственном. Необходимо как-то решить упомянутую проблему или присвоить функции другое имя.

## Хранимые процедуры и функции

**Х**ранимые процедуры и функции – это новая возможность, появившаяся в версии MySQL 5.0. Хранимая процедура представляет собой набор SQL-операторов, которые можно сохранять на сервере. После того, как это будет сделано, клиентам больше не придется повторно задавать одни и те же отдельные операторы; вместо этого они смогут обращаться к хранимой процедуре.

Ситуации, когда хранимые процедуры могут оказаться особенно полезными, таковы:

- Многочисленные клиентские приложения написаны на разных языках или работают на различных платформах, но должны выполнять одинаковые операции с базами данных.
- Безопасность играет первостепенную роль. В банках, например, хранимые процедуры используются для всех стандартных операций. Это обеспечивает совместимость и безопасность среды, а процедуры гарантируют надлежащую регистрацию каждой операции. При таком типе установки приложения и пользователи не получают непосредственный доступ к таблицам базы данных и могут выполнять только конкретные хранимые процедуры.

Хранимые процедуры помогают повысить эффективность, поскольку при их использовании объем пересылаемой между сервером и клиентом информации существенно снижается. Обратная сторона состоит в том, что это увеличивает нагрузку на систему сервера баз данных, так как в этом случае на стороне клиента (приложения) выполняется меньшая часть работы, а на стороне сервера – большая. Обязательно учитывайте это, если сразу большое количество клиентских машин (таких как Web-серверы) обслуживает только один или всего несколько серверов баз данных.

Хранимые процедуры также позволяют создавать библиотеки функций на сервере баз данных. Эта опция, используемая современными языками приложений, допускает внутреннее применение такого проекта, например, для классов. Для программиста будет выгодным использование этих особенностей языка клиентских приложений даже вне контекста баз данных.

MySQL для хранимых процедур применяет синтаксис SQL:2003, который также используется в IBM DB2.

Внедрение хранимых процедур MySQL пока еще находится в процессе разработки. Весь описываемый в данной главе синтаксис поддерживается, любые ограничения и расширения там, где это нужно, документированы.

Хранимые процедуры требуют наличия в базе данных mysql таблицы `procs`. Эта таблица создается во время процесса установки MySQL 5.0. При модернизации с более ранней версии до MySQL 5.0 не забудьте обновить таблицы привилегий, дабы иметь уверенность, что таблица `procs` существует.

## 8.1. Синтаксис хранимой процедуры

Хранимые процедуры и функции представляют собой подпрограммы, создаваемые с помощью операторов `CREATE PROCEDURE` и `CREATE FUNCTION`. Подпрограмма является либо процедурой, либо функцией. Процедура вызывается с помощью оператора `CALL` и может только передавать значения обратно, используя выходные переменные. Функции могут возвращать скалярное значение и вызываются из оператора точно так же, как и любые другие функции (то есть, через указание имени функции). Хранимые процедуры могут вызывать другие хранимые процедуры.

В настоящее время MySQL поддерживает контекст только для базы данных по умолчанию. То есть при задании в процедуре `USE имя_базы_данных` на выходе подпрограммы восстанавливается исходная база данных по умолчанию. Подпрограмма наследует базу данных по умолчанию от вызывающего оператора, поэтому в целом в процедурах либо должен использоваться оператор `USE имя_базы_данных`, либо все таблицы должны указываться с явной ссылкой на базу данных, то есть, `имя_базы_данных.имя_таблицы`.

В MySQL поддерживается самое удобное расширение, позволяющее использовать обычные операторы `SELECT` (применять курсоры и локальные переменные не нужно) внутри хранимой процедуры. Набор результатов такого запроса просто посылается непосредственно клиенту. Множественные операторы `SELECT` генерируют и множественные наборы результатов, поэтому клиенту следует использовать клиентскую библиотеку MySQL, поддерживающую множественные наборы результатов. Это означает, что клиентская библиотека должна быть из версии MySQL, по крайней мере, не ниже 4.1.

В следующем разделе описывается синтаксис, применяемый для создания, изменения, удаления и запроса хранимых процедур и функций.

### 8.1.1. Обслуживание хранимых процедур

#### 8.1.1.1. `CREATE PROCEDURE` и `CREATE FUNCTION`

```
CREATE PROCEDURE имя_хранимой_процедуры ([параметр[,...]])  
    [характеристика ...] тело_процедуры
```

```
CREATE FUNCTION имя_хранимой_процедуры ([параметр[,...]])  
    [RETURNS тип]  
    [характеристика ...] тело_процедуры
```

параметр:

```
[ IN | OUT | INOUT ] имя_параметра тип
```

тип:

Любой допустимый тип данных MySQL

Характеристика:

```
LANGUAGE SQL
| [NOT] DETERMINISTIC
| SQL SECURITY {DEFINER | INVOKER}
| COMMENT 'строка'
```

тело\_процедуры:

Допустимый оператор (операторы) SQL процедуры

Конструкция RETURNS может быть определена только для FUNCTION. Она используется для указания типа результата функции, при этом в теле функции должен присутствовать оператор RETURN значение.

Список аргументов, заключенный в круглые скобки, должен присутствовать всегда. Если аргументы отсутствуют, следует использовать пустой список аргументов (). Каждый аргумент по умолчанию является аргументом IN. Чтобы по-другому определить аргумент, перед его названием укажите ключевое слово OUT или INOUT. Значения IN, OUT или INOUT являются допустимыми только для PROCEDURE.

Оператор CREATE FUNCTION используется в более ранних версиях MySQL для поддержки функций UDF (определяемых пользователем). UDF-функции продолжают поддерживаться даже с появлением хранимых функций. UDF-функция может рассматриваться как внешняя хранимая функция. Однако стоит обратить внимание на то, что хранимые функции и функции, определяемые пользователем, разделяют одно и то же пространство имен.

Структура для внешних хранимых процедур будет представлена в ближайшем будущем. Это позволит записывать хранимые процедуры на языках, отличных от SQL. Скорее всего, одним из первых поддерживаемых языков станет PHP, потому что базовый механизм PHP невелик по размерам, безопасен в отношении потоков и легко встраивается. Поскольку структура будет общедоступной, ожидается поддержка и многих других языков.

Функция считается “детерминированной”, если она всегда возвращает один и тот же результат для одних и тех же входных аргументов, в противном случае функция является “недетерминированной”. В настоящее время характеристика DETERMINISTIC уже воспринимается, но еще не используется оптимизатором.

Характеристика SQL SECURITY может применяться для определения, должна ли процедура выполняться с использованием привилегий пользователя, создающего эту процедуру, или привилегий пользователя, ее вызывающего. Значение по умолчанию — DEFINER. Это новая функция в SQL:2003.

MySQL пока еще не использует привилегию GRANT EXECUTE.

MySQL поддерживает системную переменную sql\_mode, действующую во время создания процедуры, и при выполнении данной процедуры будет всегда применять именно то значение.

Конструкция COMMENT является расширением MySQL и может использоваться для описания хранимой процедуры. Такая информация отображается операторами SHOW CREATE PROCEDURE и SHOW CREATE FUNCTION.

MySQL разрешает, чтобы подпрограммы содержали DDL-операторы (такие как CREATE и DROP) и SQL-операторы транзакций (такие как COMMIT). Стандарт этого не требует, поэтому все зависит от реализации.

**На заметку!**

В настоящее время хранимые функции, создаваемые с помощью CREATE FUNCTION, не могут содержать ссылки на таблицы. Пожалуйста, обратите внимание, что это включает некоторые операторы SET, но и исключает некоторые операторы SELECT. Данные ограничения будут сняты, как только это станет возможным.

Ниже представлен пример простой хранимой процедуры с аргументом OUT. В примере во время определения процедуры используется команда delimiter клиента mysql для изменения разделителя оператора с ; на //. Это позволяет передать на сервер разделитель ;, указанный в теле процедуры, тем самым освобождая mysql от самостоятельной интерпретации данного разделителя.

```
mysql> delimiter //
mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
-> BEGIN
-> SELECT COUNT(*) INTO param1 FROM t;
-> END
-> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @a;
+-----+
| @a    |
+-----+
| 3     |
+-----+
1 row in set (0.00 sec)
```

Ниже представлен пример функции, которая принимает аргумент, выполняет операцию с помощью SQL-функции и возвращает результат:

```
mysql> delimiter //
mysql> CREATE FUNCTION hello (s CHAR(20)) RETURNS CHAR(50)
-> RETURN CONCAT('Hello, ',s,'!');
-> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
mysql> SELECT hello('world');
+-----+
| hello('world') |
+-----+
| Hello, world!  |
+-----+
1 row in set (0.00 sec)
```

**8.1.1.2. ALTER PROCEDURE и ALTER FUNCTION**

ALTER {PROCEDURE | FUNCTION} имя\_хранимой\_процедуры [характеристика ...]

характеристика:

```
NAME новое_имя  
| SQL SECURITY {DEFINER | INVOKER}  
| COMMENT 'строка'
```

Данный оператор можно использовать для переименования хранимой процедуры или функции, а также для изменения ее характеристик. В операторе ALTER PROCEDURE или ALTER FUNCTION разрешается указывать и более одной замены.

### 8.1.1.3. DROP PROCEDURE и DROP FUNCTION

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] имя_хранимой_процедуры
```

Данный оператор используется для удаления хранимой процедуры или функции, то есть указанная подпрограмма будет удалена с сервера.

Конструкция IF EXISTS является расширением MySQL. Она предотвращает появление ошибки, если процедура или функция не существует. В таких случаях появляется предупреждение, просмотреть которое можно с помощью SHOW WARNINGS.

### 8.1.1.4. SHOW CREATE PROCEDURE и SHOW CREATE FUNCTION

```
SHOW CREATE {PROCEDURE | FUNCTION} имя_хранимой_процедуры
```

Данный оператор является расширением MySQL. Он похож на оператор SHOW CREATE TABLE и возвращает точную строку, которую можно использовать для воссоздания именованной процедуры.

## 8.1.2. SHOW PROCEDURE STATUS и SHOW FUNCTION STATUS

```
SHOW {PROCEDURE | FUNCTION} STATUS [LIKE 'шаблон']
```

Данный оператор является расширением MySQL. Он возвращает характеристики подпрограмм, такие как имя, тип, имя создателя, а также даты создания и изменения. Если определенный шаблон (шаблон) не задан, отображается информация для всех хранимых процедур или функций, в зависимости от используемого оператора.

### 8.1.3. Оператор CALL

```
CALL имя_хранимой_процедуры([параметр[, ...]])
```

Оператор CALL используется для вызова процедуры, которая была ранее определена с помощью CREATE PROCEDURE.

### 8.1.4. Составной оператор BEGIN ... END

```
[метка_начала:] BEGIN  
statement(s)  
END [метка_конца]
```

Хранимые процедуры могут включать множественные операторы, благодаря составному оператору BEGIN...END.

Значения метка\_начала и метка\_конца, если оба заданы, должны быть одинаковыми.

Обратите внимание, что необязательная конструкция [NOT] ATOMIC пока не поддерживается. Это означает, что нет установленной транзакционной точки сохранения в на-

чале блока команд, и что используемая в таком контексте конструкция BEGIN на текущую транзакцию не влияет.

Для использования множественных операторов необходимо, чтобы у клиента была возможность посылать строки запросов, содержащие разделитель операторов ;. Добиться этого можно путем применения команды delimiter в командной строке клиента mysql. Замена завершающего запрос разделителя ; (например, на разделитель //) позволяет использовать ; в теле процедуры.

### 8.1.5. Оператор DECLARE

Оператор DECLARE используется для определения различных локальных для данной операции элементов, то есть для определения локальных переменных (см. раздел 8.1.6), условий и обработчиков (см. раздел 8.1.7), а также курсоров (см. раздел 8.1.8). Операторы SIGNAL и RESIGNAL в настоящее время не поддерживаются.

DECLARE может использоваться только внутри составного оператора BEGIN...END и размещается в самом его начале, перед любыми другими операторами.

### 8.1.6. Переменные в хранимых процедурах

Внутри подпрограммы можно объявлять и использовать переменные.

#### 8.1.6.1. Локальные переменные DECLARE

DECLARE имя\_переменной[,...] тип [DEFAULT значение]

Этот оператор используется для объявления локальных переменных. Контекст переменной ограничен блоком BEGIN ... END.

#### 8.1.6.2. Оператор установки переменных SET

SET имя\_переменной = выражение [, имя\_переменной = выражение] ...]

Оператор SET в хранимых процедурах представляет собой расширенную версию общего оператора SET. Запрашиваемые переменные могут быть как переменными, объявленными внутри процедуры, так и глобальными переменными сервера.

Оператор SET в хранимых процедурах вводится как часть существовавшего ранее синтаксиса SET, что разрешает использование расширенного синтаксиса: SET a=x, b=y, ..., в котором могут смешиваться разные типы переменных (локально объявленные переменные, переменные сервера, а также глобальные и сеансовые переменные сервера). Также, благодаря этому, допустимым становится применение комбинаций локальных переменных и некоторых опций, имеющих смысл только для глобальных/системных переменных; в этом случае, опции принимаются, но игнорируются.

#### 8.1.6.3. Оператор SELECT ...

SELECT имя\_столбца[,...] INTO имя\_переменной[,...] табличное\_выражение

При таком синтаксисе SELECT выбранные столбцы сохраняются непосредственно в переменных. Поэтому извлечена может быть только одна единственная строка. Данный оператор также крайне полезен при его применении в сочетании с курсорами.

SELECT id,data INTO x,y FROM test.t1 LIMIT 1;



## 8.1.7. Условия и обработчики

При некоторых условиях не исключена необходимость в особом типе обработки. Эти условия могут касаться как ошибок, так и общего управления потоком данных внутри подпрограммы.

### 8.1.7.1. Условия DECLARE

```
DECLARE имя_условия CONDITION FOR значение_условия
```

значение\_условия:

```
SQLSTATE [VALUE] значение_sqlstate  
| код_ошибки_mysql
```

Данный оператор определяет условия, при которых потребуется определенный тип обработки. Он связывает имя с указанным условием ошибки. Имя может впоследствии использоваться в операторе DECLARE HANDLER. См. раздел 8.1.7.2.

Кроме значений SQLSTATE также поддерживаются коды ошибок MySQL.

### 8.1.7.2. Обработчики DECLARE

```
DECLARE тип_обработчика HANDLER FOR значение_условия{,...}  
оператор_хранимой_процедуры
```

тип\_обработчика:

```
CONTINUE  
| EXIT  
| UNDO
```

значение\_условия:

```
SQLSTATE [VALUE] значение_sqlstate  
| имя_условия  
| SQLWARNING  
| NOT FOUND  
| SQLEXCEPTION  
| код_ошибки_mysql
```

С помощью данного оператора задаются обработчики, каждый из которых может отвечать за выполнение одного или более условий. При появлении одного из таких условий сразу выполняется указанный оператор.

Для обработчика CONTINUE выполнение текущей операции продолжается после выполнения оператора обработчика. Для обработчика EXIT выполнение текущего составного оператора BEGIN...END завершается. Тип обработчика UNDO пока не поддерживается.

- SQLWARNING — сокращенный вариант для всех кодов SQLSTATE, начинающихся с 01.
- NOT FOUND — сокращенный вариант для всех кодов SQLSTATE, начинающихся с 02.
- SQLEXCEPTION — сокращенный вариант для всех кодов SQLSTATE, не захваченных SQLWARNING или NOT FOUND.

Кроме значений SQLSTATE также поддерживаются коды ошибок MySQL.

Например:

```
mysql> CREATE TABLE test.t (s1 int,primary key (s1));  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> delimiter //
```

```

mysql> CREATE PROCEDURE handlerdemo ()
-> BEGIN
->   DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
->   SET @x = 1;
->   INSERT INTO test.t VALUES (1);
->   SET @x = 2;
->   INSERT INTO test.t VALUES (1);
->   SET @x = 3;
-> END;
-> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
mysql> CALL handlerdemo();
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x;
+-----+
| @x    |
+-----+
| 3     |
+-----+
1 row in set (0.00 sec)

```

Обратите внимание: @x равен 3, что указывает на то, что MySQL выполнил процедуру до конца. Если бы строка `DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;` представлена не была, MySQL принял бы значение пути по умолчанию (EXIT) после второй неудачной из-за ограничения PRIMARY KEY попытки выполнить INSERT, и оператор SELECT @x возвратил бы значение 2.

### 8.1.8. Курсоры

В хранимых процедурах и функциях поддерживаются простые курсоры. Синтаксис такой же, как во встроенном SQL. На данный момент курсоры являются нечувствительными, не перемещающимися и доступными только для чтения. “Нечувствительные” означает, что сервер может или не может делать копию таблицы результатов.

Например:

```

CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE a CHAR(16);
  DECLARE b,c INT;

  OPEN cur1;
  OPEN cur2;

  REPEAT
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
  
```

```
IF NOT done THEN
  IF b < c THEN
    INSERT INTO test.t3 VALUES (a,b);
  ELSE
    INSERT INTO test.t3 VALUES (a,c);
  END IF;
END IF;
UNTIL done END REPEAT;
CLOSE cur1;
CLOSE cur2;
END
```

### 8.1.8.1. Объявление курсоров

```
DECLARE имя_курсора CURSOR FOR оператор_sql
```

Во время процедуры можно определять многочисленные курсоры, но имя каждого из них должно быть уникальным.

### 8.1.8.2. Оператор открытия курсора OPEN

```
OPEN имя_курсора
```

Данный оператор открывает объявленный ранее курсор.

### 8.1.8.3. Оператор выборки курсора FETCH

```
FETCH имя_курсора INTO имя_переменной [, имя_переменной] ...
```

Данный оператор выполняет выборку следующей строки (если строка существует) с помощью указанного открытого курсора и продвигает указатель курсора.

### 8.1.8.4. Оператор закрытия курсора CLOSE

```
CLOSE имя_курсора
```

Данный оператор закрывает открытый ранее курсор.

## 8.1.9. Конструкции управления потоком данных

Конструкции IF, CASE, LOOP, WHILE, ITERATE и LEAVE реализованы полностью.

Каждая из этих конструкций может включать как единственный оператор, так и блок операторов при использовании составного оператора BEGIN...END. Конструкции можно представлять в форме вложений.

Циклы FOR на данный момент не поддерживаются.

### 8.1.9.1. Оператор IF

```
IF условие_поиска THEN оператор(ы)
  [ELSEIF условие_поиска THEN оператор(ы)]
  ...
  [ELSE оператор(ы)]
END IF
```

IF реализует базовую конструкцию условия. Если значение *условие\_поиска* является истинным, будет выполнен соответствующий SQL-оператор. Если совпадения с *условие\_поиска* не найдены, выполняться будет оператор, указанный в конструкции ELSE.

Обратите внимание на то, что существует также и функция IF(). См. раздел 5.2.

### 8.1.9.2. Оператор CASE

```
CASE значение_case  
  WHEN значение_when THEN оператор  
  [WHEN значение_when THEN оператор ...]  
  [ELSE оператор]
```

```
END CASE
```

Или:

```
CASE  
  WHEN условие_поиска THEN оператор  
  [WHEN условие_поиска THEN оператор ...]  
  [ELSE оператор]
```

```
END CASE
```

CASE реализует сложную конструкцию условия. Если значение *условие\_поиска* является истинным, будет выполнен соответствующий SQL-оператор. Если совпадения с *условие\_поиска* не найдены, выполняться будет оператор из конструкции ELSE.

#### На заметку!

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 END CASE, а не END. См. раздел 5.2.

### 8.1.9.3. Оператор LOOP

```
[метка_начала:] LOOP  
  оператор (ы)  
END LOOP [метка_конца]
```

LOOP реализует простую конструкцию цикла, допуская повторное выполнение какого-то конкретного оператора или группы операторов. Операторы в цикле повторяются до выхода из этого цикла, для чего обычно используется оператор LEAVE.

Значения *метка\_начала* и *метка\_конца*, если заданы оба, должны быть одинаковыми.

### 8.1.9.4. Оператор LEAVE

```
LEAVE метка
```

Данный оператор используется для выхода из конструкции управления потоком выполнения.

### 8.1.9.5. Оператор ITERATE

```
ITERATE метка
```

ITERATE может появиться только при использовании операторов LOOP, REPEAT и WHILE. ITERATE означает “повторить цикл снова”.

Например:

```
CREATE PROCEDURE doiterate(p1 INT)  
BEGIN  
  label1: LOOP  
    SET p1 = p1 + 1;  
    IF p1 < 10 THEN ITERATE label1; END IF;  
    LEAVE label1;  
  END LOOP label1;
```

```
SET @x = p1;
END
```

### 8.1.9.6. Оператор REPEAT

```
[метка_начала:] REPEAT
    оператор (ы)
UNTIL условие_поиска
END REPEAT [метка_конца]
```

Команды, указанные внутри оператора REPEAT, повторяются до тех пор, пока будет истинным условие *условие\_поиска*.

Значения *метка\_начала* и *метка\_конца*, если заданы оба, должны быть одинаковыми.

Например:

```
mysql> delimiter //
mysql> CREATE PROCEDURE dorepeat(p1 INT)
-> BEGIN
-> SET @x = 0;
-> REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
-> END
-> //
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> delimiter ;
```

```
mysql> CALL dorepeat(1000);
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> SELECT @x;
```

```
+-----+
| @x    |
+-----+
| 1001  |
+-----+
```

1 row in set (0.00 sec)

### 8.1.9.7. Оператор WHILE

```
[метка_начала:] WHILE условие_поиска DO
    оператор (ы)
END WHILE [метка_конца]
```

Команды, указанные внутри оператора WHILE повторяются до тех пор, пока будет истинным условие *условие\_поиска*.

Значения *метка\_начала* и *метка\_конца*, если заданы оба, должны быть одинаковыми.

Например:

```
CREATE PROCEDURE dowhile()
BEGIN
    DECLARE v1 INT DEFAULT 5;
    WHILE v1 > 0 DO
        ...
        SET v1 = v1 - 1;
    END WHILE;
END
```

# Обработка ошибок в MySQL

В данной главе перечислены ошибки, которые может возвращать MySQL.

## 9.1. Возвраты по ошибке

Ниже, в табл. 9.1, представлены коды ошибок, появление которых возможно при обращении к MySQL из любого языка на хосте.

Значения в столбцах “Имя” и “Код ошибки” (см. табл. 9.1) соответствуют определениям в исходном файле MySQL `include/mysqld_error.h`.

Значения в столбце “SQLSTATE” соответствуют определениям в исходном файле MySQL `include/sql_state.h`.

Код ошибки SQLSTATE отображается, только если используется версия MySQL 4.1 и выше. Коды SQLSTATE были добавлены для совместимости с поведением X/Open, ANSI и ODBC.

В разделе 9.2 можно найти текстовые описания для каждого кода ошибки.

Поскольку обновление происходит достаточно часто, вполне вероятно, что файлы `include/mysqld_error.h` и `include/sql_state.h` будут содержать дополнительные коды ошибок, которые здесь не представлены.

Таблица 9.1. Коды ошибок MySQL

Имя	Код ошибки	SQLSTATE
ER_HASHCHK	1000	HY000
ER_NISAMCHK	1001	HY000
ER_NO	1002	HY000
ER_YES	1003	HY000
ER_CANT_CREATE_FILE	1004	HY000
ER_CANT_CREATE_TABLE	1005	HY000
ER_CANT_CREATE_DB	1006	HY000
ER_DB_CREATE_EXISTS	1007	HY000
ER_DB_DROP_EXISTS	1008	HY000

Продолжение табл. 9.1

Имя	Код ошибки	SQLSTATE
ER_DB_DROP_DELETE	1009	HY000
ER_DB_DROP_RMDIR	1010	HY000
ER_CANT_DELETE_FILE	1011	HY000
ER_CANT_FIND_SYSTEM_REC	1012	HY000
ER_CANT_GET_STAT	1013	HY000
ER_CANT_GET_WD	1014	HY000
ER_CANT_LOCK	1015	HY000
ER_CANT_OPEN_FILE	1016	HY000
ER_FILE_NOT_FOUND	1017	HY000
ER_CANT_READ_DIR	1018	HY000
ER_CANT_SET_WD	1019	HY000
ER_CHECKREAD	1020	HY000
ER_DISK_FULL	1021	HY000
ER_DUP_KEY	1022	23000
ER_ERROR_ON_CLOSE	1023	HY000
ER_ERROR_ON_READ	1024	HY000
ER_ERROR_ON_RENAME	1025	HY000
ER_ERROR_ON_WRITE	1026	HY000
ER_FILE_USED	1027	HY000
ER_FILSORT_ABORT	1028	HY000
ER_FORM_NOT_FOUND	1029	HY000
ER_GET_ERRNO	1030	HY000
ER_ILLEGAL_HA	1031	HY000
ER_KEY_NOT_FOUND	1032	HY000
ER_NOT_FORM_FILE	1033	HY000
ER_NOT_KEYFILE	1034	HY000
ER_OLD_KEYFILE	1035	HY000
ER_OPEN_AS_READONLY	1036	HY000
ER_OUTOFMEMORY	1037	HY001
ER_OUT_OF_SORTMEMORY	1038	HY001
ER_UNEXPECTED_EOF	1039	HY000
ER_CON_COUNT_ERROR	1040	08004
ER_OUT_OF_RESOURCES	1041	08004
ER_BAD_HOST_ERROR	1042	08S01
ER_HANDSHAKE_ERROR	1043	08S01

Продолжение табл. 9.1

Имя	Код ошибки	SQLSTATE
ER_DBACCESS_DENIED_ERROR	1044	42000
ER_ACCESS_DENIED_ERROR	1045	42000
ER_NO_DB_ERROR	1046	42000
ER_UNKNOWN_COM_ERROR	1047	08S01
ER_BAD_NULL_ERROR	1048	23000
ER_BAD_DB_ERROR	1049	42000
ER_TABLE_EXISTS_ERROR	1050	42S01
ER_BAD_TABLE_ERROR	1051	42S02
ER_NON_UNIQ_ERROR	1052	23000
ER_SERVER_SHUTDOWN	1053	08S01
ER_BAD_FIELD_ERROR	1054	42S22
ER_WRONG_FIELD_WITH_GROUP	1055	42000
ER_WRONG_GROUP_FIELD	1056	42000
ER_WRONG_SUM_SELECT	1057	42000
ER_WRONG_VALUE_COUNT	1058	21S01
ER_TOO_LONG_IDENT	1059	42000
ER_DUP_FIELDNAME	1060	42S21
ER_DUP_KEYNAME	1061	42000
ER_DUP_ENTRY	1062	23000
ER_WRONG_FIELD_SPEC	1063	42000
ER_PARSE_ERROR	1064	42000
ER_EMPTY_QUERY	1065	42000
ER_NONUNIQ_TABLE	1066	42000
ER_INVALID_DEFAULT	1067	42000
ER_MULTIPLE_PRI_KEY	1068	42000
ER_TOO_MANY_KEYS	1069	42000
ER_TOO_MANY_KEY_PARTS	1070	42000
ER_TOO_LONG_KEY	1071	42000
ER_KEY_COLUMN_DOES_NOT_EXIST	1072	42000
ER_BLOB_USED_AS_KEY	1073	42000
ER_TOO_BIG_FIELDLENGTH	1074	42000
ER_WRONG_AUTO_KEY	1075	42000
ER_READY	1076	00000
ER_NORMAL_SHUTDOWN	1077	00000
ER_GOT_SIGNAL	1078	00000



Продолжение табл. 9.1

Имя	Код ошибки	SQLSTATE
ER_SHUTDOWN_COMPLETE	1079	00000
ER_FORCING_CLOSE	1080	08S01
ER_IPSOCK_ERROR	1081	08S01
ER_NO_SUCH_INDEX	1082	42S12
ER_WRONG_FIELD_TERMINATORS	1083	42000
ER_BLOBS_AND_NO_TERMINATED	1084	42000
ER_TEXTFILE_NOT_READABLE	1085	HY000
ER_FILE_EXISTS_ERROR	1086	HY000
ER_LOAD_INFO	1087	HY000
ER_ALTER_INFO	1088	HY000
ER_WRONG_SUB_KEY	1089	HY000
ER_CANT_REMOVE_ALL_FIELDS	1090	42000
ER_CANT_DROP_FIELD_OR_KEY	1091	42000
ER_INSERT_INFO	1092	HY000
ER_UPDATE_TABLE_USED	1093	HY000
ER_NO_SUCH_THREAD	1094	HY000
ER_KILL_DENIED_ERROR	1095	HY000
ER_NO_TABLES_USED	1096	HY000
ER_TOO_BIG_SET	1097	HY000
ER_NO_UNIQUE_LOGFILE	1098	HY000
ER_TABLE_NOT_LOCKED_FOR_WRITE	1099	HY000
ER_TABLE_NOT_LOCKED	1100	HY000
ER_BLOB_CANT_HAVE_DEFAULT	1101	42000
ER_WRONG_DB_NAME	1102	42000
ER_WRONG_TABLE_NAME	1103	42000
ER_TOO_BIG_SELECT	1104	42000
ER_UNKNOWN_ERROR	1105	HY000
ER_UNKNOWN_PROCEDURE	1106	42000
ER_WRONG_PARAMCOUNT_TO_PROCEDURE	1107	42000
ER_WRONG_PARAMETERS_TO_PROCEDURE	1108	HY000
ER_UNKNOWN_TABLE	1109	42S02
ER_FIELD_SPECIFIED_TWICE	1110	42000
ER_INVALID_GROUP_FUNC_USE	1111	42000
ER_UNSUPPORTED_EXTENSION	1112	42000
ER_TABLE_MUST_HAVE_COLUMNS	1113	42000

Продолжение табл. 9.1

Имя	Код ошибки	SQLSTATE
ER_RECORD_FILE_FULL	1114	HY000
ER_UNKNOWN_CHARACTER_SET	1115	42000
ER_TOO_MANY_TABLES	1116	HY000
ER_TOO_MANY_FIELDS	1117	HY000
ER_TOO_BIG_ROWSIZE	1118	42000
ER_STACK_OVERRUN	1119	HY000
ER_WRONG_OUTER_JOIN	1120	42000
ER_NULL_COLUMN_IN_INDEX	1121	42000
ER_CANT_FIND_UDF	1122	HY000
ER_CANT_INITIALIZE_UDF	1123	HY000
ER_UDF_NO_PATHS	1124	HY000
ER_UDF_EXISTS	1125	HY000
ER_CANT_OPEN_LIBRARY	1126	HY000
ER_CANT_FIND_DL_ENTRY	1127	HY000
ER_FUNCTION_NOT_DEFINED	1128	HY000
ER_HOST_IS_BLOCKED	1129	HY000
ER_HOST_NOT_PRIVILEGED	1130	HY000
ER_PASSWORD_ANONYMOUS_USER	1131	42000
ER_PASSWORD_NOT_ALLOWED	1132	42000
ER_PASSWORD_NO_MATCH	1133	42000
ER_UPDATE_INFO	1134	HY000
ER_CANT_CREATE_THREAD	1135	HY000
ER_WRONG_VALUE_COUNT_ON_ROW	1136	21S01
ER_CANT_REOPEN_TABLE	1137	HY000
ER_INVALID_USE_OF_NULL	1138	42000
ER_REGEXP_ERROR	1139	42000
ER_MIX_OF_GROUP_FUNC_AND_FIELDS	1140	42000
ER_NONEXISTING_GRANT	1141	42000
ER_TABLEACCESS_DENIED_ERROR	1142	42000
ER_COLUMNACCESS_DENIED_ERROR	1143	42000
ER_ILLEGAL_GRANT_FOR_TABLE	1144	42000
ER_GRANT_WRONG_HOST_OR_USER	1145	42000
ER_NO_SUCH_TABLE	1146	42S02
ER_NONEXISTING_TABLE_GRANT	1147	42000
ER_NOT_ALLOWED_COMMAND	1148	42000

Продолжение табл. 9.1

Имя	Код ошибки	SQLSTATE
ER_SYNTAX_ERROR	1149	42000
ER_DELAYED_CANT_CHANGE_LOCK	1150	HY000
ER_TOO_MANY_DELAYED_THREADS	1151	HY000
ER_ABORTING_CONNECTION	1152	08S01
ER_NET_PACKET_TOO_LARGE	1153	08S01
ER_NET_READ_ERROR_FROM_PIPE	1154	08S01
ER_NET_FCNTL_ERROR	1155	08S01
ER_NET_PACKETS_OUT_OF_ORDER	1156	08S01
ER_NET_UNCOMPRESS_ERROR	1157	08S01
ER_NET_READ_ERROR	1158	08S01
ER_NET_READ_INTERRUPTED	1159	08S01
ER_NET_ERROR_ON_WRITE	1160	08S01
ER_NET_WRITE_INTERRUPTED	1161	08S01
ER_TOO_LONG_STRING	1162	42000
ER_TABLE_CANT_HANDLE_BLOB	1163	42000
ER_TABLE_CANT_HANDLE_AUTO_INCREMENT	1164	42000
ER_DELAYED_INSERT_TABLE_LOCKED	1165	HY000
ER_WRONG_COLUMN_NAME	1166	42000
ER_WRONG_KEY_COLUMN	1167	42000
ER_WRONG_MRG_TABLE	1168	HY000
ER_DUP_UNIQUE	1169	23000
ER_BLOB_KEY_WITHOUT_LENGTH	1170	42000
ER_PRIMARY_CANT_HAVE_NULL	1171	42000
ER_TOO_MANY_ROWS	1172	42000
ER_REQUIRES_PRIMARY_KEY	1173	42000
ER_NO_RAID_COMPILED	1174	HY000
ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE	1175	HY000
ER_KEY_DOES_NOT_EXITS	1176	HY000
ER_CHECK_NO_SUCH_TABLE	1177	42000
ER_CHECK_NOT_IMPLEMENTED	1178	42000
ER_CANT_DO_THIS_DURING_AN_TRANSACTION	1179	25000
ER_ERROR_DURING_COMMIT	1180	HY000
ER_ERROR_DURING_ROLLBACK	1181	HY000
ER_ERROR_DURING_FLUSH_LOGS	1182	HY000
ER_ERROR_DURING_CHECKPOINT	1183	HY000

Продолжение табл. 9.1

Имя	Код ошибки	SQLSTATE
ER_NEW_ABORTING_CONNECTION	1184	08S01
ER_DUMP_NOT_IMPLEMENTED	1185	HY000
ER_FLUSH_MASTER_BINLOG_CLOSED	1186	HY000
ER_INDEX_REBUILD	1187	HY000
ER_MASTER	1188	HY000
ER_MASTER_NET_READ	1189	08S01
ER_MASTER_NET_WRITE	1190	08S01
ER_FT_MATCHING_KEY_NOT_FOUND	1191	HY000
ER_LOCK_OR_ACTIVE_TRANSACTION	1192	HY000
ER_UNKNOWN_SYSTEM_VARIABLE	1193	HY000
ER_CRASHED_ON_USAGE	1194	HY000
ER_CRASHED_ON_REPAIR	1195	HY000
ER_WARNING_NOT_COMPLETE_ROLLBACK	1196	HY000
ER_TRANS_CACHE_FULL	1197	HY000
ER_SLAVE_MUST_STOP	1198	HY000
ER_BAD_SLAVE	1200	HY000
ER_MASTER_INFO	1201	HY000
ER_SLAVE_THREAD	1202	HY000
ER_TOO_MANY_USER_CONNECTIONS	1203	42000
ER_SET_CONSTANTS_ONLY	1204	HY000
ER_LOCK_WAIT_TIMEOUT	1205	HY000
ER_LOCK_TABLE_FULL	1206	HY000
ER_READ_ONLY_TRANSACTION	1207	25000
ER_DROP_DB_WITH_READ_LOCK	1208	HY000
ER_CREATE_DB_WITH_READ_LOCK	1209	HY000
ER_WRONG_ARGUMENTS	1210	HY000
ER_NO_PERMISSION_TO_CREATE_USER	1211	42000
ER_UNION_TABLES_IN_DIFFERENT_DIR	1212	HY000
ER_LOCK_DEADLOCK	1213	40001
ER_TABLE_CANT_HANDLE_FULLTEXT	1214	HY000
ER_CANNOT_ADD_FOREIGN	1215	HY000
ER_NO_REFERENCED_ROW	1216	23000
ER_ROW_IS_REFERENCED	1217	23000
ER_CONNECT_TO_MASTER	1218	08S01
ER_QUERY_ON_MASTER	1219	HY000

Продолжение табл. 9.1

Имя	Код ошибки	SQLSTATE
ER_ERROR_WHEN_EXECUTING_COMMAND	1220	HY000
ER_WRONG_USAGE	1221	HY000
ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT	1222	21000
ER_CANT_UPDATE_WITH_READLOCK	1223	HY000
ER_MIXING_NOT_ALLOWED	1224	HY000
ER_DUP_ARGUMENT	1225	HY000
ER_USER_LIMIT_REACHED	1226	42000
ER_SPECIFIC_ACCESS_DENIED_ERROR	1227	HY000
ER_LOCAL_VARIABLE	1228	HY000
ER_GLOBAL_VARIABLE	1229	HY000
ER_NO_DEFAULT	1230	42000
ER_WRONG_VALUE_FOR_VAR	1231	42000
ER_WRONG_TYPE_FOR_VAR	1232	42000
ER_VAR_CANT_BE_READ	1233	HY000
ER_CANT_USE_OPTION_HERE	1234	42000
ER_NOT_SUPPORTED_YET	1235	42000
ER_MASTER_FATAL_ERROR_READING_BINLOG	1236	HY000
ER_SLAVE_IGNORED_TABLE	1237	HY000
ER_INCORRECT_GLOBAL_LOCAL_VAR	1238	HY000
ER_WRONG_FK_DEF	1239	42000
ER_KEY_REF_DO_NOT_MATCH_TABLE_REF	1240	HY000
ER_OPERAND_COLUMNS	1241	21000
ER_SUBQUERY_NO_1_ROW	1242	21000
ER_UNKNOWN_STMT_HANDLER	1243	HY000
ER_CORRUPT_HELP_DB	1244	HY000
ER_CYCLIC_REFERENCE	1245	HY000
ER_AUTO_CONVERT	1246	HY000
ER_ILLEGAL_REFERENCE	1247	42S22
ER_DERIVED_MUST_HAVE_ALIAS	1248	42000
ER_SELECT_REDUCE	1249	01000
ER_TABLENAME_NOT_ALLOWED_HERE	1250	42000
ER_NOT_SUPPORTED_AUTH_MODE	1251	08004
ER_SPATIAL_CANT_HAVE_NULL	1252	42000
ER_COLLATION_CHARSET_MISMATCH	1253	42000
ER_SLAVE_WAS_RUNNING	1254	HY000

Продолжение табл. 9.1

Имя	Код ошибки	SQLSTATE
ER_SLAVE_WAS_NOT_RUNNING	1255	HY000
ER_TOO_BIG_FOR_UNCOMPRESS	1256	HY000
ER_ZLIB_Z_MEM_ERROR	1257	HY000
ER_ZLIB_Z_BUF_ERROR	1258	HY000
ER_ZLIB_Z_DATA_ERROR	1259	HY000
ER_CUT_VALUE_GROUP_CONCAT	1260	HY000
ER_WARN_TOO_FEW_RECORDS	1261	01000
ER_WARN_TOO_MANY_RECORDS	1262	01000
ER_WARN_NULL_TO_NOTNULL	1263	01000
ER_WARN_DATA_OUT_OF_RANGE	1264	01000
ER_WARN_DATA_TRUNCATED	1265	01000
ER_WARN_USING_OTHER_HANDLER	1266	HY000
ER_CANT_AGGREGATE_2COLLATIONS	1267	HY000
ER_DROP_USER	1268	HY000
ER_REVOKE_GRANTS	1269	HY000
ER_CANT_AGGREGATE_3COLLATIONS	1270	HY000
ER_CANT_AGGREGATE_NCOLLATIONS	1271	HY000
ER_VARIABLE_IS_NOT_STRUCT	1272	HY000
ER_UNKNOWN_COLLATION	1273	HY000
ER_SLAVE_IGNORED_SSL_PARAMS	1274	HY000
ER_SERVER_IS_IN_SECURE_AUTH_MODE	1275	HY000
ER_WARN_FIELD_RESOLVED	1276	HY000
ER_BAD_SLAVE_UNTIL_COND	1277	HY000
ER_MISSING_SKIP_SLAVE	1278	HY000
ER_UNTIL_COND_IGNORED	1279	HY000
ER_WRONG_NAME_FOR_INDEX	1280	42000
ER_WRONG_NAME_FOR_CATALOG	1281	42000
ER_WARN_OC_RESIZE	1282	HY000
ER_BAD_FT_COLUMN	1283	HY000
ER_UNKNOWN_KEY_CACHE	1284	HY000
ER_WARN_HOSTNAME_WONT_WORK	1285	HY000
ER_UNKNOWN_STORAGE_ENGINE	1286	42000
ER_WARN_DEPRECATED_SYNTAX	1287	HY000
ER_NON_UPDATABLE_TABLE	1288	HY000
ER_FEATURE_DISABLED	1289	HY000

Продолжение табл. 9.1

Имя	Код ошибки	SQLSTATE
ER_OPTION_PREVENTS_STATEMENT	1290	HY000
ER_DUPLICATED_VALUE_IN_TYPE	1291	HY000
ER_TRUNCATED_WRONG_VALUE	1292	HY000
ER_TOO_MUCH_AUTO_TIMESTAMP_COLS	1293	HY000
ER_INVALID_ON_UPDATE	1294	HY000
ER_UNSUPPORTED_PS	1295	HY000
ER_SP_NO_RECURSIVE_CREATE	1296	2F003
ER_SP_ALREADY_EXISTS	1297	42000
ER_SP_DOES_NOT_EXIST	1298	42000
ER_SP_DROP_FAILED	1299	HY000
ER_SP_STORE_FAILED	1300	HY000
ER_SP_LILABEL_MISMATCH	1301	42000
ER_SP_LABEL_REDEFINE	1302	42000
ER_SP_LABEL_MISMATCH	1303	42000
ER_SP_UNINIT_VAR	1304	01000
ER_SP_BADSELECT	1305	0A000
ER_SP_BADRETURN	1306	42000
ER_SP_BADSTATEMENT	1307	0A000
ER_UPDATE_LOG_DEPRECATED_IGNORED	1308	42000
ER_UPDATE_LOG_DEPRECATED_TRANSLATED	1309	42000
ER_QUERY_INTERRUPTED	1310	70100
ER_SP_WRONG_NO_OF_ARGS	1311	42000
ER_SP_COND_MISMATCH	1312	42000
ER_SP_NORETURN	1313	42000
ER_SP_NORETURNEND	1314	2F005
ER_SP_BAD_CURSOR_QUERY	1315	42000
ER_SP_BAD_CURSOR_SELECT	1316	42000
ER_SP_CURSOR_MISMATCH	1317	42000
ER_SP_CURSOR_ALREADY_OPEN	1318	24000
ER_SP_CURSOR_NOT_OPEN	1319	24000
ER_SP_UNDECLARED_VAR	1320	42000
ER_SP_WRONG_NO_OF_FETCH_ARGS	1321	HY000
ER_SP_FETCH_NO_DATA	1322	02000
ER_SP_DUP_PARAM	1323	42000
ER_SP_DUP_VAR	1324	42000

Окончание табл. 9.1

Имя	Код ошибки	SQLSTATE
ER_SP_DUP_COND	1325	2000
ER_SP_DUP_CURS	1326	42000
ER_SP_CANT_ALTER	1327	HY000
ER_SP_SUBSELECT_NYI	1328	0A000
ER_SP_NO_USE	1329	42000
ER_SP_VARCOND_AFTER_CURSHNDLR	1330	42000
ER_SP_CURSOR_AFTER_HANDLER	1331	42000
ER_SP_CASE_NOT_FOUND	1332	20000
ER_FPARSER_TOO_BIG_FILE	1333	HY000
ER_FPARSER_BAD_HEADER	1334	HY000
ER_FPARSER_EOF_IN_COMMENT	1335	HY000
ER_FPARSER_ERROR_IN_PARAMETER	1336	HY000
ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER	1337	HY000

## 9.2. Сообщения об ошибках

Ниже, в табл. 9.2, представлены сообщения об ошибках, которые могут возникнуть при обращении к MySQL из любого языка на хосте. %d и %s представляют, соответственно, числа и строки, замещаемые в сообщении соответствующими значениями.

Таблица 9.2. Сообщения об ошибках MySQL

Код ошибки	Сообщение об ошибке
1000	hashchk
1001	isamchk
1002	NO/Нет
1003	YES/Да
1004	Can't create file '%s' (errno: %d) Не удастся создать файл '%s' (errno: %d)
1005	Can't create table '%s' (errno: %d) Не удастся создать таблицу '%s' (errno: %d)
1006	Can't create database '%s' (errno: %d) Не удастся создать базу данных '%s' (errno: %d)
1007	Can't create database '%s'; database exists Не удастся создать базу данных '%s'; база данных уже существует
1008	Can't drop database '%s'; database doesn't exist Не удастся удалить базу данных; база данных не существует



Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1009	Error dropping database (can't delete '%s', errno: %d) Ошибка при удалении базы данных (не удается удалить '%s', errno: %d)
1010	Error dropping database (can't rmdir '%s', errno: %d) Ошибка при удалении базы данных (не удается выполнить rmdir '%s', errno: %d)
1011	Error on delete of '%s' (errno: %d) Ошибка при удалении '%s' (errno: %d)
1012	Can't read record in system table Не удастся прочитать запись в системной таблице
1013	Can't get status of '%s' (errno: %d) Не удастся получить состояние '%s' (errno: %d)
1014	Can't get working directory (errno: %d) Не удастся получить рабочий каталог (errno: %d)
1015	Can't lock file (errno: %d) Не удастся заблокировать файл (errno: %d)
1016	Can't open file: '%s' (errno: %d) Не удастся открыть файл '%s' (errno: %d)
1017	Can't find file: '%s' (errno: %d) Не удастся найти файл '%s' (errno: %d)
1018	Can't read dir of '%s' (errno: %d) Не удастся прочитать каталог '%s' (errno: %d)
1019	Can't change dir to '%s' (errno: %d) Не удастся заменить каталог на '%s'
1020	Record has changed since last read in table '%s' Запись изменилась с момента последнего считывания таблицы '%s'
1021	Disk full (%s); waiting for someone to free some space Диск переполнен; необходимо освободить некоторое пространство
1022	Can't write; duplicate key in table '%s' Не удастся выполнить запись; дублированный ключ в таблице '%s'
1023	Error on close of '%s' (errno: %d) Ошибка при закрытии '%s' (errno: %d)
1024	Error reading file '%s' (errno: %d) Ошибка при чтении файла '%s' (errno: %d)
1025	Error on rename of '%s' to '%s' (errno: %d) Ошибка при переименовании '%s' в '%s' (errno: %d)
1026	Error writing file '%s' (errno: %d) Ошибка при записи файла '%s' (errno: %d)
1027	'%s' is locked against change '%s' заблокирован от изменений

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1028	Sort aborted Сортировка прервана
1029	View '%s' doesn't exist for '%s' Представление '%s' не существует для '%s'
1030	Got error %d from storage engine Получена ошибка %d от механизма хранения
1031	Table storage engine for '%s' doesn't have this option Механизм хранения таблиц для '%s' эту опцию не поддерживает
1032	Can't find record in '%s' Не удастся найти запись в '%s'
1033	Incorrect information in file: '%s' Некорректная информация в файле '%s'
1034	Incorrect key file for table '%s'; try to repair it Некорректный файл ключей для таблицы '%s'
1035	Old key file for table '%s'; repair it! Старый файл ключей для таблицы '%s'; восстановите его!
1036	Table '%s' is read only Таблица '%s' является таблицей только для чтения
1037	Out of memory; restart server and try again (needed %d bytes) Недостаточно памяти; перезапустите сервер и повторите попытку (требуется %d байт)
1038	Out of sort memory; increase server sort buffer size Недостаточно памяти для сортировки; увеличьте размер буфера сортировки на сервере
1039	Unexpected EOF found when reading file '%s' (errno: %d) Обнаружен неожиданный конец файла при чтении файла '%s' (errno: %d)
1040	Too many connections Слишком много соединений
1041	Out of memory; check if mysqld or some other process uses all available memory; if not, you may have to use 'ulimit' to allow mysqld to use more memory or you can add more swap space Недостаточно памяти; проверьте, использует ли mysqld или какой-то другой процесс всю доступную память; если нет, возможно, понадобится применить 'ulimit', чтобы разрешить mysqld использовать больше памяти, или же потребуется добавить дополнительное пространство для свопинга
1042	Can't get hostname for your address Не удастся получить имя хоста для указанного адреса
1043	Bad handshake Ошибочное подтверждение

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1044	Access denied for user '%s'@'%s' to database '%s' Пользователю '%s'@'%s' в доступе к базе данных '%s' отказано
1045	Access denied for user '%s'@'%s' (using password: %s) Пользователю '%s'@'%s' в доступе отказано (используемый пароль: %s)
1046	No database selected Не выбрана база данных
1047	Unknown command Неизвестная команда
1048	Column '%s' cannot be null Столбцы '%s' не могут быть NULL
1049	Unknown database '%s' Неизвестная база данных '%s'
1050	Table '%s' already exists Таблица '%s' уже существует
1051	Unknown table '%s' Неизвестная таблица '%s'
1052	Column '%s' in %s is ambiguous Столбец '%s' в таблице %s неоднозначен
1053	Server shutdown in progress Сервер находится в процессе завершения работы
1054	Unknown column '%s' in '%s' Неизвестный столбец '%s' в таблице '%s'
1055	'%s' isn't in GROUP BY '%s' не указано в GROUP BY
1056	Can't group on '%s' Не удастся выполнить группировку на '%s'
1057	Statement has sum functions and columns in same statement Оператор содержит суммарные функции и столбцы в одном и том же выражении
1058	Column count doesn't match value count Количество столбцов не совпадает с количеством значений
1059	Identifier name '%s' is too long Имя идентификатора '%s' слишком длинное
1060	Duplicate column name '%s' Дублированное имя столбца '%s'
1061	Duplicate key name '%s' Дублированное имя ключа '%s'
1062	Duplicate entry '%s' for key %d Дублированная запись '%s' для ключа %d

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1063	Incorrect column specifier for column '%s' Некорректный спецификатор для столбца '%s'
1064	%s near '%s' at line %d %s рядом с '%s' в строке %d
1065	Query was empty Запрос был пустой
1066	Not unique table/alias: '%s' Неуникальное имя/псевдоним таблицы: '%s'
1067	Invalid default value for '%s' Недействительное значение по умолчанию для '%s'
1068	Multiple primary key defined Определено несколько первичных ключей
1069	Too many keys specified; max %d keys allowed Указано слишком много ключей; максимально допустимое число ключей равно %d
1070	Too many key parts specified; max %d parts allowed Указано слишком много ключевых частей; максимально допускается %d частей
1071	Specified key was too long; max key length is %d bytes Указанный ключ был слишком длинным, максимальная длина ключа равняется %d байтам
1072	Key column '%s' doesn't exist in table Столбец ключей '%s' в таблице не существует
1073	BLOB column '%s' can't be used in key specification with the used table type BLOB-столбец '%s' не может быть указан в спецификации ключа с используемым типом таблицы
1074	Column length too big for column '%s' (max = %d); use BLOB instead Размер для столбца '%s' слишком большой (максимально допустимый размер равен %d); вместо этого используйте BLOB
1075	Incorrect table definition; there can be only one auto column and it must be defined as a key Некорректное определение таблицы; возможен только один столбец auto_increment и определять его следует как ключ
1076	%s: ready for connections. Version: '%s' socket: '%s' port: %d %s: готов для соединений. Версия: '%s', сокет: '%s', порт: %d
1077	%s: Normal shutdown %s: нормальное завершение работы
1078	%s: Got signal %d Aborting! %s: получен сигнал %d - принудительное завершение работы!

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1079	%s: Shutdown complete %s: работа завершена
1080	%s: Forcing close of thread %ld user: '%s' %s: принудительное закрытие потока %ld пользователь: '%s'.
1081	Can't create IP socket Не удастся создать сокет IP
1082	Table '%s' has no index like the one used in CREATE INDEX; recreate the table Таблица '%s' не содержит индекса, подобного использованному в CREATE INDEX; создайте таблицу заново
1083	Field separator argument is not what is expected; check the manual Аргумент разделителя полей не соответствует ожидаемому; сверьтесь с руководством
1084	You can't use fixed rowlength with BLOBs; please use 'fields terminated by' Нельзя использовать строки фиксированной длины в таблицах BLOB; используйте 'поля, оканчивающиеся на'
1085	The file '%s' must be in the database directory or be readable by all Файл '%s' должен находиться в каталоге базы данных или быть доступным для чтения всем
1086	File '%s' already exists Файл '%s' уже существует
1087	Records: %ld Deleted: %ld Skipped: %ld Warnings: %ld Записи: %ld Удаленные: %ld Пропущенные: %ld Предупреждения: %ld
1088	Records: %ld Duplicates: %ld Записи: %ld Дубликаты: %ld
1089	Incorrect sub part key; the used key part isn't a string, the used length is longer than the key part, or the storage engine doesn't support unique sub keys Некорректная часть подключа; используемая часть ключа не является строкой, примененная длина больше части ключа или механизм хранения не поддерживает уникальные подключи
1090	You can't delete all columns with ALTER TABLE; use DROP TABLE instead С помощью ALTER TABLE удалить все столбцы нельзя; вместо ALTER TABLE используйте DROP TABLE
1091	Can't DROP '%s'; check that column/key exists Не удастся удалить '%s' с помощью DROP; проверьте существование столбцов/ключей
1092	Records: %ld Duplicates: %ld Warnings: %ld Записи: %ld Дубликаты: %ld Предупреждения: %ld

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1093	You can't specify target table '%s' for update in FROM clause Нельзя указать целевую таблицу '%s' для обновления в конструкции FROM
1094	Unknown thread id: %lu Неизвестный идентификатор потока: %lu
1095	You are not owner of thread %lu Вы не являетесь владельцем данного потока
1096	No tables used Никакие таблицы не используются
1097	Too many strings for column %s and SET Слишком много строк для столбца %s и SET
1098	Can't generate a unique log-filename %s (1-999) Не удастся создать уникальное имя системного файла %s (1-999)
1099	Table '%s' was locked with a READ lock and can't be updated Таблица '%s' была заблокирована с помощью блокировки READ и обновить ее нельзя
1100	Table '%s' was not locked with LOCK TABLES Таблица '%s' не была заблокирована с помощью LOCK TABLES
1101	BLOB/TEXT column '%s' can't have a default value Столбец BLOB/TEXT с именем '%s' не может содержать значения по умолчанию
1102	Incorrect database name '%s' Некорректное имя базы данных '%s'
1103	Incorrect table name '%s' Некорректное имя таблицы '%s'
1104	The SELECT would examine more than MAX_JOIN_SIZE rows; check your WHERE and use SET SQL_BIG_SELECTS=1 or SET SQL_MAX_JOIN_SIZE=# if the SELECT is okay SELECT обработает больше, чем указано в MAX_JOIN_SIZE строк; проверьте условие WHERE и используйте SET SQL_BIG_SELECTS=1 или SET SQL_MAX_JOIN_SIZE=#, если SELECT задан корректно
1105	Unknown error Неизвестная ошибка
1106	Unknown procedure '%s' Неизвестная процедура '%s'
1107	Incorrect parameter count to procedure '%s' Некорректное число параметров в процедуре '%s'
1108	Incorrect parameters to procedure '%s' Некорректные параметры в процедуре '%s'
1109	Unknown table '%s' in %s Неизвестная таблица '%s' в %s

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1110	Column '%s' specified twice Столбец '%s' указан дважды
1111	Invalid use of group function Недопустимое применение групповой функции
1112	Table '%s' uses an extension that doesn't exist in this MySQL version Таблица '%s' использует расширение, которое не существует в этой версии MySQL
1113	A table must have at least 1 column Таблица должна содержать, по крайней мере, один столбец
1114	The table '%s' is full Таблица '%s' заполнена
1115	Unknown character set: '%s' Неизвестный набор символов '%s'
1116	Too many tables; MySQL can only use %d tables in a join Слишком много таблиц; MySQL может использовать не больше %d таблиц за одно соединение
1117	Too many columns Слишком много столбцов
1118	Row size too large. The maximum row size for the used table type, not counting BLOBs, is %ld. You have to change some columns to TEXT or BLOBs Размер строк слишком большой. Максимальный размер строк для используемого типа таблиц, без учета BLOB, равен %ld. Некоторые столбцы необходимо заменить на TEXT или BLOB
1119	Thread stack overrun: Used: %ld of a %ld stack .Use 'mysqld-O thread_stack=#' to specify a bigger stack if needed Переполнение стека потока: Использовано %ld из %ld процентов стека. Используйте mysqld-O thread_stack='#', чтобы при необходимости указать больший размер стека
1120	Cross dependency found in OUTER JOIN; examine your ON conditions В OUTER JOIN обнаружена перекрестная зависимость; проверьте условия ON
1121	Column '%s' is used with UNIQUE or INDEX but is not defined as NOT NULL Столбец '%s' используется с UNIQUE или INDEX и не определен как NOT NULL
1122	Can't load function '%s' Не удастся загрузить функцию '%s'
1123	Can't initialize function '%s'; %s Не удастся запустить функцию '%s'; %s
1124	No paths allowed for shared library Не разрешены никакие пути к совместно используемой библиотеке
1125	Function '%s' already exists Функция '%s' уже существует

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1126	Can't open shared library '%s' (errno: %d %s) Не удастся открыть совместно используемую библиотеку '%s' (errno: %d %s)
1127	Can't find function '%s' in library' Не удастся найти функцию '%s' в библиотеке
1128	Function '%s' is not defined Функция '%s' не определена
1129	Host '%s' is blocked because of many connection errors;unblock with 'mysqladmin flush-hosts' Хост '%s' заблокирован по причине многочисленных ошибок соединения; разблокируйте с помощью 'mysqladmin flush-hosts'
1130	Host '%s' is not allowed to connect to this MySQL server Хосту '%s' запрещено подключение к данному серверу MySQL
1131	You are using MySQL as an anonymous user and anonymous users are not allowed to change passwords Вы подключились к MySQL как анонимный пользователь, а анонимным пользователям запрещено изменять пароли
1132	You must have privileges to update tables in the mysql database to be able to change passwords for others Недостаточно привилегий для обновления таблиц в базе данных mysql и изменения паролей других пользователей
1133	Can't find any matching row in the user table Не удастся найти совпадающую строку в таблице user
1134	Rows matched: %ld Changed: %ld Warnings: %ld Совпавшие строки: %ld Измененные: %ld Предупреждения: %ld
1135	Can't create a new thread (errno %d); if you are not out of available memory, you can consult the manual for a possible OS-dependent bug Не удастся создать новый поток (errno %d); если причина не в недостаточном количестве доступной памяти, сверьтесь с руководством: не исключена ошибка, связанная с ОС
1136	Column count doesn't match value count at row %ld Количество столбцов не соответствует количеству значений в строке %ld
1137	Can't reopen table: '%s' Не удастся повторно открыть таблицу: '%s'
1138	Invalid use of NULL value Недопустимое использование NULL
1139	Got error '%s' from regexp Получена ошибка с regexp



Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1140	Mixing of GROUP columns (MIN(),MAX(),COUNT(),...) with no GROUP columns is illegal if there is no GROUP BY clause Смешивание столбцов GROUP (MIN(),MAX(),COUNT(),...) со столбцами не-GROUP недопустимо при отсутствии конструкции GROUP BY
1141	There is no such grant defined for user '%s' on host '%s' Такая привилегия не была определена для пользователя '%s' на хосте '%s'
1142	%s command denied to user '%s'@'%s' for table '%s' В использовании команды %s для таблицы '%s' пользователю '%s'@'%s' отказано
1143	%s command denied to user '%s'@'%s' for column '%s' in table '%s' Пользователю '%s'@'%s' отказано в команде %s для столбца '%s' в таблице '%s'
1144	Illegal GRANT/REVOKE command; please consult the manual to see which privileges can be used Недопустимая команда GRANT/REVOKE; проконсультируйтесь с руководством на предмет используемых привилегий
1145	The host or user argument to GRANT is too long Аргумент хоста или пользователя для GRANT слишком длинный
1146	Table '%s %s' doesn't exist Таблица '%s %s' не существует
1147	There is no such grant defined for user '%s' on host '%s' on table '%s' Такая привилегия не была определена для пользователя '%s' на хосте '%s' для таблицы '%s'
1148	The used command is not allowed with this MySQL version Используемая команда не разрешена в этой версии MySQL
1149	You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use Ошибка в синтаксисе SQL; проверьте в руководстве по используемой вами версии MySQL-сервера, какой синтаксис будет корректным
1150	Delayed insert thread couldn't get requested lock for table %s Потоку задержанных вставок не удалось получить запрашиваемую блокировку для таблицы %s
1151	Too many delayed threads in use Используется слишком много задержанных потоков
1152	Aborted connection %ld to db: '%s' user: '%s' (%s) Прервано подключение %ld к базе данных: '%s' пользователь: '%s'
1153	Got a packet bigger than 'max_allowed_packet' bytes Полученный пакет больше 'max_allowed_packet' байт
1154	Got a read error from the connection pipe Получена ошибка при чтении с канала соединения

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1155	Got an error from fcntl() Получена ошибка от fcntl()
1156	Got packets out of order Получены поврежденные пакеты
1157	Couldn't uncompress communication packet Не удается извлечь коммуникационный пакет
1158	Got an error reading communication packets Получена ошибка при чтении коммуникационных пакетов
1159	Got timeout reading communication packets Получен тайм-аут при чтении коммуникационных пакетов
1160	Got an error writing communication packets Получена ошибка при записи коммуникационных пакетов
1161	Got timeout writing communication packets Получен тайм-аут при записи коммуникационных пакетов
1162	Result string is longer than 'max_allowed_packet' bytes Строка результата длиннее 'max_allowed_packet' байт
1163	The used table type doesn't support BLOB/TEXT columns Используемый тип таблицы не поддерживает столбцы BLOB/TEXT
1164	The used table type doesn't support AUTO_INCREMENT columns Используемый тип таблицы не поддерживает столбцы AUTO_INCREMENT
1165	INSERT DELAYED can't be used with table '%s' because it is locked with LOCK TABLES Команда INSERT DELAYED не может быть использована с таблицей '%s', поскольку она заблокирована через LOCK TABLES
1166	Incorrect column name '%s' Некорректное имя столбца '%s'
1167	The used storage engine can't index column '%s' Используемый механизм хранения не может проиндексировать столбец '%s'
1168	All tables in the MERGE table are not identically defined Все таблицы в таблице MERGE не являются одинаково определенными
1169	Can't write, because of unique constraint, to table '%s' Не удастся выполнить запись в таблицу '%s' по причине ограничения уникальности
1170	BLOB/TEXT column '%s' used in key specification without a key length BLOB/TEXT-столбец '%s' использовался в спецификации ключа без указания длины ключа
1171	All parts of a PRIMARY KEY must be NOT NULL; if you need NULL in a key, use UNIQUE instead Все части PRIMARY KEY должны быть NOT NULL; при необходимости в наличии значения NULL в ключе используйте UNIQUE

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1172	Result consisted of more than one row Результат состоял из более чем одной строки
1173	This table type requires a primary key Для данной таблицы требуется первичный ключ
1174	This version of MySQL is not compiled with RAID support Данная версия MySQL не компилировалась с поддержкой RAID
1175	You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column Вы используете безопасный режим обновления и попытались обновить таблицу без условия WHERE, использующего столбец KEY
1176	Key '%s' doesn't exist in table '%s' Ключ '%s' не существует в таблице '%s'
1177	Can't open table Не удается открыть таблицу
1178	The storage engine for the table doesn't support %s Механизм хранения для таблицы не поддерживает %s
1179	You are not allowed to execute this command in a transaction Вам запрещено выполнение данной команды в транзакции
1180	Got error %d during COMMIT Получена ошибка %d во время выполнения COMMIT
1181	Got error %d during ROLLBACK Получена ошибка %d во время выполнения ROLLBACK
1182	Got error %d during FLUSH_LOGS Получена ошибка %d во время выполнения FLUSH_LOGS
1183	Got error %d during CHECKPOINT Получена ошибка %d во время выполнения CHECKPOINT
1184	Aborted connection %ld to db: '%s' user: '%s' host: '%s' (%s) Прервано подключение %ld к базе данных '%s' пользователь: '%s' хост '%s' (%s)
1185	The storage engine for the table does not support binary table dump Механизм хранения таблицы не поддерживает бинарный дамп таблиц
1186	Binlog closed, cannot RESET MASTER Бинарный журнал закрыт, не удается выполнить RESET MASTER
1187	Failed rebuilding the index of dumped table '%s' Неудачная попытка восстановления индекса из дампа таблицы '%s'
1188	Error from master: '%s' Ошибка главного сервера: '%s'
1189	Net error reading from master Сетевая ошибка при чтении с главного сервера

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1190	Net error writing to master Сетевая ошибка при выполнении записи на главный сервер
1191	Can't find FULLTEXT index matching the column list Не удастся найти индекс FULLTEXT, соответствующий списку столбца
1192	Can't execute the given command because you have active locked tables or an active transaction Не удастся выполнить данную команду по причине наличия активных заблокированных таблиц или активной транзакции
1193	Unknown system variable '%s' Неизвестная системная переменная '%s'
1194	Table '%s' is marked as crashed and should be repaired Таблица '%s' помечена как поврежденная и должна быть исправлена
1195	Table '%s' is marked as crashed and last (automatic?) repair failed/Таблица '%s' помечена как поврежденная и последняя попытка (автоматического?) исправления была неудачной
1196	Some non-transactional changed tables couldn't be rolled back Некоторые нетранзакционные измененные таблицы возврату в первоначальное состояние не подлежат
1197	Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage; increase this mysqld variable and try again Для транзакции с множественными операторами потребовалось больше, чем указано в 'max_binlog_cache_size' байтов в кэше; увеличьте значение данной переменной и повторите попытку
1198	This operation cannot be performed with a running slave; run STOP SLAVE first/Данная операция не может быть выполнена при работающем ведомом сервере, сначала выполните STOP SLAVE
1199	This operation requires a running slave; configure slave and do START SLAVE Для данной операции требуется запустить подчиненный сервер; сконфигурируйте подчиненный сервер и выполните START SLAVE
1200	The server is not configured as slave; fix in config file or with CHANGE MASTER TO Сервер не сконфигурирован как подчиненный; устраните ошибку в файле конфигурации или воспользуйтесь CHANGE MASTER TO
1201	Could not initialize master info structure; more error messages can be found in the MySQL error log Не удалось инициализировать информационную структуру на главном сервере; дополнительные сообщения об ошибках можно найти в журнале ошибок MySQL

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1202	Could not create slave thread; check system resources Не удалось создать поток для подчиненного сервера; проверьте ресурсы системы
1203	User %s has already more than 'max_user_connections' active connections Количество активных соединений пользователя %s уже превышает 'max_user_connections'
1204	You may only use constant expressions with SET С SET можно использовать только константные выражения
1205	Lock wait timeout exceeded; try restarting transaction Превышено время ожидания блокировки; попробуйте перезапустить транзакцию
1206	The total number of locks exceeds the lock table size Общее количество блокировок превышает размер таблицы блокировки
1207	Update locks cannot be acquired during a READ UNCOMMITTED transaction Получение блокировки обновления невозможно во время выполнения транзакции READ UNCOMMITTED
1208	DROP DATABASE not allowed while thread is holding global read lock Выполнение DROP DATABASE невозможно, пока в потоке используется глобальная блокировка чтения
1209	CREATE DATABASE not allowed while thread is holding global read lock Выполнение CREATE DATABASE невозможно, пока в потоке используется глобальная блокировка чтения
1210	Incorrect arguments to %s Некорректные аргументы для %s
1211	'%s'@'%s' is not allowed to create new users Пользователю '%s'@'%s' запрещено создавать новых пользователей
1212	Incorrect table definition; all MERGE tables must be in the same database Таблица определена некорректно; все таблицы MERGE должны находиться в одной и той же базе данных
1213	Deadlock found when trying to get lock; try restarting transaction Обнаружена ошибка взаимной блокировки при попытке получить блокировку; попробуйте перезапустить транзакцию
1214	The used table type doesn't support FULLTEXT indexes Используемый тип таблицы не поддерживает индексы FULLTEXT
1215	Cannot add foreign key constraint Не удастся добавить ограничение внешнего ключа
1216	Cannot add or update a child row: a foreign key constraint fails Не удастся добавить или обновить дочернюю строку: ограничение внешнего ключа дает сбой

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1217	Cannot delete or update a parent row: a foreign key constraint fails Не удастся удалить или обновить родительскую строку: ограничение внешнего ключа дает сбой
1218	Error connecting to master: %s Ошибка при подключении к главному серверу: %s
1219	Error running query on master: %s Ошибка при выполнении запроса на главном сервере: %s
1220	Error when executing command %s: %s Ошибка при выполнении команды %s: %s
1221	Incorrect usage of %s and %s Некорректное использование %s и %s
1222	The used SELECT statements have a different number of columns Используемые операторы SELECT имеют разное количество столбцов
1223	Can't execute the query because you have a conflicting read lock Не удастся выполнить запрос из-за конфликта блокировок чтения
1224	Mixing of transactional and non-transactional tables is disabled Опция смешивания транзакционных и нетранзакционных таблиц отключена
1225	Option '%s' used twice in statement Опция '%s' используется в операторе дважды
1226	User '%s' has exceeded the '%s' resource (current value: %ld) Пользователь '%s' превысил лимит по использованию ресурса '%s'
1227	Access denied; you need the %s privilege for this operation В доступе отказано; для выполнения данной операции требуется привилегия %s
1228	Variable '%s' is a SESSION variable and can't be used with SET GLOBAL Переменная '%s' является сеансовой переменной и не может использоваться в SET GLOBAL
1229	Variable '%s' is a GLOBAL variable and should be set with SET GLOBAL Переменная '%s' является глобальной переменной и должна устанавливаться с помощью SET GLOBAL
1230	Variable '%s' doesn't have a default value Переменная '%s' не имеет значения по умолчанию
1231	Variable '%s' can't be set to the value of '%s' Переменной '%s' не может быть присвоено значение '%s'
1232	Incorrect argument type to variable '%s' Некорректный тип аргумента в переменной '%s'
1233	Variable '%s' can only be set, not read Переменную '%s' можно только устанавливать, но не считывать
1234	Incorrect usage/placement of '%s' Некорректное использование/размещение '%s'

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1235	This version of MySQL doesn't yet support '%s' Данная версия MySQL еще не поддерживает '%s'
1236	Got fatal error %d: '%s' from master when reading data from binary log Получена неисправимая ошибка %d: '%s' с ведущего сервера при чтении данных из бинарного журнала
1237	Slave SQL thread ignored the query because of replicate-*-table rules Запрос был проигнорирован потоком SQL подчиненного сервера из-за правил replicate-*-table
1238	Variable '%s' is a %s variable Переменная '%s' является переменной %s
1239	Incorrect foreign key definition for '%s': %s Некорректное определение внешнего ключа для '%s': %s
1240	Key reference and table reference don't match Ссылка на ключ и ссылка на таблицу не совпадают
1241	Operand should contain %d column(s) Операнд должен содержать %d столбцов
1242	Subquery returns more than 1 row Подзапрос возвращает более чем 1 строку
1243	Unknown prepared statement handler (%s) given to %s Для %s указан неизвестный обработчик операторов (%s)
1244	Help database is corrupt or does not exist База данных справочника повреждена или не существует
1245	Cyclic reference on subqueries Циклическая ссылка в подзапросах
1246	Converting column '%s' from %s to %s Преобразование столбца '%s' из %s в %s
1247	Reference '%s' not supported (%s) Ссылка '%s' не поддерживается (%s)
1248	Every derived table must have its own alias У каждой производной таблицы должен быть свой собственный псевдоним
1249	Select %u was reduced during optimization Во время оптимизации %u было сокращено
1250	Table '%s' from one of the SELECTs cannot be used in %s Таблица '%s', указанная в одном из операторов SELECT не может быть использована в %s
1251	Client does not support authentication protocol requested by server; consider upgrading MySQL client Клиент не поддерживает протокол аутентификации, запрашиваемый сервером; требуется обновление версии клиента MySQL

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1252	All parts of a SPATIAL index must be NOT NULL Все части индекса SPATIAL должны быть NOT NULL
1253	COLLATION '%s' is not valid for CHARACTER SET '%s' Операция COLLATION '%s' недопустима для CHARACTER SET '%s'
1254	Slave is already running Подчиненный сервер уже выполняется
1255	Slave has already been stopped Подчиненный сервер уже остановлен
1256	Uncompressed data size too large; the maximum size is %d(probably, length of uncompressed data was corrupted) Размер несжатых данных слишком велик; максимально допустимый размер равен %d (не исключено повреждение размера несжатых данных)
1257	ZLIB: Not enough memory ZLIB: Недостаточно памяти
1258	ZLIB: Not enough room in the output buffer (probably, length of uncompressed data was corrupted) ZLIB: Недостаточно места в выходном буфере (не исключено повреждение размера несжатых данных)
1259	ZLIB: Input data corrupted ZLIB: Повреждение входных данных
1260	%d line(s) were cut by GROUP_CONCAT() %d строк усечено GROUP_CONCAT()
1261	Row %ld doesn't contain data for all columns Строка %ld не содержит данных для всех столбцов
1262	Row %ld was truncated; it contained more data than there were input columns Строка %ld была усечена; она содержала данных больше, чем было входных столбцов
1263	Data truncated; NULL supplied to NOT NULL column '%s' at row %ld Данные усечены; значение NULL занесено в столбец NOT NULL в строке %ld
1264	Data truncated; out of range for column '%s' at row %ld Данные усечены; недостаточный диапазон для столбца '%s' в строке %ld
1265	Data truncated for column '%s' at row %ld Данные усечены для столбца '%s' в строке %ld
1266	Using storage engine %s for table '%s' Использование механизма хранения %s для таблицы '%s'
1267	Illegal mix of collations (%s,%s) and (%s,%s) for operation '%s' Недопустимое сочетание сопоставлений (%s,%s) и (%s,%s) для операции '%s'
1268	Can't drop one or more of the requested users Не удастся удалить одного или более из запрашиваемых пользователей



Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1269	Can't revoke all privileges, grant for one or more of the requested users Не удастся отменить все привилегии, предоставленные одному или более из запрашиваемых пользователей
1270	Illegal mix of collations (%s,%s), (%s,%s), (%s,%s) for operation '%s' Недопустимое сочетание сопоставлений (%s,%s), (%s,%s), (%s,%s) для операции '%s'
1271	Illegal mix of collations for operation '%s' Недопустимое сочетание сопоставлений для операции '%s'
1272	Variable '%s' is not a variable component (can't be used as XXXX.variable_name) Переменная '%s' не является переменной составляющей (не может использоваться как XXXX.variable_name)
1273	Unknown collation: '%s' Неизвестное сопоставление: '%s'
1274	SSL parameters in CHANGE MASTER are ignored because this MySQL slave was compiled without SSL support; they can be used later if MySQL slave with SSL is started SSL-параметры в CHANGE MASTER игнорируются, потому что данный подчиненный сервер скомпилирован без поддержки SSL; их можно будет использовать потом, если подчиненный сервер MySQL будет запущен с опцией SSL
1275	Server is running in --secure-auth mode, but '%s'@'%s' has a password in the old format; please change the password to the new format Сервер работает в режиме --secure-auth, но использует пароль старого формата; измените формат пароля на новый.
1276	Field or reference '%s%s%s%s' of SELECT #%d was resolved in SELECT #%d Поле или ссылка '%s%s%s%s' в SELECT #%d была разрешена в SELECT #%d
1277	Incorrect parameter or combination of parameters for START SLAVE UNTIL Некорректный параметр или комбинация параметров для START SLAVE UNTIL
1278	It is recommended to use --skip-slave-start when doing step-bystep replication with START SLAVE UNTIL; otherwise, you will get problems if you get an unexpected slave's mysqld restart Рекомендуется использовать --skip-slave-start при выполнении пошаговой репликации с помощью START SLAVE UNTIL; иначе возникнут проблемы в случае неожиданного перезапуска mysqld на подчиненном сервере
1279	SQL thread is not to be started so UNTIL options are ignored Поток SQL пока запускаться не будет, поэтому опции UNTIL игнорируются
1280	Incorrect index name '%s' Некорректное имя индекса '%s'
1281	Incorrect catalog name '%s' Некорректное имя каталога '%s'

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1282	Query cache failed to set size %lu; new query cache size is %lu Попытка установить размер %lu для кэша запросов была неудачной; новый размер кэша запросов равен %lu
1283	Column '%s' cannot be part of FULLTEXT index Столбец '%s' не может быть частью индекса FULLTEXT
1284	Unknown key cache '%s' Неизвестный кэш ключей '%s'
1285	MySQL is started in --skip-name-resolve mode; you must restart it without this switch for this grant to work MySQL запущен в режиме --skip-name-resolve; необходимо перезапустить его без этой опции, чтобы данная привилегия работала
1286	Unknown table engine '%s' Неизвестный тип таблиц '%s'
1287	'%s' is deprecated; use '%s' instead '%s' устарела; вместо нее используйте '%s'
1288	The target table %s of the %s is not updatable Целевая таблица %s из %s не обновляется
1289	The '%s' feature is disabled; you need MySQL built with '%s' to have it working Функция '%s' отключена; чтобы она работала, потребуется выполнить сборку MySQL с '%s'
1290	The MySQL server is running with the %s option so it cannot execute this statement Сервер MySQL запущен с опцией %s, поэтому не может выполнить данный оператор
1291	Column '%s' has duplicated value '%s' in %s Столбец '%s' содержит дублированное значение '%s' в %s
1292	Truncated incorrect %s value: '%s' Усеченное некорректное значение %s: '%s'
1293	Incorrect table definition; there can be only one TIMESTAMP column with CURRENT_TIMESTAMP in DEFAULT or ON UPDATE clause Таблица определена некорректно; только один столбец может быть указан с CURRENT_TIMESTAMP в конструкции DEFAULT или ON UPDATE
1294	Invalid ON UPDATE clause for '%s' column Недопустимая конструкция ON UPDATE для столбца '%s'
1295	This command is not supported in the prepared statement protocol yet Данная команда еще не поддерживается в протоколе подготовленных операторов
1296	Can't create a %s from within another stored routine Не удастся создать %s через другую хранимую процедуру

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1297	%s %s already exists %s %s уже существует
1298	%s %s does not exist %s %s не существует
1299	Failed to DROP %s %s Не удалось удалить %s %s с помощью DROP
1300	Failed to CREATE %s %s Не удалось создать %s %s с помощью CREATE
1301	%s with no matching label: %s %s без подходящей метки: %s
1302	Redefining label %s Повторное определение метки %s
1303	End-label %s without match %s с конечной меткой без совпадений
1304	Referring to uninitialized variable %s Обращение к неинициализированной переменной %s
1305	SELECT in a stored procedure must have INTO Оператор SELECT в хранимой процедуре должен содержать INTO
1306	RETURN is only allowed in a FUNCTION RETURN разрешено использовать только в FUNCTION
1307	Statements like SELECT, INSERT, UPDATE (and others) are not allowed in a FUNCTION В FUNCTION запрещено использование операторов, подобных SELECT, INSERT, UPDATE (и других)
1308	The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been ignored Журнал регистрации обновлений устарел и был заменен бинарным журналом регистраций; команда SET SQL_LOG_UPDATE игнорируется
1309	The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been translated to SET SQL_LOG_BIN Журнал регистрации обновлений устарел и был заменен бинарным журналом регистраций; команда SET SQL_LOG_UPDATE переведена как SET SQL_LOG_BIN
1310	Query execution was interrupted Выполнение запроса было прервано
1311	Incorrect number of arguments for %s %s; expected %u, got %u Неправильное число аргументов для %s %s; ожидалось %u, получено %u
1312	Undefined CONDITION: %s Неопределенная команда CONDITION: %s
1313	No RETURN found in FUNCTION %s RETURN в FUNCTION %s не обнаружен.

Продолжение табл. 9.2

Код ошибки	Сообщение об ошибке
1314	FUNCTION %s ended without RETURN FUNCTION %s завершилась без RETURN
1315	Cursor statement must be a SELECT Оператором курсора должен быть SELECT
1316	Cursor SELECT must not have INTO Оператор курсора SELECT не должен содержать INTO
1317	Undefined CURSOR: %s Неопределенный курсор CURSOR: %s
1318	Cursor is already open Курсор уже открыт
1319	Cursor is not open Курсор не открыт
1320	Undeclared variable: %s Необъявленная переменная: %s
1321	Incorrect number of FETCH variables Неправильное число переменных FETCH
1322	No data to FETCH Данных для выборки (FETCH) нет
1323	Duplicate parameter: %s Дублированный параметр: %s
1324	Duplicate variable: %s Дублированная переменная: %s
1325	Duplicate condition: %s Дублированное условие: %s
1326	Duplicate cursor: %s Дублированный курсор: %s
1327	Failed to ALTER %s %s Не удалось изменить %s %s с помощью ALTER
1328	Subselect value not supported Значение подвыборки не поддерживается
1329	USE is not allowed in a stored procedure Использование USE в хранимой процедуре запрещено
1330	Variable or condition declaration after cursor or handler declaration Объявление переменной или условия после объявления курсора или обработчика
1331	Cursor declaration after handler declaration Объявление курсора после объявления обработчика
1332	Case not found for CASE statement Не найдена конструкция Case для оператора CASE

*Окончание табл. 9.2*

Код ошибки	Сообщение об ошибке
1333	Configuration file '%s' is too big Размер файла конфигурации слишком большой
1334	Malformed file type header in file '%s' Некорректно заданный заголовок типа файла в файле '%s'
1335	Unexpected end of file while parsing comment '%s' Неожиданный конец файла при анализе комментария '%s'
1336	Error while parsing parameter '%s' (line: '%s') Ошибка при анализе параметра '%s'
1337	Unexpected end of file while skipping unknown parameter '%s' Неожиданный конец файла при пропуске неизвестного параметра '%s'

# А

## Поиск и устранение проблем с запросами

**В** данном приложении перечислены некоторые общие проблемы и возможные сообщения об ошибках, которые могут возникнуть при выполнении SQL-операторов, а также способы выхода из той или иной проблемной ситуации.

### А.1. Проблемы, связанные с запросами

#### А.1.1. Чувствительность к регистру во время поиска

По умолчанию во время выполнения операций поиска в MySQL регистр не учитывается (хотя существуют определенные наборы символов, которые нечувствительными к регистру быть не могут, как, например, набор *czech*). Это означает, что во время поиска с помощью *имя\_столбца* `LIKE 'a%'`, будут представлены все значения столбца, которые начинаются с А или а. Если нужно, чтобы при таком поиске регистр учитывался, следует проследить, чтобы один из операндов являлся бинарной строкой. Также добиться подобного результата можно и с помощью оператора `BINARY`. Запишите условие либо как `BINARY имя_столбца LIKE 'a%'`, либо как `имя_столбца LIKE BINARY 'a%'`.

При необходимости, чтобы во время поиска имени столбца регистр учитывался всегда, объявите столбец как `BINARY`. См. раздел 6.2.5.

Простые операции сравнения (`>=`, `>`, `=`, `<`, `<=`, сортировка и группирование) основаны на “значении сортировки” каждого символа. Символы с одинаковым значением сортировки (такие как ‘Е’, ‘е’ и ‘ё’) трактуются как один и тот же символ.

Если используются данные на китайском языке в так называемой кодировке `big5`, все символьные столбцы могут быть объявлены как `BINARY`. Это сработает, поскольку порядок сортировки символов кодировки `big5` основывается на порядке кодов ASCII. Начиная с версии MySQL 4.1, объявить, что для столбца всегда должен использоваться набор символов `big5`, можно следующим образом:

```
CREATE TABLE t (name CHAR(40) CHARACTER SET big5);
```

## А.1.2. Проблемы при использовании столбцов DATE

Для значения DATE используется формат 'ГГГГ-ММ-ДД'. Согласно SQL-стандарту, любой другой формат является недопустимым. Данный формат должен применяться в выражениях UPDATE и в конструкциях WHERE операторов SELECT, например:

```
mysql> SELECT * FROM имя_таблицы WHERE date >= '2003-05-05';
```

Для удобства MySQL автоматически преобразует дату в число, если дата используется в числовом контексте (и наоборот), а также допускает использование “ослабленной” строковой формы при обновлении и в конструкции WHERE, сравнивающей дату со столбцом TIMESTAMP, DATE или DATETIME. (“Ослабленная форма” означает, что в качестве разделителя между частями может применяться любой знак пунктуации. Например, '2004-08-15' и '2004#08#15' эквивалентны.) MySQL может также преобразовывать строку, не содержащую разделителей (как, например, '20040815'), при условии что она имеет смысл в качестве даты.

Специальную дату '0000-00-00' можно сохранять и извлекать в виде '0000-00-00'. При использовании даты типа '0000-00-00' через Connector/ODBC, в версии Connector/ODBC 2.50.12 и выше она будет автоматически преобразовываться в NULL, поскольку ODBC не может обрабатывать такой формат даты.

Поскольку MySQL выполняет описанные выше типы преобразований, работать будут следующие операторы:

```
mysql> INSERT INTO имя_таблицы (idate) VALUES (19970505);
mysql> INSERT INTO имя_таблицы (idate) VALUES ('19970505');
mysql> INSERT INTO имя_таблицы (idate) VALUES ('97-05-05');
mysql> INSERT INTO имя_таблицы (idate) VALUES ('1997.05.05');
mysql> INSERT INTO имя_таблицы (idate) VALUES ('1997 05 05');
mysql> INSERT INTO имя_таблицы (idate) VALUES ('0000-00-00');

mysql> SELECT idate FROM имя_таблицы WHERE idate >= '1997-05-05';
mysql> SELECT idate FROM имя_таблицы WHERE idate >= 19970505;
mysql> SELECT MOD(idate,100) FROM имя_таблицы WHERE idate >= 19970505;
mysql> SELECT idate FROM имя_таблицы WHERE idate >= '19970505';
```

Однако такой оператор работать не будет:

```
mysql> SELECT idate FROM имя_таблицы WHERE STRCMP(idate,'20030505')=0;
```

Функция STRCMP() является строковой, поэтому она преобразует idate в строку формата 'ГГГГ-ММ-ДД' и выполняет операцию по сравнению строк. Преобразовывать '20030505' в дату и сравнивать даты она не будет.

Сервер MySQL упаковывает даты для хранения, поэтому сохранить определенную дату, если она для буфера результатов не подходит, он не может. MySQL выполняет очень ограниченную проверку того, является ли та или иная дата правильной. При сохранении некорректно указанной даты, например, '2004-2-31', MySQL сохранит ее именно в таком виде. Поддерживаются следующие правила принятия дат:

- Если MySQL может сохранять и извлекать дату в заданном формате, она принимается для столбцов DATE и DATETIME даже в случае, когда такой ее формат не является стопроцентно корректным.

- Значения дней в интервале от 0 до 31 принимаются для любой даты. Это очень удобно для Web-приложений, в которых год, месяц и день указываются в трех разных полях.
- Значение дня или месяца может быть равно нулю. Это удобно, когда, например, нужно сохранить дату рождения в столбце DATE, а она не известна полностью.

Если дата не может быть преобразована в какое-нибудь подходящее значение, в столбце DATE сохраняется значение 0, и при извлечении дата будет выглядеть как '0000-00-00'. Делается это для удобства, а также из соображений, связанных с производительностью. Мы полагаем, что сервер баз данных должен извлекать дату в сохраненном вами виде (даже если эта дата не была логически корректной во всех случаях), а также мы считаем, что приложение, а не сервер отвечает за проверку дат.

### A.1.3. Проблемы со значениями NULL

Понятие значения NULL является наиболее распространенным источником заблуждения для новичков в SQL, которые часто думают, что NULL – это то же самое, что и пустая строка ''. Это совершенно не верно. Например, следующие операторы абсолютно разные:

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);  
mysql> INSERT INTO my_table (phone) VALUES ('');
```

Оба оператора вставляют значение в столбец phone, но первый из них вставляет значение NULL, а второй – пустую строку. Смысл первого можно рассматривать как “номер телефона не известен”, а смысл второго – как “известно, что у этого человека нет телефона, а, следовательно, у него нет и номера телефона”.

С обработкой NULL помочь могут операции IS NULL и NOT NULL, а также функция IFNULL().

В SQL значение NULL никогда не бывает истинным при сравнении с любым другим значением и даже со значением NULL. Выражение, которое содержит NULL, всегда выдает значение NULL, если только в документации по операциям и функциям, задействованным в таком выражении, не было указано какое-нибудь другое значение. В следующем примере все столбцы возвращают NULL:

```
mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);
```

Для поиска в столбцах значений NULL использовать критерий `выражение = NULL` нельзя. Следующий оператор не возвращает никаких строк, поскольку `выражение = NULL` никогда не бывает истинным для любого выражения.

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

Для поиска значений NULL необходимо использовать конструкцию IS NULL. Следующие операторы демонстрируют, как найти телефонный номер NULL или пустой телефонный номер:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;  
mysql> SELECT * FROM my_table WHERE phone = '';
```

Добавлять индекс для столбца, наличие значений NULL в котором не исключено, можно при условии, что используется версия MySQL 3.23.2 или выше, а также механизм хранения MyISAM, InnoDB или BDB. Начиная с MySQL 4.0.2, механизмом хранения MEMORY также поддерживаются значения NULL в индексах. Во всех остальных случаях потребует-



ся объявлять индексированные таблицы с помощью NOT NULL, а также вставить NULL в столбец будет невозможно.

При считывании данных с помощью LOAD DATA INFILE пустые или недостающие столбцы обновляются со значениями ''. Если нужно в столбец занести значение NULL, используйте \N в файле данных. В определенных обстоятельствах также может использоваться и литеральное слово "NULL". См. раздел 6.1.5.

При применении DISTINCT, GROUP BY или ORDER BY все значения NULL рассматриваются как равные.

Когда используется ORDER BY, значения NULL предоставляются первыми или же последними, если указать DESC для сортировки в порядке убывания. Исключение: в MySQL 4.0.2 (как и в версии 4.0.10) значения NULL сортируются первыми, независимо от порядка сортировки.

Агрегатные (итоговые) функции, такие как COUNT(), MIN() и SUM(), игнорируют все значения NULL. Исключение составляет функция COUNT(\*), которая подсчитывает строки, а не отдельные значения в столбцах. Например, следующий оператор выполняет два типа подсчета. Сначала подсчитывается число строк в таблице, а затем – количество неравных NULL значений в столбце age:

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

Для некоторых типов столбцов MySQL обрабатывает значения NULL специальным образом. При вставке NULL в столбец TIMESTAMP в него добавляются значения текущей даты и времени. При вставке NULL в столбец целых чисел с атрибутом AUTO\_INCREMENT вставлено будет следующее число в последовательности.

### А.1.4. Проблемы с псевдонимами столбцов

В конструкциях GROUP BY, ORDER BY или HAVING псевдоним можно использовать для ссылки на столбец; также псевдонимы могут применяться с целью дать тому или иному столбцу более подходящее название:

```
SELECT SQRT(a*b) AS route FROM имя_таблицы GROUP BY route HAVING route > 0;  
SELECT id, COUNT(*) AS cnt FROM имя_таблицы GROUP BY id HAVING cnt > 0;  
SELECT id AS 'Customer identity' FROM имя_таблицы;
```

Стандарт SQL запрещает ссылаться на псевдоним столбца в конструкции WHERE. Причина этого заключается в том, что во время выполнения кода WHERE значение столбца может оказаться еще не определенным. Например, следующий запрос является недопустимым:

```
SELECT id, COUNT(*) AS cnt FROM имя_таблицы WHERE cnt > 0 GROUP BY id;
```

Конструкция WHERE выполняется для определения, какие строки должны быть включены в часть GROUP BY, в то время как HAVING применяется для определения, какие строки из набора результатов должны быть использованы.

### А.1.5. Сбой оператора ROLLBACK при работе с нетранзакционными таблицами

Получение следующего сообщения при попытке выполнить ROLLBACK означает, что одна или более таблиц, которые использовались в транзакции, транзакций не поддерживают.

```
Warning: Some non-transactional changed tables couldn't be rolled back
```

На такие нетранзакционные таблицы оператор `ROLLBACK` не действует.

Если намеренного смещения транзакционных и нетранзакционных таблиц при выполнении транзакции не было, причина выдачи подобного сообщения, скорее всего, будет заключаться в том, что таблица, рассматриваемая как транзакционная, на самом деле таковой не является. Такое может случиться при попытке создать таблицу с использованием транзакционного механизма хранения, который не поддерживается сервером `mysqld` (или был отключен с помощью опции запуска). Если `mysqld` не поддерживает механизм хранения, он вместо этого создает таблицу как таблицу `MyISAM`, которая транзакционной не является.

Проверить тип той или иной таблицы можно с помощью следующих операторов:

```
SHOW TABLE STATUS LIKE 'имя_таблицы';  
SHOW CREATE TABLE имя_таблицы;
```

См. разделы 6.5.3.17 и 6.5.3.6.

Проверить поддерживаемые сервером `mysqld` механизмы хранения можно с помощью следующего оператора:

```
SHOW ENGINES;
```

В версиях `MySQL`, предшествующих 4.1.2, оператор `SHOW ENGINES` недоступен. Используйте вместо него следующий оператор и проверяйте значение переменной, отвечающей за интересующий вас механизм хранения:

```
SHOW VARIABLES LIKE 'have_%';
```

Например, для определения, поддерживается ли `InnoDB`, следует проверить значение переменной `have_innodb`.

См. разделы 6.5.3.8 и 6.5.3.19.

## А.1.6. Удаление строк из связанных таблиц

В версиях `MySQL`, предшествующих 4.1, не поддерживаются подзапросы, а в версиях ниже 4.0 не поддерживается использование более чем одной таблицы в операторе `DELETE`. Если установлена именно одна из упомянутых версий, удалить строки из двух связанных между собой таблиц можно с помощью следующего подхода:

1. В главной таблице выберите строки с помощью `SELECT`, основываясь на некотором условии `WHERE`.
2. Удалите строки с помощью `DELETE` из главной таблицы, основываясь на том же самом условии.
3. Удалите строки с помощью `DELETE FROM связанная_таблица WHERE связанный_столбец IN (выбранные_строки)`.

Если общая длина оператора для таблицы `связанная_таблица` превышает 1 Мбайт (значение системной переменной `max_allowed_packet` по умолчанию), необходимо разделить его на части поменьше и выполнить сразу несколько операторов `DELETE`. Скорее всего, самый быстрый оператор `DELETE` получится при указании только от 100 до 1000 значений `связанный_столбец` в каждом таком операторе, если `связанный_столбец` проиндексирован. Если столбец `связанный_столбец` не проиндексирован, скорость выполнения не будет зависеть от количества аргументов в конструкции `IN`.

### A.1.7. Решение проблем с несовпадающими строками

При наличии сложного запроса, использующего сразу несколько таблиц, но не возвращающего при этом ни одной строки, выяснить причину такой проблемы можно следующим образом:

1. Протестируйте запрос с помощью EXPLAIN, чтобы проверить его на наличие явных ошибок.
2. Выберите только те столбцы, которые указаны в конструкции WHERE.
3. Удаляйте по одной таблице из запроса до тех пор, пока он не начнет возвращать хоть какие-нибудь строки. Если размер таблиц большой, то целесообразно указать в запросе LIMIT 10.
4. Выполните SELECT для столбца, в котором должна была совпасть строка с последней удаленной из запроса таблицей.
5. При сравнении столбцов FLOAT или DOUBLE, числа в которых содержат дробные части, использовать сравнение на предмет равенства (=) нельзя. Это наиболее распространенная проблема в большинстве компьютерных языков, поскольку не все значения с плавающей запятой могут быть сохранены с максимальной точностью. В некоторых случаях устранить такую проблему помогает замена FLOAT на DOUBLE. См. раздел A.1.8.
6. Если по-прежнему выяснить причину проблемы не удастся, создайте минимальный тест, запустить который можно с помощью `mysql test < query.sql` и который отобразит имеющиеся ошибки. Создать тестовый файл можно, выполнив дамп таблиц посредством `mysqldump-quick имя_базы_данных имя_таблицы_1 ... имя_таблицы_n > query.sql`. Откройте файл в редакторе, удалите некоторые строки со вставками (если их больше, чем требуется для отображения проблемы) и добавьте в конец файла свой оператор SELECT.

Удостоверьтесь, что тестовый файл иллюстрирует проблему, выполнив следующие команды:

```
shell> mysqladmin create test2
shell> mysql test2 < query.sql
```

С помощью `mysqlbug` отправьте тестовый файл в общий список рассылки MySQL. См. раздел I.7.1.1.

### A.1.8. Проблемы при сравнении чисел с плавающей запятой

Числа с плавающей запятой иногда доставляют неприятности, поскольку как точные значения внутри компьютерной архитектуры они не хранятся. То, что отображается на экране, обычно не является точным значением числа. К типам столбцов, которые хранят числа с плавающей запятой, относятся FLOAT, DOUBLE и DECIMAL. Столбцы DECIMAL хранят максимально точные значения, поскольку значения в них представлены в виде строк, но вот вычисления по значениям из столбцов DECIMAL могут выполняться с использованием операций с плавающей запятой.

Следующий пример иллюстрирует проблему. Он показывает, что даже для столбцов типа DECIMAL вычисления, осуществляемые посредством операций с плавающей запятой, не исключают ошибок с плавающей запятой.

```
mysql> CREATE TABLE t1 (i INT, d1 DECIMAL(9,2), d2 DECIMAL(9,2));
mysql> INSERT INTO t1 VALUES (1, 101.40, 21.40), (1, -80.00, 0.00),
-> (2, 0.00, 0.00), (2, -13.20, 0.00), (2, 59.60, 46.40),
-> (2, 30.40, 30.40), (3, 37.00, 7.40), (3, -29.60, 0.00),
-> (4, 60.00, 15.40), (4, -10.60, 0.00), (4, -34.00, 0.00),
-> (5, 33.00, 0.00), (5, -25.80, 0.00), (5, 0.00, 7.20),
-> (6, 0.00, 0.00), (6, -51.40, 0.00);
```

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b
-> FROM t1 GROUP BY i HAVING a <> b;
```

i	a	b
1	21.40	21.40
2	76.80	76.80
3	7.40	7.40
4	15.40	15.40
5	7.20	7.20
6	-51.40	0.00

Результат правильный. Хотя первые пять записей выглядят так, будто они не должны успешно проходить тест на сравнение (значения a и b разными не кажутся), однако они проходят его, поскольку разница между числами проявляется приблизительно на уровне десятого знака, в зависимости от архитектуры компьютера.

Устранить проблему с помощью ROUND() или других подобных функций нельзя, поскольку результат будет по-прежнему представлять собой число с плавающей запятой.

```
mysql> SELECT i, ROUND(SUM(d1), 2) AS a, ROUND(SUM(d2), 2) AS b
-> FROM t1 GROUP BY i HAVING a <> b;
```

i	a	b
1	21.40	21.40
2	76.80	76.80
3	7.40	7.40
4	15.40	15.40
5	7.20	7.20
6	-51.40	0.00

Так выглядят числа в столбце при отображении большего количества знаков после запятой:

```
mysql> SELECT i, ROUND(SUM(d1), 2)*1.0000000000000000 AS a,
-> ROUND(SUM(d2), 2) AS b FROM t1 GROUP BY i HAVING a <> b;
```

i	a	b
1	21.3999999999999986	21.40

	2		76.799999999999972		76.80	
	3		7.4000000000000004		7.40	
	4		15.4000000000000004		15.40	
	5		7.2000000000000002		7.20	
	6		-51.399999999999986		0.00	
+-----+-----+-----+						

Увидите вы подобные результаты или нет, будет зависеть от архитектуры вашего компьютера. Разные центральные процессоры могут выполнять операции с плавающей запятой по-разному. Например, на некоторых машинах получить “корректные” результаты можно, умножив оба аргумента на 1, как показано в следующем примере.

### Внимание!

Никогда не используйте этот метод в своих приложениях. Это отнюдь не пример надежного метода!

```
mysql> SELECT i, ROUND(SUM(d1), 2)*1 AS a, ROUND(SUM(d2), 2)*1 AS b
-> FROM t1 GROUP BY i HAVING a <> b;
```

+-----+-----+-----+		
i	a	b
+-----+-----+-----+		
6	-51.40	0.00
+-----+-----+-----+		

Кажется, что приведенный выше пример должен работать. Причина этого заключается в том, что на той определенной машине, на которой проводился тест, центральный процессор во время операций с плавающей запятой округляет числа до одинакового значения. Однако правила, что так должен делать любой центральный процессор, не существует, поэтому на такой метод полагаться нельзя.

Правильным для сравнения чисел с плавающей запятой будет сначала определиться с допустимой степенью различия между числами, а затем выполнить операцию сравнения с учетом такого допустимого значения. Например, приняв договоренность, что числа с плавающей запятой должны рассматриваться как одинаковые, если они равны с точностью до одной десятичной (0,0001), сравнение для выявления чисел, превышающих допустимый уровень отклонения, будет выполняться следующим образом:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
-> GROUP BY i HAVING ABS(a - b) > 0.0001;
```

+-----+-----+-----+		
i	a	b
+-----+-----+-----+		
6	-51.40	0.00
+-----+-----+-----+		
1 row in set (0.00 sec)		

И наоборот, если нужно получить строки, в которых числа являются одинаковыми, тест должен находить отличия в пределах допустимого значения отклонения:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
-> GROUP BY i HAVING ABS(a - b) <= 0.0001;
```

+-----+-----+-----+						
	i		a		b	
+-----+-----+-----+						
	1		21.40		21.40	

	2		76.80		76.80	
	3		7.40		7.40	
	4		15.40		15.40	
	5		7.20		7.20	
+-----+-----+-----+						

## А.2. Проблемы, связанные с оптимизатором

MySQL использует оптимизатор затрат для определения лучшего способа разрешения запроса. В большинстве случаев MySQL способен вычислить наиболее эффективный из возможных план, но иногда у него в наличии нет достаточного количества информации по данным, и ему приходится строить о них “логические” предположения.

В ситуациях, когда MySQL “делает не то, что нужно”, помочь ему можно следующим образом:

- Воспользуйтесь оператором EXPLAIN, чтобы получить информацию о том, каким образом MySQL будет обрабатывать запрос. Для этого необходимо просто добавить ключевое слово EXPLAIN в начале оператора SELECT.

```
mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
```

Оператор EXPLAIN более подробно рассматривается в книге *MySQL. Руководство администратора*.

- Используйте ANALYZE TABLE *имя\_таблицы* для обновления распределений ключей сканируемой таблицы. См. раздел 6.5.2.1.
- Используйте FORCE INDEX для сканируемой таблицы, чтобы сообщить MySQL о том, что сканирование таблиц обходится дороже по сравнению с применением данного индекса. См. раздел 6.1.7.

```
SELECT * FROM t1, t2 FORCE INDEX (индекс_по_столбцу)
WHERE t1.имя_столбца=t2.имя_столбца;
```

Также могут пригодиться USE INDEX и IGNORE INDEX.

- Используйте STRAIGHT\_JOIN как глобально, так и на уровне таблиц. См. раздел 6.1.7.
- Можно настроить глобальные и сеансовые системные переменные. Например, запустите mysqld с опцией --max-seeks-for-key=1000 или используйте SET max\_seeks\_for\_key=1000, чтобы оптимизатор при сканировании ключей разрешал не более 1000 найденных ключей.

## А.3. Проблемы, связанные с определением таблиц

### А.3.1. Проблемы с ALTER TABLE

Оператор ALTER TABLE изменяет таблицу на текущий набор символов. Если во время выполнения ALTER TABLE появляется ошибка дублирования ключей, причина будет заключаться либо в том, что новый набор символов преобразует ключи в одинаковые значения, либо в том, что таблица повреждена. В последнем случае для таблицы потребуются выполнить REPAIR TABLE.

Если при использовании ALTER TABLE появляется ошибка, указанная ниже, проблема скорее всего связана с тем, что работа MySQL была аварийно прекращена во время более ранней операции ALTER TABLE и где-то рядом осталась старая таблица с именем А-xxx или В-xxx:

```
Error on rename of './имя_базы_данных/имя.frm'  
to './имя_базы_данных/В-xxx.frm' (Errcode: 17)
```

В таком случае откройте каталог данных MySQL и удалите все файлы, имена которых начинаются с А- или В-. (Вместо удаления их можно и просто переместить в другой каталог.)

ALTER TABLE работает следующими образом:

- Создается новая таблица с именем А-xxx с заданными структурными изменениями.
- Все строки из исходной таблицы копируются в таблицу А-xxx.
- Исходная таблица переименовывается на В-xxx.
- Таблице А-xxx присваивается имя исходной таблицы.
- Таблица В-xxx удаляется.

Если во время операции по переименованию что-нибудь пойдет не так, MySQL попытается отменить изменения. Если случится нечто действительно серьезное (чего конечно произойти не должно), MySQL может оставить старую таблицу с именем В-xxx. Простое переименование файлов таблиц на уровне системы позволит вернуть ваши данные обратно.

При применении ALTER TABLE на транзакционной таблице или если используется операционная система Windows или OS/2, ALTER TABLE разблокирует таблицу через UNLOCK в том случае, когда таблица была заблокирована с помощью LOCK TABLE. Причина этого состоит в том, что InnoDB и упомянутые операционные системы не могут удалять таблицы, находящиеся в использовании.

### А.3.2. Изменение порядка столбцов в таблице

Прежде всего, потребуется определиться с тем, действительно ли необходимо изменять порядок столбцов в таблице. Весь смысл SQL заключается в том, чтобы отделить приложение от формата хранения данных. Следует всегда указывать порядок извлечения данных. Первый из представленных ниже операторов возвращает столбцы в порядке *имя\_столбца1*, *имя\_столбца2*, *имя\_столбца3*, в то время как второй возвращает их в порядке *имя\_столбца1*, *имя\_столбца3*, *имя\_столбца2*:

```
mysql> SELECT имя_столбца1, имя_столбца2, имя_столбца3 FROM имя_таблица;  
mysql> SELECT имя_столбца1, имя_столбца3, имя_столбца2 FROM имя_таблица;
```

Если принято решение изменить порядок столбцов таблицы в любом случае, сделать это можно следующим образом:

1. Создайте новую таблицу, столбцы в которой размещены в требуемом порядке.
2. Выполните такой оператор:

```
mysql> INSERT INTO новая_таблица  
-> SELECT столбцы-в-новом-порядке FROM старая_таблица;
```

3. Удалите или переименуйте старая\_таблица.

4. Присвойте новой таблице имя старой таблицы:

```
mysql> ALTER TABLE новая_таблица RENAME старая_таблица;
```

SELECT \* вполне подходит для тестирования запросов. Однако, в приложении *никогда* нельзя полагаться на использование SELECT \* и извлекать столбцы, основываясь на их позициях. Позиция и порядок, в котором возвращаются столбцы, не остаются прежними, если столбцы добавляются, перемещаются или удаляются. Простое изменение в структуре таблицы может привести к сбою в работе приложения.

### А.3.3. Проблемы с TEMPORARY TABLE

В следующем списке перечислены ограничения по использованию временных (TEMPORARY) таблиц:

- Временной (TEMPORARY) таблицей может быть только таблица типа HEAP, ISAM, MYISAM, MERGE или InnoDB.
- В одном и том же запросе нельзя ссылаться на таблицу TEMPORARY более одного раза. Например, следующий запрос не работает:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;  
ERROR 1137: Can't reopen table: 'temp_table'
```

- Оператор SHOW TABLES временные таблицы не показывает.
- Нельзя использовать RENAME для переименования таблицы TEMPORARY. Вместо этого можно применять ALTER TABLE:

```
mysql> ALTER TABLE исходное_имя RENAME новое_имя;
```



## Регулярные выражения MySQL

**Р**егулярное выражение (regular expression) – это мощное средство задания образца для сложного типа поиска.

В MySQL используется предложенная Генри Спенсером (Henry Spencer) реализация регулярных выражений, предназначенная для соответствия стандарту POSIX 1003.2. Это расширенная версия, в которой поддерживаются операции сравнения с образцом, выполняемые оператором REGEXP в операторах SQL.

В данном приложении предлагается краткий обзор выражений, с примерами специальных символов и конструкций, которые могут применяться в MySQL для операций типа REGEXP. Более подробную информацию можно найти на странице руководства Генри Спенсера (Henry Spencer) [regex\(7\)](#), которая включена в исходные дистрибутивы и находится в файле `regex.7`, расположенном в каталоге `regex`.

Регулярное выражение описывает набор строк. Самым простым регулярным выражением является то, которое не содержит в себе специальных символов. Например, регулярное выражение `hello` совпадает с `hello` и ничем другим.

В нетривиальных регулярных выражениях используются специальные конструкции, так что совпадать они смогут больше, чем с одной строкой. Например, регулярное выражение `hello|word` соответствует как строке `hello`, так и строке `word`.

В качестве более сложного примера возьмем регулярное выражение `B[an]*s`; оно будет соответствовать любой из строк `Bananas`, `Baaaaas`, `Bs` или любой другой строке, начинающейся с `B`, заканчивающейся на `s` и содержащей любое количество символов `a` или `n` между ними.

В регулярном выражении для оператора REGEXP могут использоваться следующие специальные символы и конструкции:

- `^`. Соответствовать началу строки.

```
mysql> SELECT 'fo\nfo' REGEXP '^fo$';      -> 0
mysql> SELECT 'fofo' REGEXP '^fo';         -> 1
```

- `$`. Соответствовать концу строки.

```
mysql> SELECT 'fo\no' REGEXP '^fo\no$';    -> 1
mysql> SELECT 'fo\no' REGEXP '^fo$';       -> 0
```

- `.` Соответствовать любому символу (включая символы возврата каретки и новой строки).

```
mysql> SELECT 'fofo' REGEXP '^f.*$';          -> 1
mysql> SELECT 'fo\r\nfo' REGEXP '^f.*$';      -> 1
```

- `a*`. Соответствовать любой последовательности, состоящей из нуля или большего числа символов `a`.

```
mysql> SELECT 'Ban' REGEXP '^Ba*n';           -> 1
mysql> SELECT 'Baaan' REGEXP '^Ba*n';         -> 1
mysql> SELECT 'Bn' REGEXP '^Ba*n';            -> 1
```

- `a+`. Соответствовать любой последовательности, состоящей из одного или больше символов `a`.

```
mysql> SELECT 'Ban' REGEXP '^Ba+n';           -> 1
mysql> SELECT 'Bn' REGEXP '^Ba+n';            -> 0
```

- `a?`. Соответствовать нулю или одному символу `a`.

```
mysql> SELECT 'Bn' REGEXP '^Ba?n';            -> 1
mysql> SELECT 'Ban' REGEXP '^Ba?n';           -> 1
mysql> SELECT 'Baan' REGEXP '^Ba?n';          -> 0
```

- `de|abc`. Соответствовать как последовательности `de`, так и последовательности `abc`.

```
mysql> SELECT 'pi' REGEXP 'pi|apa';           -> 1
mysql> SELECT 'axe' REGEXP 'pi|apa';          -> 0
mysql> SELECT 'apa' REGEXP 'pi|apa';          -> 1
mysql> SELECT 'apa' REGEXP '^ (pi|apa) $';     -> 1
mysql> SELECT 'pi' REGEXP '^ (pi|apa) $';     -> 1
mysql> SELECT 'pix' REGEXP '^ (pi|apa) $';    -> 0
```

- `(abc)*`. Соответствовать нулю или большему числу экземпляров последовательности `abc`.

```
mysql> SELECT 'pi' REGEXP '^ (pi) * $';       -> 1
mysql> SELECT 'pip' REGEXP '^ (pi) * $';      -> 0
mysql> SELECT 'pippi' REGEXP '^ (pi) * $';    -> 1
```

- `{1}`, `{2,3}`. Запись через `{n}` или `{m,n}` – это более общий способ написания регулярных выражений, совпадающих со многими экземплярами предыдущего атома (или “элемента”) образца. `m` и `n` представляют собой целые числа.

- `a*`. Можно записать как `a{0,}`.
- `a+`. Можно записать как `a{1,}`.
- `a?`. Можно записать как `a{0,1}`.

То есть `a{n}` в точности соответствует `n` числу экземпляров символа `a`. `a{n,}` соответствует `n` или большему числу экземпляров символа `a`. `a{m,n}` соответствует от `m` до `n` экземпляров символа `a`, включительно.

Числа `m` и `n` должны находиться в интервале от 0 до `RE_DUP_MAX` (по умолчанию 255) включительно. Если одновременно задаются `m` и `n`, аргумент `m` должен быть меньше или равен по значению аргументу `n`.

```
mysql> SELECT 'abcde' REGEXP 'a[bcd]{2}e';    -> 0
mysql> SELECT 'abcde' REGEXP 'a[bcd]{3}e';    -> 1
mysql> SELECT 'abcde' REGEXP 'a[bcd]{1,10}e'; -> 1
```

- `[a-dX]`, `^[a-dX]`. Соответствует любому символу, который является (или не является, если используется `^`) любым из символов `a`, `b`, `c`, `d` или `X`. Символ `-` между двумя другими символами образует интервал, соответствующий всем символам от первого указанного символа до второго. Например, `[0-9]` будет соответствовать любой десятичной цифре. При включении литерального символа `]`, его следует ставить сразу же после открывающей скобки `[`. При включении литерального символа `-`, его следует ставить в самом начале или в самом конце. Любой символ, не имеющий специального определенного значения внутри пары скобок `[]`, будет совпадать только с самим собой.

```
mysql> SELECT 'aXbc' REGEXP '[a-dXYZ]';      -> 1
mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]$';    -> 0
mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]+$';    -> 1
mysql> SELECT 'aXbc' REGEXP '^[^a-dXYZ]+$';   -> 0
mysql> SELECT 'gheis' REGEXP '^[^a-dXYZ]+$';  -> 1
mysql> SELECT 'gheisa' REGEXP '^[^a-dXYZ]+$'; -> 0
```

- `[.символы.]`. В выражении со скобками (записанном с помощью `'` и `'`) совпадает с последовательностью символов этого элемента сравнения. `.символы.` — это либо один символ, либо наименование символа, например `newline`. Полный перечень наименований символов можно найти в файле `regex/cname.h`.

```
mysql> SELECT '~' REGEXP '[[.~.]]';          -> 1
mysql> SELECT '~' REGEXP '[[.tilde.]]';      -> 1
```

- `[=класс_символов=]`. В выражении со скобками (записанном с помощью `'` и `'`) `[=класс_символов=]` представляет класс эквивалентности: символы из этого класса будут совпадать со всеми символами, имеющими одинаковое значение соответствия, включая самих себя. Например, если `o` и `(+)` являются членами класса эквивалентности, тогда `[=o=]`, `[={+}=]` и `[o(+=)]` все будут синонимичными. Класс эквивалентности нельзя использовать в качестве конечной точки диапазона.
- `[[:класс_символов:]]`. В выражении со скобками (записанном с помощью `'` и `'`) `[[:класс_символов:]]` представляет класс символов, который будет соответствовать всем символам, принадлежащим этому классу. Ниже представлены имена стандартных классов:

<code>alnum</code>	Алфавитно-цифровые символы.
<code>alpha</code>	Алфавитные символы.
<code>blank</code>	Символы табуляции.
<code>cntrl</code>	Управляющие символы.
<code>digit</code>	Цифровые символы.
<code>graph</code>	Графические символы.
<code>lower</code>	Символы букв нижнего регистра.
<code>print</code>	Графические символы или символы пробела.
<code>punct</code>	Знаки пунктуации.
<code>space</code>	Символы пробела, табуляции, абзаца и возврата каретки.
<code>upper</code>	Символы букв верхнего регистра.
<code>xdigit</code>	Символы шестнадцатеричных цифр.

Эти имена указывают на классы символов, определенных на странице руководства *ctype(3)*. Конкретный национальный набор символов может порождать и другие имена классов. Класс символов нельзя использовать в качестве конечной точки диапазона.

```
mysql> SELECT 'justalnums' REGEXP '[:alnum:]+';      -> 1
mysql> SELECT '!' REGEXP '[:alnum:]+';              -> 0
```

- `[[:<:]]`, `[[:>:]]`. С помощью данных маркеров обозначаются границы слова. Они ищут соответствие в начале и конце слова. Слово представляет собой последовательность символов слова, которой не предшествуют или за которой не следуют никакие символы слов. Символ слова – это алфавитно-цифровой символ из класса `alnum` или же символ подчеркивания (`_`).

```
mysql> SELECT 'a word a' REGEXP '[[:<:]]word[[:>:]]'; -> 1
mysql> SELECT 'a xword a' REGEXP '[[:<:]]word[[:>:]]'; -> 0
```

При использовании литерального экземпляра специального символа в регулярном выражении перед ним необходимо ставить две обратных косых черты (`\\`). Одна обратная косая черта интерпретируется анализатором MySQL, а вторая интерпретируется через библиотеку регулярных выражений. Например, для совпадения со строкой `1+2`, содержащей специальный символ `+`, корректным будет только последнее из следующих регулярных выражений:

```
mysql> SELECT '1+2' REGEXP '1+2';                  -> 0
mysql> SELECT '1+2' REGEXP '1\\+2';                 -> 0
mysql> SELECT '1+2' REGEXP '1\\\\+2';                 -> 1
```

# Предметный указатель

## А

API-интерфейс, 19; 20; 31; 68; 78; 346

## Г

GPL, См. Лицензия GNU General Public License, 18; 29; 32–34; 36; 39; 56

## О

OGC, См. Консорциум Open GIS Consortium, 331; 332; 339

## С

SRID, См. Идентификатор пространственной системы координат, 333; 334; 341–343; 348

## А

Атомарная операция, 62

Атрибут

BINARY, 60; 90; 102; 103; 114; 124; 125; 136; 137; 143; 150; 151; 163; 164; 194; 262; 319; 320; 402

NATIONAL, 111; 124; 125

UNICODE, 110; 124; 125; 137; 262

UNSIGNED, 60; 79; 91; 120–123; 126–128; 143; 185; 193–195; 258; 262; 265

ZEROFILL, 60; 91; 120–123; 126; 127; 262

## Б

Библиотека

readline, 32

regex, 32; 45; 388; 415

## В

Взаимная блокировка, 47; 70

## Г

Геометрия, 332; 335–337; 354; 355

## Д

Двойная лицензия, 15

## И

Идентификатор пространственной системы координат, 333

## К

Класс

Curve, 332; 333; 335; 337; 338

GeometryCollection, 332; 333; 336; 339; 340; 343; 352

LineString, 332–340; 342–344; 346–352; 358

MultiCurve, 332; 333; 337

MultiLineString, 332; 333; 337–340; 342; 343; 350; 358

MultiPoint, 332–334; 337; 339; 340; 343; 354

MultiPolygon, 333; 334; 338–340; 342; 343; 347; 351; 352; 354

MultiSurface, 333; 338

Point, 332–337; 339; 340; 343; 348–350; 352–354

Polygon, 332–334; 336; 338–340; 342–344; 347; 351; 353; 354; 357; 358

Surface, 332; 333; 336; 338

Ключ

внешний, 29; 37; 38; 43; 65; 66; 68; 215; 225; 253; 257; 259; 267; 272; 273; 299

Ключевое слово

INTERVAL, 90; 136; 148; 171–174; 180

DAY, 90; 91; 171–174; 176; 180; 181

DAY\_HOUR, 90; 173

DAY\_MICROSECOND, 90; 172; 173

DAY\_MINUTE, 90; 173; 176

- DAY\_SECOND, 90; 173; 174  
 HOUR, 90; 91; 133; 172–174; 177; 178;  
 181; 284; 288  
 HOUR\_MICROSECOND, 90; 172; 173  
 HOUR\_MINUTE, 90; 173  
 HOUR\_SECOND, 90; 173  
 MICROSECOND, 172; 173; 176; 178  
 MINUTE, 91; 172–174; 179; 181  
 MINUTE\_MICROSECOND, 91; 172;  
 173  
 MINUTE\_SECOND, 91; 173; 174  
 MONTH, 91; 173; 174; 179; 181  
 QUARTER, 91; 172; 173; 179; 181  
 SECOND, 90; 91; 133; 172–174; 179;  
 181  
 SECOND\_MICROSECOND, 91; 172–  
 174  
 WEEK, 91; 172; 173; 181; 183; 184; 185  
 YEAR, 20; 25; 91; 123; 128; 129; 135;  
 136; 142; 173; 174; 176; 181; 184  
 YEAR\_MONTH, 91; 173; 174; 176
- Компания MySQL AB  
<http://www.mysql.com>, 15; 18; 27–36; 50  
 коммерческая лицензия, 29; 30  
 поддержка, 21; 24; 28; 31; 40; 41; 43;  
 56; 57; 66; 92; 108; 232  
 сертификация, 28
- Консорциум Open GIS Consortium, 331
- Конструкция  
 CHANGE имя\_столбца, 59; 255  
 COLLATE, 76; 90; 96–98; 101–108; 124;  
 200; 253; 254; 260; 263; 264  
 DELAYED, 46; 59; 63; 71; 90; 217–223;  
 231; 281; 309; 318; 375; 390  
 DROP INDEX, 59; 254–256; 275; 279;  
 285; 356  
 DROP имя\_столбца, 59; 255  
 GROUP BY, 21; 44–46; 60; 61; 73; 84;  
 102; 138; 139; 206–212; 232–235;  
 237; 240; 242; 247; 267; 271; 308;  
 318; 383; 389; 405; 408; 409  
 модификатор  
 LIMIT, 39; 46; 47; 59; 84; 91; 169;  
 201; 202; 211; 213–217; 219; 232;  
 235; 237; 238; 241; 242; 247; 248;  
 251–253; 296; 301; 304; 313; 320;  
 364; 377; 407  
 WITH ROLLUP, 209–211; 232; 235  
 IF EXISTS, 17; 25; 59; 275; 276; 313; 363
- IGNORE, 44; 58; 59; 68; 72; 84; 90; 144;  
 213; 214; 217–219; 221; 224; 225;  
 227; 233; 238–240; 251–253; 255;  
 258; 263; 272; 318; 319; 357; 358; 410  
 LIKE, 40; 55; 60; 73; 88; 91; 94; 107;  
 109; 140; 141; 151; 160–163; 261;  
 270; 272; 296–298; 301–303; 309–  
 314; 363; 402; 406  
 LIMIT, 39; 46; 47; 59; 84; 91; 169; 201;  
 202; 211; 213–217; 219; 232; 235;  
 237; 238; 241; 242; 247; 248; 251;  
 252; 253; 296; 301; 304; 313; 320;  
 364; 377; 407  
 LOW\_PRIORITY, 59; 91; 213; 214; 217–  
 219; 221; 224; 231; 251; 252; 280; 281  
 NOT LIKE, 163  
 NOT REGEXP, 60; 163  
 NOT RLIKE, 163  
 ORDER BY, 21; 46; 59; 72–74; 84; 102;  
 103; 105; 136; 138; 144; 169; 186;  
 194; 207; 208; 211–215; 232–234;  
 240–242; 245; 247; 248; 251–253;  
 254; 256; 267; 308; 318; 405  
 REGEXP, 60; 91; 163; 164; 374; 413–416  
 RLIKE, 91; 163; 164  
 SQL\_SMALL\_RESULT, 59; 91; 232; 237  
 WHERE, 20; 21; 58; 61; 64; 66; 70; 74;  
 75; 80; 82; 91; 102; 103; 110; 139;  
 141; 144; 145; 147; 157; 169; 171;  
 185–190; 198; 201; 207; 212–216;  
 219; 221; 232–235; 238–253; 264;  
 277; 278; 282; 283; 287; 291; 300;  
 356–358; 386; 391; 403–407; 410
- Контрольная сумма MD5, 61  
 Курсор, 42; 364; 367
- Л**
- Лицензия GNU General Public License, 15;  
 18; 32
- М**
- Механизм хранения  
 BDB, 23; 62; 71  
 HEAP, 40; 70; 73  
 InnoDB, 23; 24; 29; 37; 38; 43; 62; 64;  
 65; 68; 72  
 MEMORY, 42  
 MERGE, 25; 42; 72; 75

MyISAM, 20; 22–25; 37; 38; 40; 42; 45;  
46; 59; 62; 64; 65; 68; 72

## Н

Недействительный результат чтения, 63

## О

### Оболочка

Bourne, 17

sh, 16

tcsh, 17

### Оператор

ALTER DATABASE, 96; 253

ALTER FUNCTION, 362; 363

ALTER PROCEDURE, 362; 363

ALTER TABLE, 24; 45; 46; 59; 71; 75;  
96; 114; 124; 128; 133; 185; 192; 193;  
222; 225; 253–260; 272; 274; 275;  
279; 282; 285; 294; 299; 309; 313;  
314; 315; 318; 344; 356; 385; 410–412

ANALYZE TABLE, 59; 71; 72; 192;  
291; 294; 305; 309; 410

BACKUP TABLE, 291; 292; 296

BEGIN, 278; 279; 299; 362–369

CACHE INDEX, 315; 316; 318

CALL, 90; 360; 362; 363; 366; 369

CHANGE MASTER TO, 321–323; 392

CHECK TABLE, 55; 59; 292; 293

CHECKSUM TABLE, 269; 294

COMMIT, 62; 71; 251; 278–280; 299;  
361; 375; 391

CREATE DATABASE, 59; 83; 96; 98;  
108; 260; 302; 393

CREATE FUNCTION, 360–362

CREATE INDEX, 185; 259–261; 279;  
285; 356; 385

CREATE PROCEDURE, 360; 362; 363;  
366; 368; 369

CREATE TABLE, 24; 25; 40; 45; 53; 59;  
61; 65; 71; 80; 81; 83; 90; 96–98; 108;  
110; 113; 124; 153; 185; 193; 202;  
209; 227; 242; 246; 254; 255; 258–  
261; 263; 264; 266–268; 271–274;  
276; 277; 279; 285; 302; 303; 311;  
313; 314; 344; 345; 356; 365; 402;  
408

AVG\_ROW\_LENGTH, 24; 262; 268;  
269

MAX\_ROWS, 24; 263; 269

DECLARE, 90; 364–367; 369

DELETE, 21; 38; 42; 44; 45; 59; 65; 66;  
70; 72; 90; 144; 213–215; 231; 238;  
242; 248; 250; 251; 259; 262; 267;  
270; 272–274; 280; 283; 285–287;  
300; 318; 371; 406

DESCRIBE, 17; 90; 110; 143; 218; 254;  
264; 274; 275; 277; 302; 356

DO, 59; 205; 216; 242; 369; 375; 377

DROP DATABASE, 59; 275; 279; 393

DROP FUNCTION, 363

DROP INDEX, 59; 254–256; 275; 279;  
285; 356

DROP PROCEDURE, 363

DROP TABLE, 17; 25; 59; 70; 256; 260;  
276; 279; 285; 313; 327; 329; 385

DROP USER, 283; 287

EXPLAIN, 21; 52; 53; 59

EXPLAIN SELECT, 52; 53; 59; 357; 410

FLUSH, 53; 59; 71; 75; 198; 222; 223;  
282; 283; 285; 287; 290; 291; 308;  
309; 316; 317; 319; 320; 325; 375;  
376; 391

GRANT, 90; 283–290; 305; 361; 374;  
389

HANDLER, 57; 216; 217; 365; 366; 377;  
378; 380

INSERT, 17; 21; 25; 38; 41; 43; 45; 46;  
59; 61; 63; 65; 68; 69; 71; 72; 90; 106;  
110; 123; 128; 132; 156; 157; 185;  
196; 202; 214; 217–223; 230; 231;  
242; 247; 250; 255; 258; 261; 263;  
270; 277; 280; 281; 285; 286; 288;  
290; 294; 299; 301; 309; 313; 314;  
318; 344; 345; 366; 367; 373; 375;  
390; 399; 403; 404; 408; 411

KILL, 70; 90; 223; 281; 285; 317; 318; 373

LOAD DATA FROM MASTER, 324; 325

LOAD DATA INFILE, 44; 47; 59; 70;  
72; 80; 84; 128; 132; 139; 224–228;  
230–232; 236; 285; 313; 324; 325;  
405

LOAD INDEX INTO CACHE, 318; 319

LOAD TABLE FROM MASTER, 324

LOCK TABLES, 63; 64; 279–282; 285;  
292; 386; 390

- OPTIMIZE TABLE, 59; 71; 72; 192; 214; 294; 295; 309
- PURGE MASTER LOGS, 285; 319; 320
- RENAME DATABASE, 43
- RENAME TABLE, 42; 59; 255; 260; 276; 277; 279; 309
- REPAIR TABLE, 55; 59; 71; 72; 192; 225; 256; 295; 296; 309; 410
- REPLACE, 21; 44; 59; 70; 72; 91; 106; 159; 221; 222; 224; 225; 231; 232; 250; 263; 272
- RESET, 17; 59; 317; 319; 320; 325; 328; 391
- RESET MASTER, 320; 391
- RESET SLAVE, 325; 328
- RESTORE TABLE, 296
- REVOKE, 61; 91; 283; 284; 285; 287; 288; 290; 378; 389
- ROLLBACK, 62–64; 278–280; 299; 375; 376; 391; 405; 406
- ROLLBACK TO SAVEPOINT, 279
- SAVEPOINT, 279
- SELECT, 17; 20; 21; 26; 39; 40; 43–47; 52; 53; 58–61; 65; 66; 70–74; 76–80; 82; 84–86; 88; 89; 91; 100–107; 110; 111; 138–141; 144–190; 193–196; 198–212; 215–218; 220–223; 226–228; 231–250; 253; 263; 264; 266; 270–272; 277; 278; 280–282; 285–288; 298–301; 304; 308; 309; 313; 315; 318; 320; 324; 325; 345–354; 356–358; 360; 362; 364; 366; 369; 372; 373; 377; 379; 386; 394; 395; 397; 399; 400; 403–416
- SET AUTOCOMMIT, 278; 279; 299
- SET BIG TABLES, 299
- SET CHARACTER SET, 73; 99; 100; 299
- SET FOREIGN\_KEY\_CHECKS, 299
- SET GLOBAL
  - SQL\_SLAVE\_SKIP\_COUNTER, 325
- SET IDENTITY, 299
- SET INSERT\_ID, 299
- SET LAST\_INSERT\_ID, 299
- SET NAMES, 299
- SET PASSWORD, 287; 290; 291; 297
- SET SQL\_AUTO\_IS\_NULL, 300
- SET SQL\_BIG\_SELECTS, 300
- SET SQL\_BUFFER\_RESULT, 300
- SET SQL\_LOG\_BIN, 300; 320; 399
- SET SQL\_LOG\_OFF, 300
- SET SQL\_LOG\_UPDATE, 300
- SET SQL\_QUOTE\_SHOW\_CREATE, 300
- SET SQL\_SAFE\_UPDATES, 300
- SET SQL\_SELECT\_LIMIT, 301
- SET SQL\_WARNINGS, 301
- SET TIMESTAMP, 301
- SET TRANSACTION, 279; 282
- SET UNIQUE\_CHECKS, 301
- SHOW BINLOG EVENTS, 297; 320; 329
- SHOW CHARACTER SET, 93; 107; 114; 301
- SHOW COLLATION, 94; 107; 301; 302
- SHOW COLUMNS, 42; 45; 53; 107; 140; 141; 277; 302
- SHOW CREATE DATABASE, 107; 108; 296; 302
- SHOW CREATE FUNCTION, 361; 363
- SHOW CREATE PROCEDURE, 361; 363
- SHOW CREATE TABLE, 53; 107; 108; 259; 265; 275; 277; 296; 300; 302; 303; 363; 406
- SHOW DATABASES, 83; 285; 286; 288; 296; 303; 324
- SHOW ENGINES, 303; 406
- SHOW ERRORS, 296; 304; 313
- SHOW FULL COLUMNS, 108; 265
- SHOW FUNCTION STATUS, 363
- SHOW GRANTS, 283; 296; 304; 305
- SHOW INDEX, 291; 296; 305; 306
- SHOW INNODB STATUS, 273; 296; 306
- SHOW LOGS, 306
- SHOW MASTER LOGS, 297; 320; 321
- SHOW MASTER STATUS, 297; 321
- SHOW PRIVILEGES, 296; 306
- SHOW PROCEDURE STATUS, 363
- SHOW PROCESSLIST, 307; 308; 317; 326
- SHOW SLAVE HOSTS, 297; 321
- SHOW SLAVE STATUS, 297; 320; 325; 326; 329; 330
- SHOW STATUS, 46; 53; 223; 296; 309; 310
- SHOW TABLE STATUS, 24; 277; 296; 310; 311; 406
- SHOW TABLES, 83; 288; 311; 312; 412



SHOW VARIABLES, 53; 55; 86; 109;  
270; 298; 312–314; 406  
SHOW WARNINGS, 40; 68; 231; 296;  
304; 313–315; 363  
START SLAVE, 322; 323; 328–330; 392;  
397  
START TRANSACTION, 278; 279; 299  
STOP SLAVE, 322; 323; 325; 329; 330;  
392  
TRUNCATE TABLE, 38; 213; 251; 279  
UNLOCK TABLES, 63; 279; 280; 282;  
317  
UPDATE, 21; 38; 41; 44–46; 59; 64; 65;  
67; 68; 74; 75; 91; 123; 132; 144; 157;  
203; 217; 219; 220; 221; 230; 232;  
237; 238; 242; 248; 251–253; 259;  
262; 267; 270; 272–274; 278; 280;  
282; 285; 286; 288; 290; 291; 293;  
294; 300; 309; 313; 318; 373–375;  
377; 379; 398; 399; 403  
USE, 69; 91; 98; 191; 192; 218; 225; 233;  
238–240; 277; 295; 306; 360; 373;  
374; 377; 380; 400; 410

## Операционная система

Linux, 24  
Solaris, 24  
UNIX, 16; 21; 22; 38  
Windows, 15; 16; 21; 22; 38; 49; 51; 52;  
55; 56

## Операция

BETWEEN, 90; 147  
BINARY, 60; 90; 102; 103; 114; 124;  
125; 136; 137; 143; 150; 151; 163;  
164; 194; 262; 319; 320; 402  
IS NOT NULL, 146  
IS NULL, 146; 147; 212; 239; 250; 264;  
300; 404  
NOT BETWEEN, 147

## Опция

--ansi, 57  
AVG\_ROW\_LENGTH, 24; 262; 268; 269  
--help, 16; 22  
--log, 72; 74  
--log-bin, 72  
MAX\_ROWS, 24; 263; 269  
REAL\_AS\_FLOAT, 58; 122; 128  
--safe-recover, 295  
--sql-mode, 57; 58

--transaction-isolation, 58  
--with-charset, 95; 112  
--with-collation, 95; 112

## П

## Переменная

character\_set\_client, 99; 100; 299; 300  
character\_set\_connection, 99; 100; 101;  
105–107; 299; 300  
character\_set\_results, 99; 100; 110; 299  
character\_set\_server, 95; 96; 99  
collation\_connection, 99; 100; 101; 105–  
107  
collation\_server, 95; 96; 99  
ft\_max\_word\_len, 191; 192  
ft\_min\_word\_len, 191; 192  
key\_buffer\_size, 72; 87; 88; 316  
key\_cache\_age\_threshold, 87  
key\_cache\_block\_size, 87; 88  
key\_cache\_division\_limit, 87  
sort\_buffer\_size, 85; 86; 255; 298  
серверная  
flush, 71  
key\_buffer\_size, 72  
log, 47; 71; 74  
long\_query\_time, 45  
lower\_case\_table\_names, 55  
max\_sort\_length, 73  
sql\_mode, 57; 58  
version, 52; 54

Подзапрос, 39; 61; 66; 238; 241–244; 246–  
250; 406

Полнотекстовый поиск, 23; 185; 189; 190

Порядок сопоставления, 92; 103; 104; 112;  
113; 116

armSCII8\_bin, 117  
armSCII8\_general\_ci, 115; 117  
ascii\_bin, 115  
ascii\_general\_ci, 112; 114; 115  
big5\_bin, 119  
big5\_chinese\_ci, 93; 112; 114; 119  
cp1250\_bin, 117  
cp1250\_czech\_ci, 117  
cp1250\_general\_ci, 113; 115; 117  
cp1251\_bin, 118  
cp1251\_bulgarian\_ci, 112; 114; 118  
cp1251\_general\_ci, 118  
cp1251\_general\_cs, 118

- cp1251\_ukrainian\_ci, 113; 118
- cp1256\_bin, 117
- cp1256\_general\_ci, 115; 117
- cp1257\_bin, 118
- cp1257\_general\_ci, 115; 118
- cp1257\_lithuanian\_ci, 113; 118
- cp850\_bin, 116
- cp850\_general\_ci, 93; 112; 114; 116
- cp852\_bin, 117
- cp852\_general\_ci, 115; 117
- cp866\_bin, 118
- cp866\_general\_ci, 115; 118
- dec8\_bin, 116
- dec8\_swedish\_ci, 93; 112; 114; 116
- euckr\_bin, 119
- euckr\_korean\_ci, 113; 114; 119
- gb2312\_bin, 119
- gb2312\_chinese\_ci, 113; 115; 119
- gbk\_bin, 119
- gbk\_chinese\_ci, 113; 115; 119
- geostd8\_bin, 117
- geostd8\_general\_ci, 115; 117
- greek\_bin, 117
- greek\_general\_ci, 113; 115; 117
- hebrew\_bin, 118
- hebrew\_general\_ci, 113; 114; 118
- hp8\_bin, 116
- hp8\_english\_ci, 93; 112; 114; 116
- keybcs2\_bin, 117
- keybcs2\_general\_ci, 115; 117
- koi8r\_bin, 118
- koi8r\_general\_ci, 93; 112; 114; 118
- koi8u\_bin, 118
- koi8u\_general\_ci, 115; 118
- koi8u\_ukrainian\_ci, 113
- latin1\_bin, 94; 103; 108; 116; 124; 302
- latin1\_danish\_ci, 76; 94; 97; 98; 101; 108; 112; 116; 302
- latin1\_general\_ci, 94; 108; 116; 302
- latin1\_general\_cs, 94; 108; 116; 302
- latin1\_german1\_ci, 94; 95; 97; 98; 101; 105; 107; 112; 116; 301
- latin1\_german2\_ci, 94; 102; 105; 108; 113; 116; 302
- latin1\_spanish\_ci, 94; 108; 116; 302
- latin1\_swedish\_ci, 93–96; 98; 101; 103–105; 107; 112; 114; 116; 200; 301; 302
- latin2\_bin, 98; 104; 117
- latin2\_croatian\_ci, 113; 117
- latin2\_czech\_ci, 98; 112; 117
- latin2\_general\_ci, 93; 107; 112; 114; 117; 301
- latin2\_hungarian\_ci, 113; 117
- latin5\_bin, 118
- latin5\_turkish\_ci, 107; 113; 115; 118; 301
- latin7\_bin, 118
- latin7\_estonian\_ci, 113
- latin7\_estonian\_cs, 118
- latin7\_general\_ci, 107; 115; 118; 301
- latin7\_general\_cs, 118
- macce\_bin, 117
- macroman\_bin, 116
- macroman\_general\_ci, 115; 116
- sjis\_bin, 119
- sjis\_japanese\_ci, 112; 114; 119
- swe7\_bin, 116
- swe7\_swedish\_ci, 112; 114; 116
- tis620\_bin, 119
- tis620\_thai\_ci, 113; 114; 119
- ucs2\_bin, 115
- ucs2\_general\_ci, 115
- ucs2\_general\_uca, 115
- ujis\_bin, 119
- ujis\_japanese\_ci, 112; 114; 119
- utf8\_bin, 107; 115; 264
- utf8\_general\_ci, 104; 115; 200
- Потоки MIT-pthreads, 47; 73
- Правило, 103
- Представление, 37; 43; 66
- Привилегия, 285
- Программа
  - mysql, 15–18; 21; 25; 26; 28–34; 36–39; 41–46; 48; 50; 51; 53–56; 58; 60; 61; 64; 67; 68; 71; 72; 74
  - mysqldadmin, 16; 46; 52–54; 74; 204; 223; 260; 285; 290; 307; 310; 312; 317; 318; 388; 407
  - mysqlcheck, 22
  - mysqld, 38; 43; 46; 47; 53; 54; 55; 57; 71; 74; 81; 83–85; 88; 89; 95; 112; 138; 185; 187; 188; 191; 192; 195; 222–224; 236; 283; 290; 294; 309; 312; 316; 317; 370; 382; 387; 392; 397; 406; 410
  - ansi, 57
  - sql-mode, 57; 58; 134; 144; 195; 264

- mysqldump, 53; 54; 65; 66
- Пространственные расширения
  - функция
    - Area(), 347; 351; 352
    - AsBinary(), 346
    - AsText(), 345–347; 349–352; 356–358
    - BdMPolyFromText(), 342
    - BdMPolyFromWKB(), 343
    - BdPolyFromText(), 342
    - BdPolyFromWKB(), 343
    - Boundary(), 348
    - Buffer(), 353
    - Centroid(), 352
    - Contains(), 354
    - ConvexHull(), 353
    - Crosses(), 354
    - Difference(), 353
    - Dimension(), 347
    - Disjoint(), 354
    - Distance(), 354; 355
    - EndPoint(), 349; 350; 352
    - Envelope(), 347; 352
    - Equals(), 355
    - ExteriorRing(), 351; 352
    - GeomCollFromText(), 341; 345
    - GeomCollFromWKB(), 342
    - GeometryCollection(), 343; 352
    - GeometryN(), 352
    - GeometryType(), 347; 348
    - GeomFromText(), 341; 344–354; 357; 358
    - GeomFromWKB(), 342; 343; 345; 347
    - GLength(), 349; 350; 358
    - InteriorRingN(), 351; 352
    - Intersection(), 353
    - Intersects(), 355
    - IsClosed(), 349; 350
    - IsEmpty(), 348
    - IsRing(), 350
    - IsSimple(), 348
    - LineFromText(), 341; 347
    - LineFromWKB(), 342; 347
    - LineString(), 343; 346–350; 352
    - MBRContains(), 353; 356–358
    - MBRDisjoint(), 353
    - MBREqual(), 354
    - MBRIntersects(), 354
    - MBROverlaps(), 354
    - MBRTouches(), 354
    - MBRWithin(), 354; 356
    - MLineFromText(), 341
    - MLineFromWKB(), 342
    - MPointFromText(), 341
    - MPointFromWKB(), 342
    - MPolyFromText(), 341
    - MPolyFromWKB(), 342
    - MultiLineString(), 343; 350
    - MultiPoint(), 343
    - MultiPolygon(), 343; 351
    - NumGeometries(), 352
    - NumInteriorRings(), 351
    - NumPoints(), 349
    - Overlaps(), 355
    - Point(), 343; 348; 349; 352; 353
    - PointFromText(), 341; 345; 347
    - PointFromWKB(), 342; 347
    - PointN(), 349; 350; 352
    - PointOnSurface(), 352
    - PolyFromText(), 341
    - PolyFromWKB(), 342
    - Polygon(), 344; 351; 353; 354; 357; 358
    - Related(), 354; 355
    - SRID(), 348
    - StartPoint(), 349; 350; 352
    - SymDifference(), 353
    - Touches(), 355
    - Union(), 353
    - Within(), 355
- Протокол
  - клиент-серверный, 43
- P**
- Режим
  - ANSI, 18; 47; 56; 57; 58; 61
  - ANSI\_QUOTES, 58
  - IGNORE\_SPACE, 58
  - ONLY\_FULL\_GROUP\_BY, 58
  - PIPES\_AS\_CONCAT, 58
  - REAL\_AS\_FLOAT, 58
- Репликация, 23; 40; 71
- Репозиторий BitKeeper, 39; 41; 42
- C**
- Семафор, 46; 47
- Сервер баз данных, 18
  - встроенный, 39
- Символьный набор, 92; 94–97; 99–102; 104; 107; 109–113
  - armSCII8, 115; 117

- ascii, 112; 114; 115
- big5, 22; 93; 112; 114; 119; 402
- binary, 115; 151; 391; 395; 399
- cp1250, 113; 115; 117
- cp1251, 99; 112–114; 118
- cp1256, 115; 117
- cp1257, 113; 115; 118
- cp850, 93; 112; 114; 116
- cp852, 115; 117
- cp866, 115; 118
- Croat, 113
- Czech, 115
- danish, 112
- dec8, 93; 112; 114; 116
- dos, 112
- estonia, 113
- euc\_kr, 113
- euckr, 113; 114; 119
- gb2312, 113; 115; 119
- gbk, 113; 115; 119
- Gbk, 113; 115; 119
- geostd8, 115; 117
- german1, 112
- greek, 113; 115; 117
- Greek, 113; 115; 117
- hebrew, 113; 114; 118
- hp8, 93; 112; 114; 116
- hungarian, 113
- keybcs2, 115; 117
- koi8\_ru, 112
- koi8\_ukr, 113
- koi8r, 93; 100; 112; 114; 118
- koi8u, 113; 115; 118
- latin1, 22; 38; 76; 93–108; 110; 112–114; 116; 124; 125; 137; 151; 200; 257; 259; 301; 302
- latin1\_de, 38; 113
- latin2, 93; 98; 100; 104; 107; 112; 113; 114; 117; 301
- latin5, 107; 113; 115; 118; 301
- latin7, 107; 113; 115; 118; 301
- macroman, 115; 116
- sjis, 112; 114; 119
- swe7, 112; 114; 116
- tis620, 113; 114; 119
- ucs2, 40; 108; 109; 110; 115; 125; 137; 190; 193
- ujis, 22; 112; 114; 119
- usa7, 112
- utf8, 40; 99; 104; 106–111; 113–115; 124; 190; 193; 200; 203; 257; 264
- win1250, 113
- win1251, 113
- win1251ukr, 113
- Система управления базами данных, 18
- Система хранения
  - BDB, 23; 62; 71; 90; 207; 214; 261; 262; 264–266; 268; 278; 291; 294; 296; 304; 306; 404
  - InnoDB, 23; 24; 29; 37; 38; 43; 62; 64; 65; 68; 72; 83; 92; 207; 214–216; 251; 253; 257–259; 261; 262; 266–268; 272–274; 278; 279; 283; 291; 292; 299; 301; 304; 306; 311; 324; 404; 406; 411; 412
  - MyISAM, 20; 22–25; 37; 38; 40; 42; 45; 46; 59; 62; 64; 65; 68; 72; 83; 92; 125; 141; 185; 190; 192; 207; 214; 216; 219; 222; 224; 225; 251; 255; 256; 258; 259; 261; 263; 264; 266–271; 281; 282; 291–296; 303; 304; 314; 315; 318; 324; 325; 331; 344; 404; 406; 412
- Список рассылки MySQL, 48
  - announce, 48
  - benchmarks, 49
  - benchmarks-digest, 49
  - bugs, 48; 50; 51; 53; 54
  - bugs-digest, 48
  - gui-tools, 49
  - gui-tools-digest, 49
  - internals, 48; 250
  - internals-digest, 48
  - java, 49
  - java-digest, 49
  - mysql-mysql-modules, 50
  - mysql-mysql-modules-digest, 50
  - myodbc, 49
  - myodbc-digest, 49
  - mysql, 15–18; 21; 25–37; 39; 41–46; 48; 50; 51; 53–56; 58; 60; 61; 64; 67; 68; 71; 72; 74; 77–82; 84–86; 88–90; 93; 94; 100; 103; 104; 107–109; 111; 114; 134; 138–141; 144–174; 176–190; 192; 194–196; 198–212; 214; 218–221; 223; 225; 227; 228; 230–236;

238–241; 251; 252; 255; 258; 259;  
271; 277; 280–291; 298; 299; 301–  
303; 305; 306; 310; 312–317; 319;  
320; 322–324; 326; 344–354; 356–  
358; 360; 362; 364–366; 369; 388;  
403–405; 407–416

mysql-digest, 48

mysqldoc, 49

mysqldoc-digest, 49

packagers, 49

packagers-digest, 49

plusplus, 49; 50

plusplus-digest, 50

win32, 49

win32-digest, 49

Стандарт

SQL

1999, 18; 56

2003, 18; 42; 56; 359; 361

SQL-92, 18; 56; 250

СУБД, См. Система управления базами  
данных, 18; 19; 28; 29; 31; 33; 35; 36; 39;  
47; 49; 56; 60; 62; 63; 67; 68; 93; 143

Сценарий

configure, 46; 54; 95; 112; 270; 392

mysqlaccess, 54

mysqld\_safe, 47; 74

## T

Тип

BIT, 43; 61

DATE, 20; 25; 45; 46; 69

DATETIME, 20; 25; 44; 46; 69

SET, 20; 44; 46; 57–60; 64; 67; 69; 73–75

TIMESTAMP, 20; 25; 26; 44; 46

TINYINT, 43

Тип столбца

BIGINT, 73; 90; 121; 126; 141; 143; 153;  
156; 164–168; 195; 206; 207; 255;  
262; 265

BIT, 43; 61; 91; 121; 153; 196; 206; 207

BLOB, 20; 21; 47; 60; 73; 78–90; 125;  
136–138; 141; 142; 150; 151; 154;  
196; 197; 208; 221; 230; 236; 257;  
260–262; 265–267; 274; 294; 340;  
342; 343; 345; 346; 372; 373; 375;  
384–387; 390

BOOL, 121

BOOLEAN, 121; 185; 188; 190–192

CHAR, 20; 21; 61; 73; 90; 97; 98; 100;  
106–111; 113; 114; 122; 124; 125;  
136; 137; 142; 143; 153; 154; 157;  
162; 185; 193; 194; 196; 197; 221;  
228; 242; 246; 254; 257; 258; 260–  
262; 264; 266; 267; 269; 271; 274;  
295; 314; 315; 362; 366; 402

DATE, 20; 25; 45; 46; 69; 91; 123; 128–  
131; 135; 136; 142; 147; 171–174;  
176; 177; 179–183; 193; 194; 262;  
403; 404

DATETIME, 20; 25; 44; 46; 69; 123;  
128–133; 135; 142; 145; 147; 172;  
179; 182; 193; 194; 262; 403

DEC, 90; 93; 114; 116; 123; 126

DECIMAL, 73; 90; 122; 123; 126–128;  
141; 143; 194; 262; 273; 407; 408

DOUBLE, 20; 73; 90; 121; 122; 126–128;  
141; 143; 151; 262; 407

DOUBLE PRECISION, 122; 126; 128

ENUM, 20; 60; 69; 73; 74; 91; 124; 126;  
136; 138–140; 142; 194; 208; 231;  
262; 265; 275

FIXED, 123; 263; 270

FLOAT, 20; 90; 122; 126–128; 141; 143;  
246; 262; 407

INT, 58; 80; 81; 90; 121; 126; 128; 141;  
143; 185; 202; 209; 227; 229; 242;  
246; 258; 262; 266; 271; 273; 274;  
303; 362; 366; 368; 369; 408

INTEGER, 90; 121; 126; 141; 149; 151;  
153; 193; 255; 258; 262

LONGBLOB, 91; 125; 137; 142; 262

LONGTEXT, 91; 125; 137; 142; 262

MEDIUMBLOB, 91; 125; 142; 143; 262

MEDIUMINT, 60; 91; 121; 126; 141;  
143; 262

MEDIUMTEXT, 91; 124; 125; 137; 138;  
142; 143; 262; 274

NCHAR, 111; 124

NUMERIC, 91; 123; 126–128; 141; 262

SET, 20; 44; 46; 57–60; 64; 67; 69; 73–  
75; 81; 84–86; 88; 91; 96–100; 104;  
107–111; 114; 124; 126; 132; 134;  
136; 140–142; 144; 156; 157; 203;  
208; 217–219; 225; 231; 233; 242;  
248; 251–254; 257–260; 262–264;

272; 273; 275; 278; 279; 282; 285;  
287; 290; 291; 296–300; 302; 312;  
315; 316; 320; 323; 325; 344–346;  
349; 350–354; 362; 364; 366; 368;  
369; 371; 373; 374; 376; 386; 393;  
394; 396; 399; 402; 410  
SMALLINT, 91; 121; 126; 141; 143; 262  
TEXT, 20; 21; 44; 60; 73; 91; 97; 108;  
124; 125; 136; 137; 138; 141; 142;  
185; 221; 227; 230; 257; 260; 261;  
262; 265–267; 274; 294; 386; 387;  
390  
TIME, 20; 26; 47; 91; 123; 128; 129; 134;  
135; 142; 172; 174; 177; 178; 180–  
183; 193; 194; 262  
TIMESTAMP, 20; 25; 26; 44; 46; 89; 91;  
123; 128–133; 135; 136; 142; 145;  
172; 177; 181–183; 230; 258; 262;  
264; 265; 274; 301; 379; 398; 403;  
405  
TINYBLOB, 91; 125; 137; 142; 262  
TINYINT, 43; 91; 120; 121; 126; 141;  
143; 258; 262; 272; 314  
TINYTEXT, 91; 125; 137; 142; 262  
VARCHAR, 20; 21; 42; 61; 73; 91; 97;  
108; 109; 111; 124; 125; 136–139;  
142; 143; 154; 185; 196; 209; 221;  
228; 254; 256; 257; 260; 261; 262;  
266; 267; 269; 271; 274; 294  
YEAR, 20; 25; 91; 123; 128; 129; 135;  
136; 142; 173; 174; 176; 181; 184  
Триггер, 37; 65

## У

Уровень изоляции транзакций  
SERIALIZABLE, 58  
Утилита  
myisamchk, 22; 24; 55  
myisampack, 24; 45; 270; 275  
mysqlaccess, 54  
mysqlbinlog, 72  
replace, 67  
tar, 54

## Ф

Файловая система  
ext2, 24  
ReiserFS, 24

Формат  
WKB, 339–347  
WKT, 339; 341; 342; 344–347  
Функция  
ABS(), 90; 165; 409  
ACOS(), 165  
ADD\_TO\_SET, 44  
ADDDATE(), 171  
ADDTIME(), 171  
AES\_DECRYPT(), 196  
AES\_ENCRYPT(), 196  
ANY(), 45  
Area(), 347; 351; 352  
AsBinary(), 346  
ASCII(), 153; 158  
ASIN(), 165  
AsText(), 345–347; 349–352; 356–358  
ATAN(), 166  
ATAN2(), 166  
AVG(), 21; 206; 247  
BdMPolyFromText(), 342  
BdMPolyFromWKB(), 343  
BdPolyFromText(), 342  
BdPolyFromWKB(), 343  
BENCHMARK(), 199; 200; 249  
BIN(), 153  
BIT\_AND(), 61; 206  
BIT\_COUNT(), 61; 196  
BIT\_LENGTH(), 153  
BIT\_OR(), 61; 207  
BIT\_XOR(), 61; 207  
Boundary(), 348  
Buffer(), 353  
CASE(), 61; 90; 106; 144; 151; 153; 367;  
368; 380; 400  
CAST(), 79; 106; 107; 139; 149; 151;  
162; 193–195  
CEIL(), 166  
CEILING(), 166  
Centroid(), 352  
CHAR(), 61; 73; 98; 100; 106; 109–111;  
113; 114; 124; 125; 136; 137; 142;  
143; 153; 154; 197; 242; 246; 258;  
262; 264; 314; 362; 366; 402  
CHAR\_LENGTH(), 154; 157  
CHARACTER\_LENGTH(), 154  
CHARSET(), 200  
COALESCE, 147

- COERCIBILITY, 103; 200  
 COERCIBILITY(), 103; 200  
 COLLATION(), 104; 200  
 COMPRESS(), 154; 161  
 CONCAT(), 21; 44; 60; 61; 106; 139;  
     154; 155; 162; 194; 212; 233; 250;  
     362; 404  
 CONCAT\_WS(), 155  
 CONNECTION\_ID(), 200  
 Contains(), 354  
 CONV(), 106; 153; 155; 156; 158  
 CONVERT(), 106; 151; 193; 200  
 ConvexHull(), 353  
 COS(), 166  
 COT(), 166  
 COUNT(), 21; 38; 44; 46; 60; 207; 234;  
     242; 243; 271; 277; 304; 313; 356;  
     362; 389; 405  
 COUNT(DISTINCT), 21; 38; 60; 207  
 CRC32(), 166  
 Crosses(), 354  
 CURDATE(), 171; 172  
 CURRENT\_DATE(), 90; 130; 172; 265  
 CURRENT\_TIME(), 90; 134; 172; 398  
 CURRENT\_TIMESTAMP(), 90; 172;  
     398  
 CURRENT\_USER(), 109; 201; 203; 305  
 CURTIME(), 171; 172  
 DATABASE(), 201  
 DATE(), 172; 183  
 DATE\_ADD(), 129; 136; 171–174; 176;  
     180  
 DATE\_FORMAT(), 174; 177; 179–181  
 DATE\_SUB(), 129; 171–174; 176; 180  
 DATEDIFF(), 172  
 DAY(), 176  
 DAYNAME(), 176  
 DAYOFMONTH(), 171; 176  
 DAYOFWEEK(), 176  
 DAYOFYEAR(), 176  
 DECODE(), 61; 197  
 DEGREES(), 166  
 DES\_DECRYPT(), 197  
 DES\_ENCRYPT(), 197; 198  
 Difference(), 353  
 Dimension(), 347  
 Disjoint(), 354  
 Distance(), 354; 355  
 ELT(), 61; 73; 106; 155; 156  
 ENCODE(), 61; 197; 199  
 ENCRYPT(), 61; 198  
 EndPoint(), 349; 350; 352  
 Envelope(), 347; 352  
 Equals(), 355  
 EVERY(), 45  
 EXP(), 167  
 EXPORT\_SET(), 155  
 ExteriorRing(), 351; 352  
 EXTRACT(), 176; 194  
 FIELD(), 155; 156  
 FIND\_IN\_SET(), 141; 156  
 FLOOR(), 165; 167; 169; 212  
 FORMAT(), 61; 106; 204  
 FOUND\_ROWS(), 39; 201; 202; 237  
 FROM\_DAYS(), 61; 176  
 FROM\_UNIXTIME(), 176; 177  
 GeomCollFromText(), 341; 345  
 GeomCollFromWKB(), 342  
 GeometryCollection(), 343; 352  
 GeometryN(), 352  
 GeometryType(), 347; 348  
 GeomFromText(), 341; 344–354; 357;  
     358  
 GeomFromWKB(), 342; 343; 345; 347  
 GET\_FORMAT(), 177  
 GET\_LOCK(), 47; 204; 205; 282; 309;  
     318  
 GLength(), 349; 350; 358  
 GREATEST, 106; 147; 148  
 GROUP\_CONCAT(), 21; 41; 61; 207;  
     208; 396  
 HEX(), 79; 106; 156; 161  
 HOUR(), 133; 177; 178  
 IF(), 61; 106; 152; 367  
 IFNULL(), 152; 153; 404  
 INET\_ATON(), 204; 205  
 INET\_NTOA(), 205  
 INSERT(), 156  
 INSTR(), 106; 156  
 InteriorRingN(), 351; 352  
 Intersection(), 353  
 Intersects(), 355  
 INTERVAL(), 90; 136; 148; 171–174;  
     180  
 IS\_FREE\_LOCK(), 204; 205  
 IS\_USED\_LOCK(), 205

- IsClosed(), 349; 350
- IsEmpty(), 348
- ISNULL(), 148
- IsRing(), 350
- IsSimple(), 348
- LAST\_DAY(), 178
- LAST\_INSERT\_ID(), 60; 64; 202; 203; 220; 222; 282; 299
- LCASE(), 106; 157
- LEAST(), 106; 147-149
- LEFT(), 73; 157; 194
- LENGTH(), 154; 157; 158
- LineFromText(), 341; 347
- LineFromWKB(), 342; 347
- LineString(), 343; 346-350; 352
- LN(), 167
- LOAD\_FILE(), 157
- LOCALTIME(), 91; 178
- LOCALTIMESTAMP(), 91; 178
- LOCATE(), 156; 157; 159
- LOG(), 167
- LOG10(), 167; 168
- LOG2(), 167
- LOWER(), 106; 151; 157
- LPAD(), 158
- LTRIM(), 106; 158
- MAKE\_SET(), 158
- MAKEDATE(), 178
- MAKETIME(), 178
- MASTER\_POS\_WAIT(), 205; 325
- MATCH(), 185-191
- MBRContains(), 353; 356-358
- MBRDisjoint(), 353
- MBREqual(), 354
- MBRIntersects(), 354
- MBROverlaps(), 354
- MBRTouches(), 354
- MBRWithin(), 354; 356
- MD5(), 61; 198; 199
- MICROSECOND(), 178
- MID(), 106; 158; 184
- MINUTE(), 179
- MLineFromText(), 341
- MLineFromWKB(), 342
- MOD(), 60; 144; 168; 403
- MONTH(), 179
- MONTHNAME(), 179
- MPointFromText(), 341
- MPointFromWKB(), 342
- MPolyFromText(), 341
- MPolyFromWKB(), 342
- MultiLineString(), 343; 350
- MultiPoint(), 343
- MultiPolygon(), 343; 351
- NOW(), 45; 130; 132; 135; 171; 172; 178-180; 194; 265
- NULLIF(), 153
- NumGeometries(), 352
- NumInteriorRings(), 351
- NumPoints(), 349
- OCTET\_LENGTH(), 158
- OLD\_PASSWORD(), 198
- ORD(), 153; 158
- Overlaps(), 355
- PASSWORD(), 61; 198; 199; 287; 290; 291
- PERIOD\_ADD(), 61; 179
- PERIOD\_DIFF(), 61; 179
- PI(), 166; 168; 170
- Point(), 343; 348; 349; 352; 353
- PointFromText(), 341; 345; 347
- PointFromWKB(), 342; 347
- PointN(), 349; 350; 352
- PointOnSurface(), 352
- PolyFromText(), 341
- PolyFromWKB(), 342
- Polygon(), 344; 351; 353; 354; 357; 358
- POSITION(), 159
- POW(), 168
- POWER(), 168
- QUARTER(), 179
- QUOTE(), 159
- RADIANS(), 168
- RAND(), 168; 169; 212
- Related(), 354; 355
- RELEASE\_LOCK(), 204; 205; 216; 282
- REMOVE\_FROM\_SET, 44
- REPEAT(), 106; 154; 159; 161; 241
- REPLACE(), 106; 159
- REVERSE(), 106; 159
- RIGHT(), 106; 159
- ROUND(), 169; 408; 409
- RPAD(), 72; 73; 106; 159; 212
- RTRIM(), 106; 159
- SEC\_TO\_TIME(), 179
- SECOND(), 133; 179



SESSION\_USER(), 109; 203  
 SHA(), 199  
 SHA1(), 198; 199  
 SIGN(), 169; 170  
 SIN(), 170  
 SOME(), 45  
 SOUNDEX(), 106; 160  
 SPACE(), 106; 160  
 SQRT(), 170; 405  
 SRID(), 348  
 StartPoint(), 349; 350; 352  
 STD(), 21; 61; 208  
 STDDEV(), 208  
 STR\_TO\_DATE(), 177; 179; 180  
 STRCMP(), 152; 164; 403  
 SUBDATE(), 171; 180  
 SUBSTRING(), 106; 138; 160  
 SUBSTRING\_INDEX(), 104; 160; 203  
 SUBTIME(), 180  
 SUM(), 21; 43; 44; 60; 208–211; 247;  
     278; 282; 405; 408; 409  
 SymDifference(), 353  
 SYSDATE(), 180  
 SYSTEM\_USER(), 109; 203  
 TAN(), 170  
 TIME(), 172; 180; 183  
 TIME\_FORMAT(), 181; 182  
 TIME\_TO\_SEC(), 182  
 TIMEDIFF(), 180; 181  
 TIMESTAMP(), 123; 133; 172; 181; 183

TIMESTAMPADD(), 181  
 TIMESTAMPDIF(), 181  
 TO\_DAYS(), 61; 182  
 Touches(), 355  
 TRIM(), 61; 106; 161  
 TRUNCATE(), 169; 170  
 UCASE(), 106; 161  
 UNCOMPRESS(), 161  
 UNCOMPRESSED\_LENGTH(), 161  
 UNHEX(), 161  
 Union(), 353  
 UNIX\_TIMESTAMP(), 133; 177; 182  
 UPPER(), 106; 151; 161; 162; 243  
 USER(), 104; 109; 110; 200; 201; 203  
 UTC\_DATE(), 91; 171; 183  
 UTC\_TIME(), 91; 171; 183  
 UTC\_TIMESTAMP(), 91; 171; 183  
 UUID(), 205; 206  
 VARIANCE(), 208  
 VERSION(), 109; 203  
 WEEK(), 183; 184; 185  
 WEEKOFDAY(), 184  
 WEEKOFYEAR(), 184  
 Within(), 355  
 YEAR(), 184  
 YEARWEEK(), 184; 185

## X

Хранимая процедура, 37; 42; 65; 359; 360;  
     363

*Научно-популярное издание*

**Компания MySQL AB**

# **MySQL. Справочник по языку**

Верстка *Т.Н. Артеменко*

Художественный редактор *В.Г. Павлютин*

Издательский дом "Вильямс".  
101509, Москва, ул. Лесная, д. 43, стр. 1.

Подписано в печать 18.04.2005. Формат 70×100/16.  
Гарнитура Times. Печать офсетная.  
Усл. печ. л. 34,83. Уч.-изд. л. 24,13.  
Тираж 3000 экз. Заказ № 1544.

Отпечатано с диапозитивов в ФГУП "Печатный двор"  
Министерства РФ по делам печати,  
телерадиовещания и средств массовых коммуникаций.  
197110, Санкт-Петербург, Чкаловский пр., 15.

# MYSQL. РУКОВОДСТВО АДМИНИСТРАТОРА

**Компания MySQL AB**



[www.williamspublishing.com](http://www.williamspublishing.com)

MySQL занимает лидирующие позиции среди множества систем управления базами данных с открытым исходным кодом. Благодаря высокой производительности и простоте настройки, богатому выбору API-интерфейсов, а также функциональным средствам работы с сетями, сервер MySQL стал одним из наиболее удачных вариантов для разработки Web-приложений, взаимодействующих с базами данных.

В этой книге предложен всеобъемлющий подход к установке, обслуживанию и администрированию сервера баз данных MySQL. Являясь, фактически, официальной документацией, книга покрывает весь спектр вопросов, касающихся администрирования, а также предлагает информацию, ориентированную на опытных пользователей и администраторов.

Книга рассчитана на администраторов и разработчиков Web-приложений любой квалификации, а также на студентов и преподавателей соответствующих дисциплин.

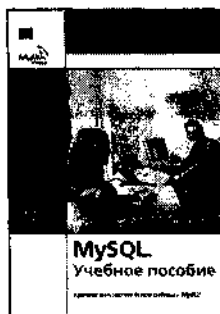
**ISBN 5-8459-0805-1**

**в продаже**

# MySQL

## Учебное пособие

Люк Веллинг и Лора Томсон



ISBN: 5-8459-0769-1

В продаже

**К**нига представляет собой краткое, но ясное изложение как основных теоретических принципов, так и практических приемов работы с MySQL. Она научит начинающего пользователя MySQL создавать сложные базы данных, которые можно использовать дома, на работе или в Web. Независимо от того, кем вы являетесь — новичком в деле освоения баз данных или профессионалом, стремящимся понять особенности работы MySQL, — это учебное пособие предоставит вам всю необходимую информацию для начала работы с MySQL и быстрого освоения этой системы.