САМОУЧИТЕЛЬ

bhv

MySQL 5

Максим Кузнецов
Игорь Симдянов

- Полный список функций MySQL 5
- Язык запросов SQL
- Хранимые процедуры
- Представления
- Безопасность
- Целостность данных

Изучение на примерах — гарантия успешного освоения MySQL 5

УДК 681.3.06
ББК 32.973.26-018.2
К89

Кузнецов М. В., Симдянов И. В.

К89 Самоучитель MySQL 5. — СПб.: БХВ-Петербург, 2006. — 560 с.: ил.

ISBN 978-5-94157-754-5

Описывается пятая версия популярной бесплатной СУБД MySQL. В начале книги происходит знакомство читателя с MySQL и простейшими SQL-запросами, такими как создание баз данных и таблиц, их заполнение, извлечение и удаление записей. Далее рассматриваются сложные вопросы SQL-программирования: встроенные функции, полнотекстовый поиск, транзакции, временные таблицы. В заключительной части разбираются различные нововведения, появившиеся только в MySQL 5: вложенные запросы, хранимые процедуры и функции, представления, триггеры, курсоры, информационные схемы и т. п.

На компакт-диске размещена учебная база данных, на примере которой на протяжении всей книги демонстрируются особенности диалекта MySQL, и дистрибутивы MySQL версии 4.0, 4.1 и 5.0 для Windows и Linux, распространяемые в соответствии с лицензией GNU/GPL.

Для программистов и Web-разработчиков

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

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

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 16.02.06.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 45,15.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

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

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

ISBN 978-5-94157-754-5

Оглавление

Введение **1**

Для кого и о чем эта книга 1
Благодарности 2

ЧАСТЬ I. **3**

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

1.1. История развития СУБД. Реляционные базы данных 5
1.1.1. Иерархические базы данных 6
1.1.2. Сетевые базы данных 7
1.1.3. Реляционные базы данных 8
1.1.4. Объектно-ориентированные и гибридные базы данных 9
1.2. Особенности реляционных баз данных 10
1.2.1. Первичные ключи 10
1.2.2. Нормализация базы данных 12
1.2.3. Правила Кодда 14
1.3. СУБД и сети 15
1.3.1. Централизованная архитектура 16
1.3.2. Архитектура "клиент-сервер" 16
1.3.2.1. Два признака клиент-серверной архитектуры 17
1.3.3. Трехуровневая архитектура Интернета 18
1.4. Как работают базы данных и что такое SQL 19
1.5. Версии MySQL 21
1.5.1. Что нового в MySQL 4.1 23
1.5.2. Что нового в MySQL 5.0 23

Глава 2. Установка MySQL 5 ..... **25**

2.1. Получение дистрибутива 25

2.2. Установка на платформу Windows 26
2.3. Установка на платформу Linux 39

Глава 3. Работа с утилитами MySQL 41

3.1. Утилита *mysql* 42
3.2. Утилита *mysqldump* 50

Глава 4. Создание баз данных и таблиц. Типы данных 53

4.1. Создание базы данных 53
4.2. Создание таблицы 57
4.3. Типы данных 59
4.3.1. Числовые данные 60
4.3.2. Строковые данные 63
4.3.3. Календарные данные 66
4.3.4. Тип данных *NULL* 69
4.3.5. Выбор типа данных 71
4.4. Учебная база данных 72

Глава 5. Индексы 77

5.1. Первичный ключ 78
5.2. Обычный и уникальный индексы 81

Глава 6. Добавление данных 87

6.1. Однострочный оператор *INSERT* 87
6.2. Многострочный оператор *INSERT* 93
6.3. Пакетная загрузка данных 94

Глава 7. Выборка данных 99

7.1. Изменение количества и порядка следования столбцов 99
7.2. Условия 101
7.3. Сортировка 104
7.4. Ограничение выборки 107
7.5. Использование функций 108
7.6. Группировка записей 111
7.7. Объединение таблиц 116
7.8. Сохранение результатов во внешний файл 119

Глава 8. Удаление данных 123

8.1. Оператор *DELETE* 123
8.2. Оператор *TRUNCATE TABLE* 124

Глава 9. Обновление данных 125

9.1. Оператор *UPDATE* 125
9.2. Оператор *REPLACE* 129

Глава 10. Редактирование структуры таблиц 131

10.1. Добавление и удаление столбцов 131
10.2. Изменение уже существующих столбцов 133
10.3. Добавление и удаление индексов 135
10.4. Преобразование таблицы 137

ЧАСТЬ II 141

Глава 11. Типы и структура таблиц 143

11.1. MyISAM 143
11.2. MERGE 144
11.3. MEMORY (HEAP) 146
11.4. EXAMPLE 147
11.5. BDB (BerkeleyDB) 148
11.6. InnoDB 149
11.7. NDB Cluster 150
11.8. ARCHIVE 150
11.9. CSV 151
11.10. FEDERATED 152
11.11. Оператор *CREATE TABLE* 154
11.11.1. Структура таблицы 155
11.11.2. Параметры таблицы 161
ENGINE (TYPE) 161
AUTO_INCREMENT 162
AVG_ROW_LENGTH 162
[DEFAULT] CHARACTER SET 162
CHECKSUM 163
COMMENT 163
DATA DIRECTORY 164
DELAY_KEY_WRITE 164
INDEX DIRECTORY 164
INSERT_METHOD 164
MAX_ROWS 164
MIN_ROWS 165
PACK_KEYS 165
PASSWORD 165
ROW_FORMAT 165
UNION 166
11.12. Изменение типа таблицы 166

Глава 12. Приведение типов 167

12.1. Ключевое слово *BINARY* 167
12.2. Функция *CAST* 168
12.3. Функция *CONVERT* 170

Глава 13. Операторы и математические функции 171

13.1. Операторы 171
13.1.1. Арифметические операторы 171
13.1.2. Операторы сравнения 175
13.1.2.1. Равенство = ... 175
13.1.2.2. Оператор <=> 176
13.1.2.3. Оператор <> 177
13.1.2.4. Оператор < 177
13.1.2.5. Оператор <= 178
13.1.2.6. Оператор > 178
13.1.2.7. Оператор >= 179
13.1.2.8. Конструкция *IS NULL* 179
13.1.2.9. Конструкция *IS NOT NULL* 180
13.1.2.10. Конструкция *BETWEEN min AND max* 181
13.1.2.11. Конструкция *NOT BETWEEN min AND max* 182
13.1.2.12. Функция *COALESCE* 182
13.1.2.13. Функция *GREATEST* 183
13.1.2.14. Функция *LEAST* 183
13.1.2.15. Конструкция *IN* 184
13.1.2.16. Конструкция *NOT IN* 185
13.1.2.17. Функция *INTERVAL* 185
13.1.3. Логические операторы 186
13.1.3.1. Оператор *NOT* 186
13.1.3.2. Оператор *OR* 187
13.1.3.3. Оператор *AND* 187
13.1.3.4. Оператор *XOR* 188
13.1.4. Битовые операторы 188
13.1.4.1. Оператор & 188
13.1.4.2. Оператор | 189
13.1.4.3. Оператор ^ 190
13.1.4.4. Оператор ~ 191
13.1.4.5. Оператор << 191
13.1.4.6. Оператор >> 192
13.1.4.7. Функция *BIT_COUNT* 193
13.1.5. Приоритет операторов 193
13.2. Математические функции 194
13.2.1. Функция *ABS* 195
13.2.2. Функция *ACOS* 195
13.2.3. Функция *ASIN* 195
13.2.4. Функция *ATAN* 196
13.2.5. Функция *ATAN2* 196
13.2.6. Функция *CEILING* 196
13.2.7. Функция *COS* 197
13.2.8. Функция *COT* 198

13.2.9. Функция *CRC32* ... 198
13.2.10. Функция *DEGREES* ... 198
13.2.11. Функция *EXP* ... 199
13.2.12. Функция *FLOOR* ... 199
13.2.13. Функция *LOG* ... 199
13.2.14. Функция *LOG2* ... 200
13.2.15. Функция *LOG10* ... 200
13.2.16. Функция *MOD* ... 200
13.2.17. Функция *PI* ... 201
13.2.18. Функция *POW* ... 201
13.2.19. Функция *RADIANS* ... 201
13.2.20. Функция *RAND* ... 202
13.2.21. Функция *ROUND* ... 203
13.2.22. Функция *SIGN* ... 204
13.2.23. Функция *SIN* ... 204
13.2.24. Функция *SQPT* ... 204
13.2.25. Функция *TAN* ... 205
13.2.26. Функция *TRUNCATE* ... 205

Глава 14. Функции даты и времени ... 207

14.1. Функция *ADDDATE* ... 207
14.2. Функция *ADDTIME* ... 212
14.3. Функция *CONVERT_TZ* ... 213
14.4. Функция *CURDATE* ... 213
14.5. Функция *CURTIME* ... 214
14.6. Функция *DATE* ... 214
14.7. Функция *DATEDIFF* ... 215
14.8. Функция *DATE_FORMAT* ... 215
14.9. Функция *DAY* ... 218
14.10. Функция *DAYNAME* ... 218
14.11. Функция *DAYOFMONTH* ... 219
14.12. Функция *DAYOFWEEK* ... 220
14.13. Функция *DAYOFYEAR* ... 220
14.14. Функция *EXTRACT* ... 220
14.15. Функция *FROM_DAYS* ... 221
14.16. Функция *FROM_UNIXTIME* ... 222
14.17. Функция *GET_FORMAT* ... 223
14.18. Функция *HOUR* ... 225
14.19. Функция *LAST_DAY* ... 225
14.20. Функция *MAKEDATE* ... 226
14.21. Функция *MAKETIME* ... 227
14.22. Функция *MICROSECOND* ... 227
14.23. Функция *MINUTE* ... 228
14.24. Функция *MONTH* ... 228

14.25. Функция *MONTHNAME* ... 228
14.26. Функция *NOW* ... 229
14.27. Функция *PERIOD_ADD* ... 230
14.28. Функция *PERIOD_DIFF* ... 230
14.29. Функция *QUARTER* ... 231
14.30. Функция *SECOND* ... 231
14.31. Функция *SEC_TO_TIME* ... 231
14.32. Функция *STR_TO_DATE* ... 232
14.33. Функция *SUBDATE* ... 232
14.34. Функция *SUBTIME* ... 233
14.35. Функция *TIME* ... 234
14.36. Функция *TIMEDIFF* ... 234
14.37. Функция *TIMESTAMP* ... 235
14.38. Функция *TIMESTAMPADD* ... 235
14.39. Функция *TIMESTAMPDIFF* ... 236
14.40. Функция *TIME_FORMAT* ... 237
14.41. Функция *TIME_TO_SEC* ... 237
14.42. Функция *TO_DAYS* ... 238
14.43. Функция *UNIX_TIMESTAMP* ... 239
14.44. Функция *UTC_DATE* ... 239
14.45. Функция *UTC_TIME* ... 240
14.46. Функция *UTC_TIMESTAMP* ... 240
14.47. Функция *WEEK* ... 241
14.48. Функция *WEEKDAY* ... 242
14.49. Функция *WEEKOFYEAR* ... 242
14.50. Функция *YEAR* ... 243
14.51. Функция *YEARWEEK* ... 243

Глава 15. Строковые функции ... 245

15.1. Функция *ASCII* ... 245
15.2. Функция *BIN* ... 246
15.3. Функция *BIT_LENGTH* ... 246
15.4. Функция *CHAR* ... 246
15.5. Функция *CHAR_LENGTH* ... 247
15.6. Функция *CHARSET* ... 248
15.7. Функция *COLLATION* ... 249
15.8. Функция *COMPRESS* ... 249
15.9. Функция *CONCAT* ... 251
15.10. Функция *CONCAT_WS* ... 252
15.11. Функция *CONV* ... 252
15.12. Функция *ELT* ... 253
15.13. Функция *EXPORT_SET* ... 253
15.14. Функция *FIELD* ... 254
15.15. Функция *FIND_IN_SET* ... 255

15.16. Функция *FORMAT* ... 255
15.17. Функция *HEX* ... 256
15.18. Функция *INSERT* ... 256
15.19. Функция *INSTR* ... 257
15.20. Функция *LEFT* ... 257
15.21. Функция *LENGTH* ... 258
15.22. Функция *LOAD_FILE* ... 258
15.23. Функция *LOCATE* ... 259
15.24. Функция *LOWER* ... 259
15.25. Функция *LPAD* ... 260
15.26. Функция *LTRIM* ... 260
15.27. Функция *MAKE_SET* ... 261
15.28. Функция *MID* ... 261
15.29. Функция *OCT* ... 262
15.30. Функция *ORD* ... 263
15.31. Функция *POSITION* ... 263
15.32. Функция *QUOTE* ... 263
15.33. Функция *REPEAT* ... 264
15.34. Функция *REPLACE* ... 264
15.35. Функция *REVERSE* ... 266
15.36. Функция *RIGHT* ... 266
15.37. Функция *RDAP* ... 267
15.38. Функция *RTRIM* ... 267
15.39. Функция *SOUNDEX* ... 268
15.40. Функция *SPACE* ... 268
15.41. Функция *SUBSTRING* ... 268
15.42. Функция *SUBSTRING_INDEX* ... 269
15.43. Функция *TRIM* ... 270
15.44. Функция *UNCOMPRESS* ... 271
15.45. Функция *UNCOMPRESSED_LENGTH* ... 271
15.46. Функция *UNHEX* ... 272
15.47. Функция *UPPER* ... 272

Глава 16. Безопасность и MySQL ... 275

16.1. Функции *AES_ENCRYPT* и *AES_DECRYPT* ... 275
16.2. Функции *ENCODE* и *DECODE* ... 277
16.3. Функции *DES_ENCRYPT* и *DES_DECRYPT* ... 278
16.4. Функция *ENCRYPT* ... 279
16.5. Функция *MD5* ... 280
16.6. Функция *PASSWORD* ... 280
16.7. Функция *SHA1* ... 281

Глава 17. Поиск и регулярные выражения ... 283

17.1. Оператор *LIKE* ... 283
17.2. Оператор *NOT LIKE* ... 287

17.3. Оператор *SOUND LIKE* 288
17.4. Оператор *RLIKE (REGEXP)* 288
17.5. Оператор *NOT RLIKE* 299
17.6. Функция *STRCMP* 300

Глава 18. Полнотекстовый поиск 301

18.1. Индекс *FULLTEXT* 301
18.2. Конструкция *MATCH (...) AGAINST (...)* 303
18.3. Логический режим 307

Глава 19. Функции, применяемые вместе с конструкцией *GROUP BY* 313

19.1. Функция *AVG()* 313
19.2. Функция *BIT_AND()* 316
19.3. Функция *BIT_OR()* 317
19.4. Функция *BIT_XOR()* 317
19.5. Функция *COUNT()* 318
19.6. Функция *GROUP_CONCAT()* 321
19.7. Функция *MIN()* 322
19.8. Функция *MAX()* 323
19.9. Функция *STD()* 324
19.10. Функция *STDDEV_SAMP()* 325
19.11. Функция *SUM()* 326
19.12. Функция *VAR_POP()* 327
19.13. Функция *VAR_SAMP()* 327
19.14. Конструкция *WITH ROLLUP* 327

Глава 20. Разные функции 329

20.1. Функции управления потоком выполнения 329
20.1.1. Функция *CASE* 329
20.1.2. Функция *IF()* 330
20.1.3. Функция *IFNULL()* 331
20.1.4. Функция *NULLIF()* 333
20.2. Информационные функции 333
20.2.1. Функция *BENCHMARK()* 333
20.2.2. Функция *CONNECTION_ID()* 334
20.2.3. Функция *CURRENT_USER()* 334
20.2.4. Функция *DATABASE()* 335
20.2.5. Функция *FOUND_ROWS()* 335
20.2.6. Функция *LAST_INSERT_ID()* 336
20.2.7. Функция *USER()* 338
20.2.8. Функция *VERSION()* 339
20.3. Разные функции 339
20.3.1. Функция *DEFAULT()* 339

20.3.2. Функция *GET_LOCK()* ... 340
20.3.3. Функция *INET_ATON()* ... 341
20.3.4. Функция *INET_NTOA()* ... 342
20.3.5. Функция *IS_FREE_LOCK()* ... 342
20.3.6. Функция *IS_USED_LOCK()* ... 343
20.3.7. Функция *RELEASE_LOCK()* ... 343
20.3.8. Функция *UUID()* ... 343

Глава 21. Переменные и временные таблицы ... 345

21.1. Переменные SQL ... 345
21.2. Временные таблицы ... 348

Глава 22. Многотабличные запросы ... 353

22.1. Перекрестное объединение таблиц ... 353
22.2. Объединение таблиц при помощи *JOIN* ... 363
22.3. Обновление нескольких таблиц ... 369
22.4. Удаление из нескольких таблиц ... 372

Глава 23. Вложенные запросы ... 375

23.1. Вложенный запрос как скалярный операнд ... 377
23.2. Вложенные запросы, возвращающие несколько строк ... 380
23.2.1. Ключевое слово *IN* ... 381
23.2.2. Ключевое слово *ANY (SOME)* ... 382
23.2.3. Ключевое слово *ALL* ... 384
23.3. Проверка на существование ... 386
23.4. Коррелированные запросы ... 388
23.5. Вложенные запросы, возвращающие несколько столбцов ... 389
23.6. Подзапросы в конструкции *FROM* ... 391
23.7. Вложенные запросы в операторе *CREATE TABLE* ... 393
23.8. Вложенные запросы в операторе *INSERT* ... 396
23.9. Ссылочная целостность ... 398

Глава 24. Транзакции и блокировки ... 407

24.1. Транзакции ... 407
24.2. Блокировка таблиц ... 412

Глава 25. Управление учетными записями пользователей ... 417

25.1. Учетные записи СУБД MySQL ... 417
25.2. Оператор *CREATE USER* ... 420
25.3. Оператор *DROP USER* ... 422
25.4. Оператор *RENAME USER* ... 424
25.5. Оператор *GRANT* ... 425
25.6. Оператор *REVOKE* ... 434

Глава 26. Предотвращение катастроф и восстановление 437

26.1. Оператор *CHECK TABLE* 437
26.2. Оператор *ANALYZE TABLE* 440
26.3. Оператор *CHECKSUM TABLE* 440
26.4. Оператор *OPTIMIZE TABLE* 443
26.5. Оператор *REPAIR TABLE* 443
26.6. Оператор *BACKUP TABLE* 445
26.7. Оператор *RESTORE TABLE* 446

Глава 27. Административные команды 449

ЧАСТЬ III 451

Глава 28. Хранимые процедуры 453

28.1. Хранимые процедуры и привилегии 454
28.2. Создание хранимой процедуры 454
28.2.1. Тело процедуры 456
28.2.2. Параметры процедуры 459
28.2.3. Работа с таблицами базы данных 462
28.2.4. Хранимые функции 469
28.3. Группа характеристик хранимых процедур 471
28.4. Операторы управления потоком данных 473
28.4.1. Оператор *IF...THEN...ELSE* 474
28.4.2. Оператор *CASE* 477
28.4.3. Оператор *WHILE* 479
28.4.4. Оператор *REPEAT* 484
28.4.5. Оператор *LOOP* 485
28.4.6. Оператор *GOTO* 486
28.5. Метаданные 488
28.5.1. Оператор *SHOW PROCEDURE STATUS* 488
28.5.2. Оператор *SHOW CREATE* 490
28.5.3. Извлечение информации из таблицы *mysql.proc* 491
28.6. Удаление хранимых процедур 493
28.7. Редактирование хранимых процедур 494
28.8. Обработчики ошибок 495
28.9. Курсоры 500

Глава 29. Триггеры 505

29.1. Оператор *CREATE TRIGGER* 505
29.2. Оператор *DROP TRIGGER* 509

Глава 30. Представления 511

30.1. Создание представлений 512
30.2. Удаление представлений 524

30.3. Редактирование представлений 525
30.4. Оператор *SHOW CREATE VIEW* 525

Глава 31. Информационная схема 527

Заключение 529

Приложение. Описание компакт-диска 531

Предметный указатель 533

ГЛАВА 1

История развития баз данных. Понятие реляционной базы данных

Компьютерные технологии постоянно эволюционируют. Это связано с тем, что программы, которые от них требуются, становятся все более и более изощренными, а чем сложнее программа, тем больше кода необходимо для ее создания. За время существования этой отрасли программисты сменили машинные коды на ассемблер, ассемблер на языки высокого уровня. Каждый новый виток в технологии программирования обеспечивал одну простую вещь — программист мог создавать более сложные программы, содержащие большее число строк. Когда мощности языков высокого уровня стало не хватать и программы достигли уровня десятков тысяч строк, появилось структурное программирование, позволившее увеличить объем программы до 100 000 строк. Очень скоро и данного предела оказалось недостаточно, и был предложен объектно-ориентированный подход, который позволил создавать еще более сложные и объемные программы.

Программы — это процессы, которые манипулируют данными. Как же происходила эволюция данных? В далеких 60-х годах прошлого столетия программисты использовали машинные коды и файлы. В настоящий момент ни прикладное, ни даже системное программное обеспечение в машинных кодах не пишется, однако файлы используются повсеместно. Значит ли это, что эволюции не было? Ответу на этот вопрос посвящена данная глава.

1.1. История развития СУБД. Реляционные базы данных

Задача длительного хранения и обработки информации появилась практически сразу с созданием первых компьютеров. Для решения этой задачи в конце 60-х годов XX века были разработаны специализированные программы, получившие название *систем управления базами данных* (СУБД). СУБД проделали длительный путь эволюции от системы управления файлами через иерархические и сетевые базы данных к реляционным моделям. В конце 80-х годов XX века доминирующей стала *система управления реляционными базами данных* (СУРБД). С этого времени такие СУБД

стали стандартом де-факто. Каждая СУБД имела свой собственный язык запросов, и для того, чтобы унифицировать работу с ними, был разработан *структурированный язык запросов* SQL (Structured Query Language), который представляет собой язык управления реляционными базами данных. Цель данного раздела — рассмотреть реляционную модель данных, сравнив ее с более ранними СУБД: иерархическими и сетевыми.

Замечание

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

Существуют следующие разновидности баз данных:

- иерархические;
- сетевые;
- реляционные;
- объектно-ориентированные;
- гибридные.

1.1.1. Иерархические базы данных

Первыми базами данных были иерархические. Иерархическая база данных основана на древовидной структуре хранения информации и в этом смысле напоминает файловую систему компьютера. Наиболее известным и распространенным примером иерархической базы данных является Information Management System (IMS) фирмы IBM, первая версия которой появилась в 1968 г. С точки зрения организации хранения информации иерархическая база данных состоит из упорядоченного набора деревьев одного типа, т. е. каждая запись в базе данных реализована виде отношений "предок-потомок". На рис. 1.1 показана схема организации такой иерархической базы данных.

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

В этой схеме Войсковая часть является предком для объектов Командир и Бойцы, а объекты Командир и Бойцы — это потомки объекта Войсковая часть, и между этими типами записи поддерживаются связи.

Реальная иерархическая база данных войсковой части может выглядеть, к примеру, так, как показано на рис. 1.2.

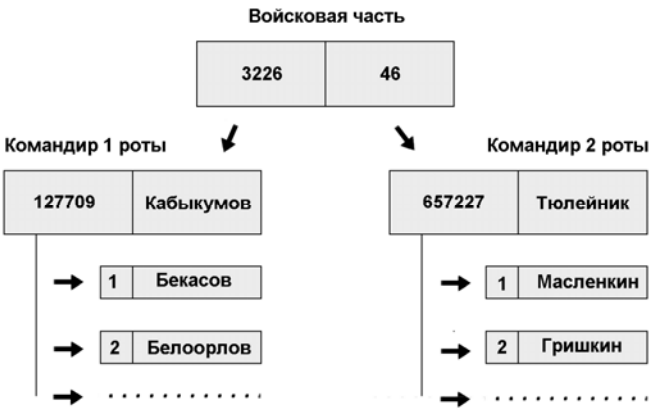

Рис. 1.2. Иерархическая база данных

Основной недостаток иерархической структуры базы данных — невозможность реализовать отношения "многие-ко-многим", а также ситуации, когда запись имеет несколько предков.

Иерархические базы данных наиболее пригодны для моделирования структур, являющихся иерархическими по своей природе. Самым простым примером такой структуры является воинское подразделение, имеющее очевидную естественную иерархию. Однако существует очень много структур, которые невозможно свести к простой иерархии, и в этом случае применение иерархической базы данных становится невозможным. Примером такой структуры является всем известное генеалогическое древо, которое, казалось бы, как нельзя лучше подходит для описания его в терминах иерархической базы данных, однако это сделать невозможно, поскольку каждый потомок имеет двух родителей.

Примечание

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

1.1.2. Сетевые базы данных

Иерархические базы данных в силу большого количества недостатков просуществовали недолго и были заменены сетевыми базами данных, являющимися, по сути,

расширением иерархических. То есть если в иерархических базах данных структура "запись-потомок" должна иметь только одного предка, то в сетевой базе данных потомок может иметь любое количество предков.

Примечание

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

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

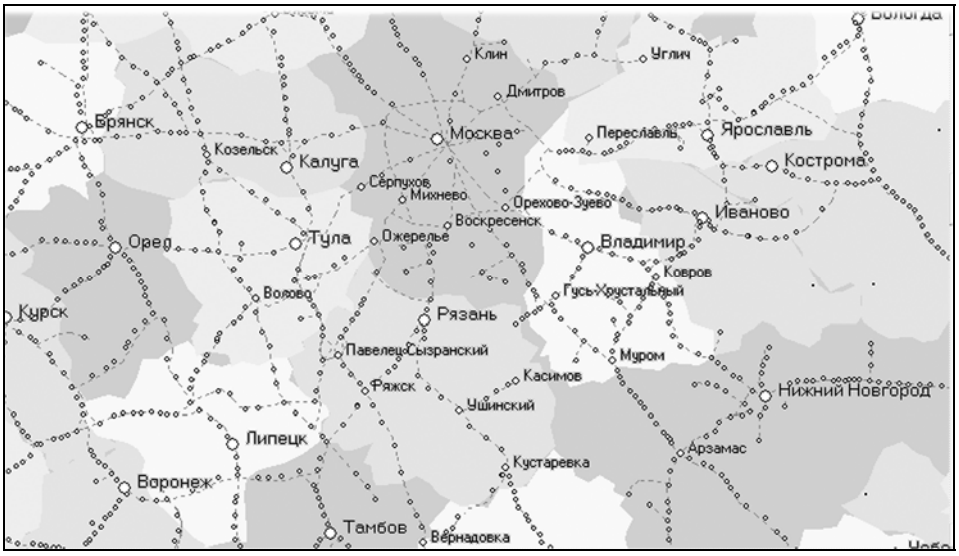

Рис. 1.3. Схема организации сетевой базы данных

Основным недостатком сетевых баз данных является сложность разработки серьезных приложений.

1.1.3. Реляционные базы данных

Настоящий прорыв в развитии теории баз данных произошел тогда, когда возросшая мощность компьютеров позволила реализовать реляционную модель данных. Теория реляционных баз данных была разработана доктором Коддом из компании IBM в 1970 году. Одной из задач реляционной модели была попытка упростить структуру базы данных. В ней отсутствовали явные указатели на предков и потомков, а все дан-

ные были представлены в виде простых таблиц, разбитых на строки и столбцы, на пересечении которых расположены данные.

Запросы к таким таблицам возвращают новые таблицы, которые сами могут становиться предметом дальнейших запросов. Даже если возвращается одно значение, это все равно считается таблицей, состоящей из одного столбца и одной строки. То, что SQL-запрос возвращает таблицу, очень важно, это означает, что результаты запроса можно записать обратно в базу данных в виде таблицы, а результаты двух или более запросов, имеющие одинаковую структуру, можно объединить в одну таблицу.

Каждая база данных может включать несколько таблиц, которые, как правило, связаны друг с другом, откуда и произошло название "реляционные".

ряд ⇩

строка ⇨

id_catalog	name	description
1	Мониторы	Информация о мониторах
2	Материнские платы	Информация о материнских платах
3	Жесткие диски	Информация о жестких дисках
4	Разное	Информация о различных комплектующих

Рис. 1.4. Таблица реляционной базы данных

На рис. 1.4 приведен пример таблицы `catalog` базы данных электронного магазина компьютерных комплектующих, которые распределены по разделам. Каждая строка этой таблицы представляет собой один вид товарных позиций, для описания которых используется поле `id_catalog` — уникальный номер раздела, `name` — название раздела и `description` — его описание.

Таким образом, можно дать следующее простое определение реляционной базы данных.

Определение

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

1.1.4. Объектно-ориентированные и гибридные базы данных

В *объектно-ориентированных* базах данных данные хранятся в виде объектов. С такими базами данных удобно работать, применяя объектно-ориентированный подход. Однако на сегодняшний день такие базы данных еще не достигли популярности реля-

ционных, поскольку пока значительно уступают им в производительности. *Гибридные (объектно-реляционные)* СУБД совмещают в себе возможности реляционных и объектно-ориентированных баз данных. В последнее время такие базы данных часто называют объектно-реляционными базами данных. Появление гибридных СУБД связано с тем, что объектные СУБД, пока что не получившие широкого применения, несомненно, будут развиваться в будущем. Поэтому разработчики реляционных баз данных включают в свои базы средства работы с объектными типами данных. Такие базы данных называются объектно-реляционными или гибридными. Примером объектно-реляционной СУБД является Oracle, бывшая ранее чисто реляционной базой данных. Возможность хранения и обработки объектов поддерживается в Oracle, начиная с восьмой версии.

1.2. Особенности реляционных баз данных

Можно кратко сформулировать особенности реляционной базы данных.

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

Более строгое определение реляционных баз данных основано на работе Кодда. В ней сформулированы 12 правил, которым должна соответствовать реляционная база данных. Сейчас эти правила так и называют: *правила Кодда* — и они являются определением понятия "реляционная база данных".

> ***Примечание***
>
> Правила Кодда и их разъяснение приведены в конце данной главы. Они не являются догматическими — реляционные базы данных постоянно развиваются и совершенствуются, на рынке появляются новые продукты, которые поддерживают не все требования к реляционным базам данных. Например, MySQL долго не соответствовала многим стандартам SQL. Исторически сложилось так, что последним стандартам SQL не соответствует ни одна из баз данных.

1.2.1. Первичные ключи

Строки в реляционной базе данных неупорядочены. То есть в таблице нет "первой", "последней", "тридцать шестой" и "сорок третьей" строки, а есть просто неупорядо-

ченный набор строк. Возникает вопрос: "Каким же образом выбирать в таблице конкретную строку?" Для этого в правильно спроектированной базе данных для каждой таблицы создается один или несколько столбцов, значения которых во всех строках различны. Такой столбец называется *первичным ключом* таблицы. Первичный ключ обычно сокращенно обозначают как PK (primary key). Никакие из двух записей таблицы не могут иметь одинаковых значений первичного ключа — т. е., благодаря наличию первичных ключей, каждая строка таблицы обладает своим уникальным идентификатором.

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

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

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

Если подходящих примеров для естественного задания первичного ключа не находится, пользуются суррогатным ключом. Суррогатный ключ представляет собой дополнительное поле в базе данных, предназначенное для обеспечения записей первичным ключом. Таким ключом, к примеру, является столбец `id_products`, базы данных товаров компьютерного магазина (см. рис. 1.4).

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

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

Например, рассмотрим базу данных какого-либо форума. В любом форуме обязательно присутствуют темы, в которых пользователи оставляют сообщения. Таким образом, в данном приближении мы имеем две таблицы: таблицу с названиями тем `themes` и таблицу `posts` с названиями сообщений. Еще раз обратим внимание на то, что строки в таблицах неупорядочены: т. е. у нас в таблице тем вперемешку содержатся строки с названиями тем, а в таблице сообщений — тексты сообщений. А как мы знаем, сообщения должны относиться к строго определенным темам. Следова-

тельно, встает задача установления взаимно однозначного соответствия между названиями тем и сообщениями в этих темах. Такое соответствие также устанавливается с помощью первичных ключей (рис. 1.5).

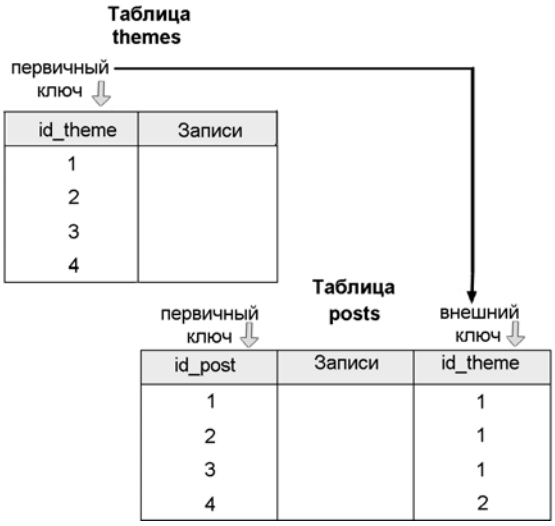

Рис. 1.5. Связь между таблицами `themes` и `posts`

Первичным ключом таблицы `themes` является `id_theme`, а таблицы `posts` — `id_post`. Обратите внимание, что поле `id_theme` присутствует и в таблице `posts`. Каждое значение этого поля в таблице `posts` является *внешним ключом* (в данном случае это внешний ключ для первичного ключа таблицы `themes`). Внешний ключ сокращенно обозначают как FK (foreign key).

Примечание

Внешний ключ также часто называют *вторичным ключом*.

Как видно из рис. 1.5, внешний ключ ссылается на первичный ключ таблицы `themes`, устанавливая однозначную логическую связь между записями таблиц `themes` и `posts`. Иначе говоря, если внешний ключ для записи (сообщения) с PK=1 в таблице `posts` имеет значение внешнего ключа равное 1, то это значит, что данное сообщение относится к теме с PK=1 таблицы `themes`.

1.2.2. Нормализация базы данных

Определение

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

Для того чтобы лучше уяснить приведенное определение нормализации, рассмотрим следующий пример. На рис. 1.6 показана таблица, в которой указаны фамилии сотрудников и их профессии.

Должность	Сотрудник
Инженер	Кузнецов
Инженер	Скворцов
Рабочий	Гришкин
Рабочий	Бекасов
Рабочий	Кабыкумов

Рис. 1.6. Пример избыточности в таблице базы данных

Эта таблица избыточна — для каждого из сотрудников повторяются одинаковые названия профессий. То есть схема такой базы данных не нормализована. Для небольшой базы данных это не критично, но в больших по объему базах это скажется на размере базы и, в конечном счете, на скорости доступа. Для нормализации необходимо разбить эту таблицу на две — профессий и фамилий сотрудников, как это показано на рис. 1.7.

Таблица должностей

Должность	Первичный ключ
"Инженер"	1
"Рабочий"	2

Таблица сотрудников

Должность (внешний ключ)	Сотрудник
1	Кузнецов
1	Скворцов
2	Гришкин
2	Бекасов
2	Кабыкумов

Рис. 1.7. Нормализованная схема соответствия профессий и сотрудников

Теперь каждый тип профессии обозначен уникальным числом, и строка в базе данных присутствует только в единственном экземпляре: в таблице профессий. Размер поля "профессия" уменьшился, т. к. строка занимает больше памяти, чем число.

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

Замечание

Хотя в теории баз данных и говорится о том, что схема базы данных должна быть полностью нормализована, в реальности при работе с полностью нормализованными базами данных необходимо применять весьма сложные SQL-запросы, что при-

водит к обратному эффекту — замедлению работы базы данных. Поэтому иногда для упрощения запросов даже прибегают к обратной процедуре — денормализации.

1.2.3. Правила Кодда

1. *Правило информации.* Вся информация в базе данных должна быть представлена только на логическом уровне и только в виде значений, содержащихся в таблицах. Это правило, по сути, является основным определением понятия "реляционная база данных".
2. *Правило гарантированного доступа.* Логический доступ к каждому элементу данных должен обеспечиваться использованием комбинации имени таблицы, первичного ключа и имени столбца. Данное правило определяет роль первичных ключей при поиске информации в базе данных и говорит о том, каким образом должен происходить этот поиск: по имени таблицы ищется нужная таблица, по имени столбца — нужный столбец, а первичный ключ позволяет найти ту строку, в которой содержится нужный элемент данных.
3. *Правило обработки отсутствующих значений.* Должна быть реализована поддержка отсутствующих значений, которые не являются ни строками нулевой длины, ни строками пробельных символов, ни нулем или каким-либо другим числом, а используются для представления отсутствующих данных, независимо от типа этих данных. Это правило говорит о том, что отсутствующие значения должны представляться значениями `NULL` (значения `NULL` описаны в *главе 4*).
4. *Правило динамического каталога.* Описание базы данных должно быть представлено в том же виде, что и описание самих данных. Это правило говорит о том, что реляционная базы данных должна сама себя описывать, т. е. содержать набор системных таблиц, описывающих саму структуру базы данных.
5. *Правило исчерпывающего подъязыка базы данных.* Реляционная база данных может поддерживать различные языки и способы взаимодействия с пользователем. Однако должен существовать по крайней мере один язык, инструкциями которого в соответствии с четко определенным синтаксисом можно описать следующие элементы:
   - определение данных;
   - обработку данных;
   - определение представлений;
   - идентификацию прав доступа;
   - условия целостности данных;
   - границы транзакций.
6. *Правило обновления представлений.* Все представления, которые теоретически возможно обновить, должны быть доступны для обновления.

Примечание

Представления, по сути, являются виртуальными таблицами, позволяющими показывать пользователям фрагменты структуры базы данных. В MySQL работа с представлениями возможна, только начиная с версии 5.0, и рассматривается в *главе 30*.

7. *Правило добавления, обновления и удаления.* Возможность работать с отношением как с одним операндом должна существовать не только при выборке данных, но и при операциях добавления, удаления и обновления данных. Это правило говорит о том, что все перечисленные операции можно было производить не только над одной строкой, а и над множествами строк.
8. *Правило физической независимости данных.* Прикладные программы для работы с данными должны оставаться нетронутыми при любых изменениях способов хранения данных или методов доступа к ним. Это правило, как и следующее, говорит о необходимости отделения пользователей и прикладных программ от низкоуровневой организации базы данных, т. е. конкретные способы реализации хранения данных и методы доступа к ним не должны влиять на возможность пользователя осуществлять работу с этими данными.
9. *Правило логической независимости данных.* Прикладные программы для работы с данными должны оставаться нетронутыми при внесении в таблицы базы данных любых изменений, которые позволяют сохранить нетронутыми содержащиеся в этих таблицах данные.
10. *Правило независимости условий целостности баз данных.* Должна существовать возможность определять условия целостности, специфичные для каждой конкретной базы данных, на подъязыке этой базы данных, и хранить их в динамическом каталоге, а не в прикладной программе. Это правило говорит о том, что язык управления базой данных должен поддерживать наложение ограничений и условий как на сами данные, так и на способы работы с ними.
11. *Правило независимости размещения.* База данных не должна зависеть от особенностей системы, на которой она расположена. Суть данного правила заключается в том, что язык управления базой данных должен обеспечивать возможность работы с распределенными данными.
12. Если в реляционной базе данных есть низкоуровневый язык, то должна отсутствовать возможность использования его для обхода правил и условий целостности данных, сформулированных на языке высокого уровня. Это правило запрещает иные средства работы с базой данных, кроме ее внутреннего языка.

Примечание

Низкоуровневым называется язык, обрабатывающий одну запись за один раз, а высокоуровневым, соответственно, язык, обрабатывающий несколько записей за один раз.

1.3. СУБД и сети

Развитие компьютерных сетей, а тем более рост популярности Интернета, сыграли серьезную роль в развитии СУБД. Если раньше и приложение, и СУБД находились на

одном центральном главном компьютере (мэйнфрейме), то теперь они отделены друг от друга: клиентская часть выполняется на рабочих машинах, а СУБД и собственно сама база данных находятся на сервере. В этой ситуации язык SQL играет важнейшую роль, являясь связующим звеном между приложением, выполняющимся на клиентских машинах, и находящейся на сервере СУБД. Рассмотрим основные этапы эволюции СУБД, связанной с развитием компьютерных сетей.

1.3.1. Централизованная архитектура

При централизованной архитектуре и приложение, и СУБД, и сама база данных размещаются на одном центральном мэйнфрейме[1], пользователи (иногда до сотни на один мэйнфрейм) подключались к нему посредством терминалов, как это показано на рис. 1.8. Сам терминал представлял собой клавиатуру, монитор и сетевую карту, посредством которой происходил обмен данных терминала с мэйнфреймом. Роль приложения в данном случае состоит в принятии вводимых данных с пользовательского терминала (клавиатура на рисунке) и передаче их мэйнфрейму, который передавал их на обработку СУБД, полученный от СУБД ответ отправлялся на монитор терминала.

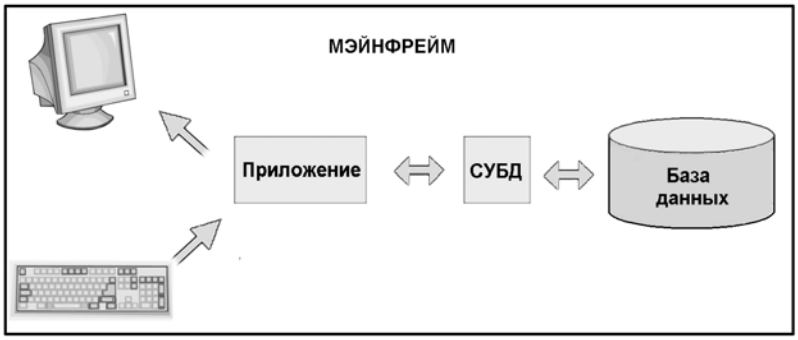

Рис. 1.8. Организация работы СУБД в централизованной архитектуре

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

1.3.2. Архитектура "клиент-сервер"

В клиент-серверной архитектуре персональные компьютеры (клиенты) объединены в локальную сеть, в этой же сети находится и сервер баз данных, на котором содержатся общие для всех клиентов базы данные и СУБД. То есть отличие данной архитектуры от централизованной состоит в том, что СУБД и база данных (находящиеся на сервере) отделены от многочисленных приложений (выполняющихся на клиентских машинах). Схема организации архитектуры "клиент-сервер" изображена на рис. 1.9.

---

[1] От англ. mainframe — базовая универсальная вычислительная машина. — *Ред.*

Замечание

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

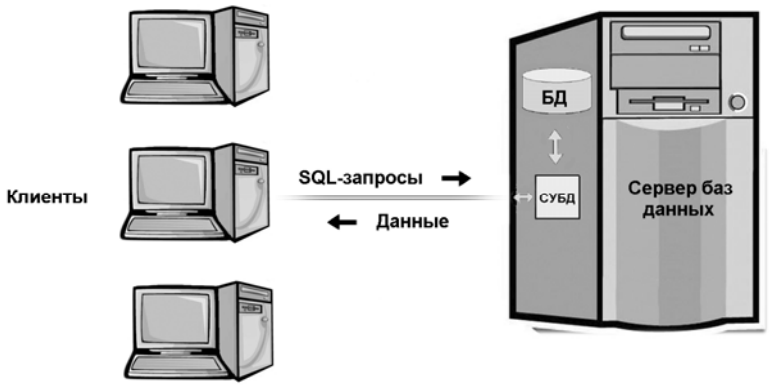

Рис. 1.9. Место СУБД в архитектуре "клиент-сервер"

О клиентах и серверах

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

1.3.2.1. Два признака клиент-серверной архитектуры

Во-первых, это то, что приложения и СУБД выполняются на разных компьютерах. Этим признаком клиент-серверная архитектура отличается от централизованной ар-

хитектуры. А во-вторых, это то, что клиент и сервер находятся в одной локальной сети. Это достаточно важный признак, о котором часто забывают. Между тем, в том числе и по этому признаку клиент-серверная архитектура отличается от трехуровневой архитектуры Интернета, которая рассмотрена ниже и в которой клиенты и серверы объединены не в локальной, а уже в глобальной сети.

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

1.3.3. Трехуровневая архитектура Интернета

С развитием Интернета архитектура сетевого управления базами данных получила дальнейшее развитие. Ранний Интернет обеспечивал доступ к базам данных напрямую. Через некоторое время появилась среда Web, которая предоставила пользователям Интернета дружественный интерфейс. За формирование этого интерфейса несет ответственность Web-сервер, т. е. на пути между пользователем-клиентом и базой данных появился еще один посредник, который стал выполнять функции клиентской программы. Такой подход позволяет многочисленным пользователям Интернета иметь единственную программу Web-браузер для работы с различными базами данных, будь это интернет-магазин или форум, не прибегая к услугам специфических программ-клиентов.

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

Схема работы трехуровневой архитектуры следующая. Допустим, клиенты какой-либо компании, находясь за тысячи километров от нее, желают ознакомиться со списком товаров, доступных на данный момент. В этом случае они используют браузер для посещения сайта этой компании. Страницу со списком товаров при этом формирует специальный модуль — скрипт, выполняющийся на Web-сервере компании. Для получения нужной информации этот скрипт обращается (посылает SQL-запросы) к СУБД, находящейся на сервере баз данных. Иллюстративно такая схема взаимодействия изображена на рис. 1.10.

Таким образом, в трехуровневой архитектуре Интернета интерфейсом пользователя является Web-браузер или другой "тонкий клиент". Это — клиентский уровень. Браузер взаимодействует с Web-сервером, посылая ему запросы на отображение той или иной Web-страницы. Уровень Web-сервера — прикладной уровень. Web-приложение, выполняющееся на Web-сервере, формирует SQL-запрос к СУБД, которая в свою очередь возвращает необходимые данные из базы данных. СУБД и база данных при

этом размещены на сервере баз данных и представляют собой третий, информационный уровень трехуровневой архитектуры Интернета.

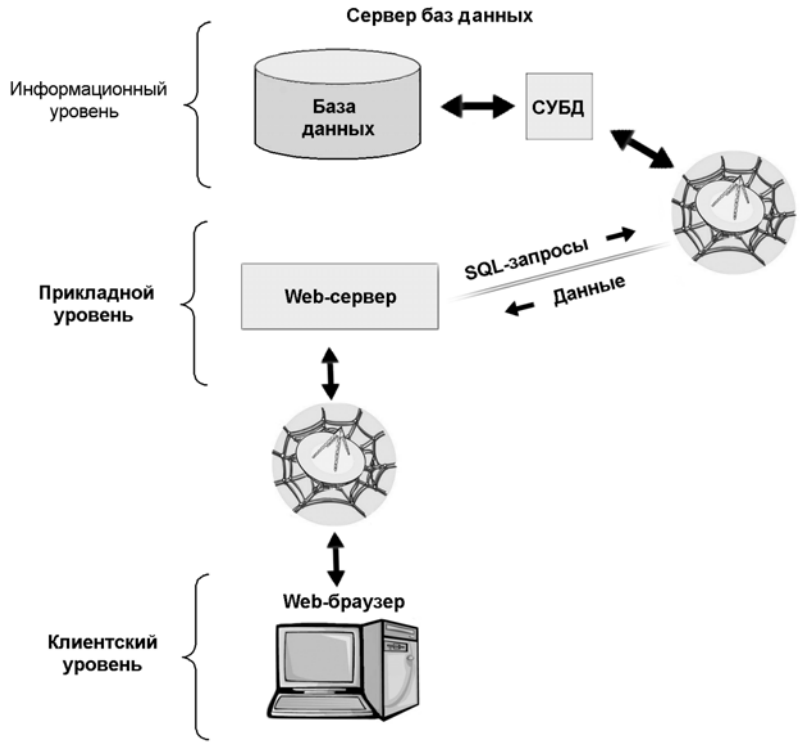

Рис. 1.10. Роль СУБД в трехуровневой архитектуре Интернета

1.4. Как работают базы данных и что такое SQL

Рассмотрим упрощенную схему работы пользователя с базой данных, показанную на рис. 1.11. Согласно этой схеме, в системе имеется база данных, в которой хранится некоторая информация, допустим, информация о работниках. По сути, база данных — это те же файлы, в которых хранится информация. Сами по себе базы данных не представляли бы никакого интереса, если бы не было систем управления базами данных, сокращенно — СУБД. СУБД — это программный комплекс, который управляет базой данных, т. е. берет на себя все низкоуровневые операции по работе с файлами, благодаря чему программисту при работе с базой данных нужно оперировать лишь логическими понятиями при помощи языка программирования, предназначенного для организации взаимодействия пользователя с базой данных.

Примечание

Итак, еще раз уточним понятия. База данных — это просто файловое хранилище информации, и не более. А программные продукты типа MySQL, Oracle, Dbase, Informix, PostgreSQL и др. — это системы управления базами данных. Ведь по сути базы данных везде одинаковы — это файлы с записанной в них информацией. Все вышеприведенные программные продукты отличаются друг от друга именно способом организации работы с файловой системой. Однако для краткости эти СУБД часто называют просто базами данных. Так будем поступать и мы. Следует помнить, что когда мы будем говорить "база данных" — речь идет именно о СУБД.

Рис. 1.11. Простейшая схема организации работы с базой данных

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

Замечание

После публикации исследований Кодда компанией IBM был инициирован проект System/R по созданию первой реляционной базы данных, результатом которого стало создание минимального прототипа реляционной СУБД. Кроме разработки самой СУБД, проводилась работа над созданием языков запроса к базе данных. Один из этих языков был назван SEQUEL (Structured English Query Language, структурированный английский язык запросов). Позже по юридическим соображениям язык был переименован в SQL.

Таким образом, для того чтобы получить какую-либо информацию из базы данных, необходимо направить базе данных запрос, созданный с использованием SQL, результатом выполнения которого будет таблица с данными (см. рис. 1.11).

Несмотря на то, что SQL называется "языком запросов", в настоящее время этот язык представляет собой нечто большее, чем просто инструмент для создания запросов. Ведь запрос это, по сути, предложение вида "Выбрать из таких-то таблиц такие то данные". К примеру, на рис. 1.11 показан запрос, суть которого состоит в выборке всех фамилий из таблицы работников. С помощью SQL осуществляется реализация всех возможностей, которые представляются пользователям разработчиками СУБД, а именно:

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

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

> ***Примечание***
>
> Несмотря на то, что SQL называют языком программирования баз данных, на самом деле это не совсем корректно, т. к. SQL работает только с базами данных определенного типа — реляционными базами данных. Однако не будет ошибкой сказать, что SQL на сегодняшний день это единственный язык программирования баз данных — по той причине, что реляционные базы данных фактически являются единственными типами баз данных, используемыми в настоящее время.

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

1.5. Версии MySQL

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