

Линн Бейли

Изучаем SQL



Приведи
в порядок
свои
отношения
с данными



Прекрати путать
первичные
и внешние ключи



Будь готов
объяснить суть
нормализованной
таблицы



Освой концепцию
и синтаксис SQL
максимально
эффективно



Перестань
смущаться
команды
ALTER



Проверь свои
знания SQL
на интересных
упражнениях

O'REILLY®

ПИТЕР®

Head First SQL

Lynn Beighley

Wouldn't it be dreamy
if there was a book that
could teach me SQL without making
me want to relocate to a remote island
in the Pacific where there are
no databases? It's probably
nothing but a fantasy...



O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Изучаем SQL

Линн Бейли

Как хорошо было бы найти книгу по SQL, от которой бы мне не хотелось уплыть на необитаемый остров без единой базы данных... Наверное, об этом можно только мечтать...



 **ПИТЕР®**

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск

2012

ББК 32.973.2-018.1
УДК 004.434
Б41

Бейли Л.
Б41 Изучаем SQL. — СПб.: Питер, 2012. — 592 с.: ил.

ISBN 978-5-459-00421-2

В современном мире наивысшую ценность имеет информация, но не менее важно уметь этой информацией управлять. Эта книга посвящена языку запросов SQL и управлению базами данных. Материал излагается, начиная с описания базовых запросов и заканчивая сложными манипуляциями с помощью объединений, подзапросов и транзакций. Если вы пытаетесь разобраться в организации и управлении базами данных, эта книга будет отличным практическим пособием и предоставит вам все необходимые инструменты. Особенностью данного издания является уникальный способ подачи материала, выделяющий серию «Head First» издательства O'Reilly в ряду множества скучных книг, посвященных программированию.

ББК 32.973.2-018.1
УДК 004.434

Права на издание получены по соглашению с O'Reilly. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

ISBN 978-0596526849 англ.

© Authorized Russian translation of the English edition of Head SQL © O'Reilly Media, Inc. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

ISBN 978-5-459-00421-2

© Перевод на русский язык
ООО Издательство «Питер», 2012
© Издание на русском языке, оформление
ООО Издательство «Питер», 2012

Посвящается нашему миру, изобилующему данными.
И вам, желающим управлять этими данными.



↑ Линн Бейли

Линн — писатель-беллетрист, занявшийся написанием технической литературы. Когда обнаружилось, что за технические книги неплохо платят, она приняла свое новое призвание и стала получать удовольствие от него.

После получения ученой степени магистра в области компьютерных технологий Линн работала в лабораториях NRL и LANL. Затем она открыла для себя Flash и написала свой первый бестселлер.

Ей не повезло с выбором момента — она переехала в Кремниевую Долину как раз перед крахом. Линн несколько лет проработала в Yahoo!, писала другие книги и разрабатывала учебные курсы. Наконец, поддавшись своим творческим наклонностям, она переехала в Нью-Йорк для получения степени магистра искусств в области писательского мастерства.

Ее дипломная работа, написанная в стиле серии Head First, была представлена в аудитории, плотно забитой профессорами и однокурсниками. Работа была исключительно хорошо принята, Линн получила ученую степень, закончила работу над книгой и не может дождаться начала работы над следующей книгой.

Линн любит путешествия, любит готовить и выдумывать подробные истории о совершенно незнакомых людях. Немного побаивается клоунов.



SQL? Почему бы не назвать книгу Head First SQL?

↑ Вид из окна Линн.

	Введение	25
1	Данные и таблицы: <i>Всему свое место</i>	37
2	Команда SELECT: <i>Выборка данных</i>	87
3	DELETE и UPDATE: <i>О пользе изменений</i>	153
4	Проектирование таблиц: <i>Как важно быть нормальным</i>	193
5	ALTER: <i>Как изменить прошлое</i>	231
6	Расширенные возможности SELECT: <i>Взглянуть на данные под другим углом</i>	267
7	Многотабличные базы данных: <i>Когда в одной таблице тесно</i>	311
8	Соединения и многотабличные операции: <i>Не могли бы мы остаться в одиночестве?</i>	373
9	Подзапросы: <i>Запросы внутри запросов</i>	409
10	Внешние соединения, самосоединения и союзы: <i>Новые приемы</i>	445
11	Ограничения, представления и транзакции: <i>У семи нянек</i>	481
12	Безопасность: <i>Защита данных</i>	519
	Приложение I: Прочее	549
	Приложение II: Установка MySQL	567
	Приложение III: Список инструментов	575

Содержание (настоящее)

Введение

Настройте свой мозг на SQL. Вот что вам понадобится, когда вы пытаетесь что-то выучить, в то время как ваш мозг не хочет воспринимать информацию. Ваш мозг считает: «Лучше уж я подумаю о более важных вещах, например об опасных диких животных или почему нельзя голышом прокатиться на сноуборде». Как же заставить свой мозг думать, что ваша жизнь зависит от овладения SQL?

Для кого написана эта книга?	26
Мы знаем, о чем вы думаете	27
Метапознание: наука о мышлении	29
Заставить свой мозг повиноваться	31
Примите к сведению	32
Технические рецензенты	34
Благодарности	35

1

Всему свое место

Разве не обидно потерять? Что угодно — ключи от машины, купон на скидку в 25%, данные приложения... Нет ничего хуже, чем невозможность **найти то, что вам нужно...** и именно тогда, когда нужно. А в том, что касается приложений, для хранения важной информации не найти места лучше, чем **таблица**. Так что переверните страницу и присоединяйтесь к нашей прогулке по миру **реляционных баз данных**.

Определение данных	38
Рассматриваем данные по категориям	43
Что такое «база данных»?	44
Посмотрим на базу данных через волшебные очки...	46
В базах данных хранится логически связанная информация	48
Таблицы под увеличительным стеклом	49
Командуйте!	53
Создание таблицы: команда CREATE TABLE	55
Создание более сложных таблиц	56
Посмотрите, как просто пишется код SQL	57
Наконец-то создаем таблицу my_contacts	58
Таблица готова	59
Знакомство с типами данных	60
Описание таблицы	64
Нельзя заново создать существующую таблицу или базу данных!	66
Долой старые таблицы!	68
Для добавления данных в таблицу используется команда INSERT	70
Создание команды INSERT	73
Модификации команды INSERT	77
Столбцы без значений	78
Команда SELECT читает данные из таблицы	79
Управление NULL в таблицах	81
NOT NULL в выходных данных DESC	83
DEFAULT и значения по умолчанию	84
Новые инструменты	85

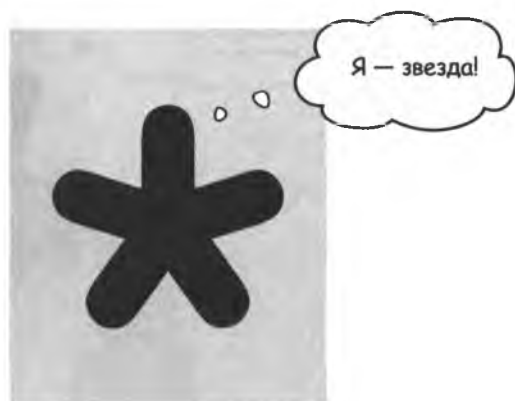


2 Команда SELECT

Выборка данных

При работе с базами данных операция выборки обычно выполняется чаще, чем операция вставки данных в базу. В этой главе вы познакомитесь с могущественной командой **SELECT** и узнаете, как получить доступ к важной информации, которую вы сохранили в своих таблицах. Также вы научитесь использовать условия **WHERE**, **AND** и **OR** для избирательной выборки данных и предотвращения вывода ненужных данных.

Трудный поиск	88
Улучшенная команда SELECT	91
Что это за * ?	92
Как запрашивать разные типы данных	98
Проблемы со знаками препинания	99
Непарный апостроф	100
Апострофы как специальные символы	101
Команда INSERT с внутренним апострофом	102
Выборка ограниченного набора столбцов	105
Отбор столбцов ускоряет получение результатов	107
Объединение условий	114
Поиск числовых значений	117
Операторы сравнения	120
Операторы сравнения при поиске числовых данных	122
Операторы сравнения при поиске текстовых данных	125
Быть ИЛИ не быть	127
Чем AND отличается от OR	130
Использование IS NULL для поиска NULL	133
LIKE: слово для экономии времени	135
Специальные символы	135
Проверка диапазонов с использованием AND и операторов сравнения	139
Только МЕЖДУ нами... Есть и другой способ	140
Условие IN	143
Ключевые слова NOT IN	144
Другие применения NOT	145
Новые инструменты	150



3

DELETE и UPDATE

О пользе изменений

Никак не можете прийти к окончательному решению? И ладно!

Команды, с которыми вы познакомитесь в этой главе — **DELETE** и **UPDATE**, — избавят вас от возни с данными, которые вы ввели полгода назад. Команда **UPDATE** изменяет данные, а команда **DELETE** удаляет из таблицы данные, которые вам больше не нужны. Но мы не только рассмотрим новые инструменты; в этой главе вы узнаете, как избирательно применять новые возможности и как предотвратить случайное удаление полезных данных.

Клоуны вокруг нас	154
Информация о клоунах	155
Перемещения клоунов	156
Как вводятся сведения о клоунах	160
Бонзо, у нас проблема	162
Уничтожение записей командой DELETE	163
Использование команды DELETE	165
Правила DELETE	166
Танцы INSERT-DELETE	169
Будьте внимательны при выполнении DELETE	174
Проблемы с неточными условиями DELETE	178
Изменение данных командой UPDATE	180
Правила UPDATE	181
UPDATE как замена INSERT-DELETE	182
UPDATE в действии	183
Обновление цен	188
Достаточно одной команды UPDATE	190
Новые инструменты	192



Не ждали?

4

Как важно быть нормальным

До настоящего момента мы не особо задумывались при создании таблиц. Работают — и ладно; в конце концов, с ними можно выполнять команды `SELECT`, `INSERT`, `DELETE` и `UPDATE`. Но при увеличении объема данных постепенно становится ясно, что следовало бы сделать при создании таблицы для упрощения условий `WHERE`: ее следовало бы сделать более нормальной.

Две таблицы	194
Логические связи как суть таблицы	198
Атомарные данные	202
Атомарные данные и таблицы	204
О пользе нормализации	208
Преимущества нормализованных таблиц	209
Ненормализованные клоуны	210
На полпути к 1НФ	211
Правила первичных ключей	212
Как прийти в НОРМУ	215
Исправление таблицы Грега	216
Старая команда <code>CREATE TABLE</code>	217
Сначала покажи деньги таблицу	218
Команда для экономии времени	219
Команда <code>CREATE TABLE</code> с назначением первичного ключа	220
1, 2, 3 и так далее	222
Добавление первичного ключа в существующую таблицу	226
<code>ALTER TABLE</code> и добавление первичного ключа	227
Новые инструменты	228

Минутку, у меня полная таблица данных. И вы хотите, чтобы я удалил ее командой `DROP TABLE`, как в главе 1, и ввел все данные снова — только для того, чтобы создать первичный ключ в каждой записи?



5

Как изменить прошлое**Вам никогда не хотелось исправить прошлые ошибки?**

Что же, теперь у вас есть такая возможность. Благодаря команде **ALTER** вы сможете применить свои новые знания к таблицам, созданным много дней, месяцев и даже лет назад. А самое замечательное, что это никак не повредит существующим данным! К настоящему моменту вы уже знаете, что означает понятие **нормализованный**, и можете применять его ко всем таблицам — как прошлым, так и будущим.

ТАБЛИЦА НА ПРОКАЧКУ

Пришло время превратить вашу старую, скучную таблицу в настоящую «бомбу». Вы и не подозревали, что такие превращения возможны!



Нужно внести пару изменений	232
Изменение таблиц	237
Капитальный ремонт таблицы	238
Переименование таблицы	239
Грандиозные планы	241
Перепланировка столбцов	242
Структурные изменения	243
ALTER и CHANGE	244
Изменение двух столбцов одной командой SQL	245
Стоп! Никаких лишних столбцов!	249
Неатомарный столбец location	256
В поисках закономерности	257
Удобные строковые функции	258
Заполнение нового столбца существующими данными	263
Как работает комбинация UPDATE с SET	264
Новые инструменты	266

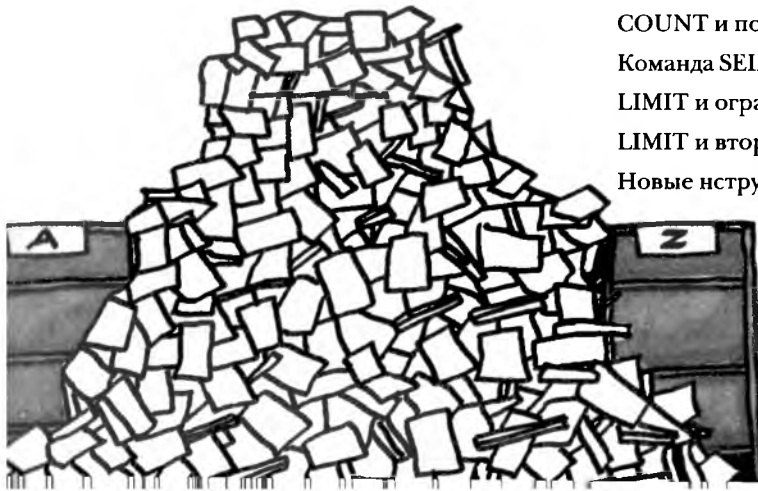
6

Взглянуть на данные под другим углом

Пора обзавестись более точными инструментами. Вы уже знаете, как выполнять выборку данных, и умеете работать с условиями WHERE. Но в некоторых ситуациях нужна **точность**, на которую SELECT и WHERE не способны. В этой главе вы научитесь **упорядочивать и группировать** свои данные, а также выполнять **математические операции** с полученными результатами.

Перестройка в видеотеке	268
Недостатки существующей таблицы	269
Классификация существующих данных	270
Заполнение нового столбца	271
UPDATE с выражением CASE	274
Похоже, у нас проблемы	276
Трудности с таблицами	281
Упорядочение результатов выборки	282
ORDER BY	285
Упорядочение по одному столбцу	286
ORDER с двумя столбцами	289
ORDER с несколькими столбцами	290
Упорядоченная таблица	291
DESC и изменение порядка данных	293
Проблемы с печеньем	295
SUM сложит числа за нас	297
Суммирование с использованием GROUP BY	298
Функция AVG с GROUP BY	299
MIN и MAX	300
COUNT и подсчет дней	301
Команда SELECT DISTINCT	303
LIMIT и ограничение результатов	306
LIMIT и второе место	307
Новые инструменты	310

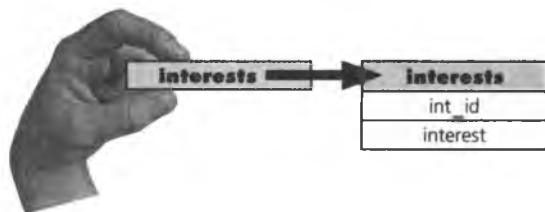
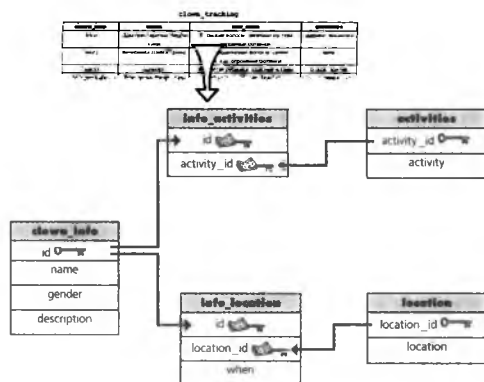
DATAVILLE Video



7

Когда в одной таблице тесно

Иногда в одной таблице становится попросту тесно. Данные стали более сложными, и с одной таблицей работать уже неудобно. Ваша единственная таблица забита избыточной информацией, которая только попусту расходует место и замедляет обработку запросов. Вы выжали из одной таблицы все, что только можно, но окружающий мир огромен, и для хранения данных и работы с ними нередко приходится использовать **несколько таблиц**.



Как найти Найджелу подружку	312
Одной таблицы недостаточно	324
Многотабличная база данных с информацией о клоунах	325
Схема базы данных clown_tracking	326
Как из одной таблицы сделать две	328
Связывание таблиц	333
Что нужно знать о внешних ключах	334
Связи между таблицами	339
Типы связей: один-к-одному	339
Когда используются таблицы со связями типа «один-к-одному»	340
Типы связей: «один-ко-многим»	341
Типы связей: «многие-ко-многим»	342
Нам нужна соединительная таблица	345
Типы связей: «многие-ко-многим»	346
Исправляем таблицу Грега	349
Наконец-то – 1НФ...	351
Составные ключи состоят из нескольких столбцов	352
Сокращенная запись	354
Супергеройские зависимости	355
Частичные функциональные зависимости	355
Транзитивные функциональные зависимости	356
Вторая нормальная форма	360
Возможно, таблица уже находится в 2НФ...	361
Третья нормальная форма (наконец-то!)	366
Что делать с таблицей my_contacts?	367
Новые инструменты	370

8 Соединения и многотабличные операции

Не могли бы мы остаться в одиночестве?

Добро пожаловать в многотабличный мир! Базы данных, состоящие из **нескольких таблиц**, удобны, но чтобы успешно работать с ними, вам придется освоить некоторые новые *инструменты и приемы*. При работе с несколькими таблицами может возникнуть путаница, поэтому вам понадобятся **псевдонимы**. А **соединения** помогут установить связь между таблицами, чтобы снова собрать воедино информацию, разбросанную по разным таблицам. Приготовьтесь, пора **снова взять базу данных под свой полный контроль!**

И все равно повторения, повторения...	374
Заполнение таблиц	375
Проблемы с нормализацией	377
Особые увлечения (столбец)	378
Разделение увлечений	379
Обновление столбцов	380
Вывод списка	381
Дороги, которые мы выбираем	382
(Почти) одновременное выполнение CREATE, SELECT и INSERT	382
Одновременное выполнение CREATE, SELECT и INSERT	383
Зачем нужно AS?	384
Псевдонимы столбцов	385
Кому нужны псевдонимы таблиц?	386
Все, что вы хотели знать о внутренних соединениях	387
Перекрестное соединение	388
Открой свое внутреннее соединение	393
Внутреннее соединение в действии: эквисоединение	394
Внутреннее соединение в действии: неэквивалентное соединение	397
Последнее внутреннее соединение: естественное соединение	398
Встроенные запросы?	405
Новые инструменты	407



...вот откуда
на самом деле берутся
таблицы результатов.

9

Запросы внутри запросов

Мне, пожалуйста, запрос из двух частей. Соединения — хорошая штука, но иногда возникает необходимость обратиться к *базе данных сразу с несколькими вопросами*. Или *взять результат одного запроса и использовать его в качестве входных данных другого запроса*. В этом вам помогут подзапросы, также называемые подчиненными запросами. Они предотвращают дублирование данных, делают запросы более динамичными и даже помогут вам попасть на вечеринку в высшем обществе. (А может, и нет — но два из трех тоже неплохо!)

Грег берет за поиски работы	410
В списке Грега появляются новые таблицы	411
Грег использует внутреннее соединение	412
Но он хочет опробовать другие запросы	414
Подзапросы	416
Два запроса преобразуются в запрос с подзапросом	417
Подзапросы: если одного запроса недостаточно	418
Подзапрос в действии	419
Правила для подзапросов	421
Построение подзапроса	424
Подзапрос как столбец SELECT	427
Другой пример: подзапрос с естественным соединением	428
Некоррелированный подзапрос	429
Некоррелированный подзапрос с несколькими значениями: IN, NOT IN	433
Коррелированные подзапросы	438
Коррелированный подзапрос с NOT EXISTS	439
EXISTS и NOT EXISTS	440
Служба поиска работы Грега принимает заказы	442
По дороге на вечеринку	443
Новые инструменты	444

ВНЕШНИЙ запрос

ВНУТРЕННИЙ запрос

```

SELECT some_column, another_column
FROM table
WHERE column = (SELECT column FROM table);
    
```

Внешний запрос

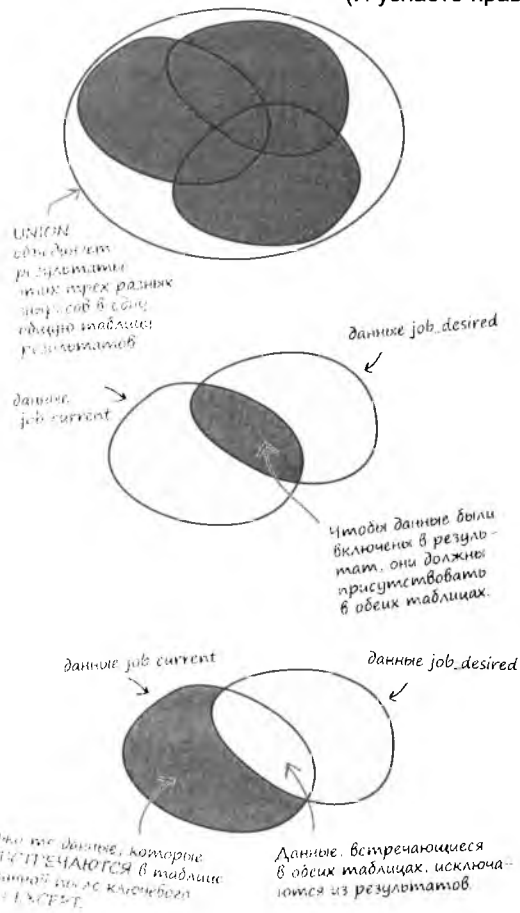
Внутренний запрос

10

Внешние соединения, самосоединения и союзы

Новые приемы

Пока вы знаете только половину того, что необходимо знать о соединениях. Вы видели перекрестные соединения, которые возвращают все возможные комбинации записей, и внутренние соединения, которые возвращают записи обеих таблиц при наличии совпадения. Однако существуют еще и **внешние соединения**, которые возвращают записи, *не имеющие совпадений в другой таблице*, **самосоединения**, которые (как ни странно) *соединяют таблицу саму с собой*, и **союзы**, которые *объединяют результаты запросов*. Освоив эти приемы, вы сможете получить все данные точно в том виде, в котором они вам нужны. (И узнаете правду о подзапросах, как мы и обещали!)



Уничтожение старых данных	446
Левое, правое...	447
Пример левого внешнего соединения	448
Внешние соединения и множественные совпадения	453
Правое внешнее соединение	454
Пока мы занимались внешними соединениями...	457
Создание новой таблицы	458
Место новой таблицы в схеме	459
Рефлексивный внешний ключ	460
Соединение таблицы с ней самой	461
Потребуется самосоединение	463
Другой способ получения многотабличной информации	464
Союзы	465
Ограничения союзов	466
Правила союзов в действии	467
UNION ALL	468
Создание таблицы на основе союза	469
INTERSECT и EXCEPT	470
С союзами разобрались, пора переходить к...	471
Сравнение подзапросов и соединений	471
Преобразование подзапроса в соединение	472
Самосоединение как подзапрос	477
Компания Грега растет	478
Новые инструменты	480

11

Ограничения, представления и транзакции

У семи нянек

Ваша база данных выросла, и теперь с ней будут работать другие люди. К сожалению, далеко не все они так же хорошо разбираются в SQL, как вы. Вам придется позаботиться о том, чтобы **предотвратить ввод неверных данных, запретить просмотр лишних данных, а также предотвратить возможные конфликты при одновременном вводе данных.** В этой главе мы займемся защитой данных от чужих ошибок. Итак — Защита Баз Данных, часть 1.

Грег нанимает помощников	482
Первый день: вставка данных нового клиента	483
Джим не хочет использовать NULL	484
Три месяца спустя	485
Добавление ограничения CHECK	486
Ограничение CHECK для столбца gender	487
Однообразная работа Фрэнка	489
Создание представления	491
Просмотр представлений	492
Как работает представление	493
Что такое представление	494
Вставка, обновление и удаление в представлениях	497
Обновление данных через представление	498
Представление с CHECK OPTION	501
Представление может быть обновляемым, если...	502
Удаление представлений	503
Когда хорошая база данных плохо ведет себя	504
Что произошло с банкоматом	505
Новые неприятности с банкоматами	506
Это не мечты, а транзакции	508
Свойства транзакций	509
SQL помогает работать с транзакциями	510
Как должен был работать банкомат	511
Как работать с транзакциями в MySQL	512
Теперь попробуйте сами	513
Новые инструменты	516



12

Безопасность

Защита данных

Вы потратили массу времени и сил на создание базы данных.

И если теперь с ней что-нибудь случится, это будет полной катастрофой. Кроме того, вам приходится предоставлять другим пользователям **доступ к данным**, и вы опасаетесь, что они могут ошибиться с вставкой или обновлением — или и того хуже, **удалить нужные данные**. В этой главе вы узнаете, как **защитить** базу данных и хранящиеся в ней объекты и как установить контроль над тем, **какие операции с данными разрешены тем или иным пользователям**.

Проблемы с пользователями	520
Предотвращение ошибок в базе данных	521
Защита учетной записи root	523
Создание нового пользователя	524
Решите, что необходимо каждому пользователю	525
Простая команда GRANT	526
Разновидности GRANT	529
Команда REVOKE	530
Отзыв использованной привилегии GRANT OPTION	531
Проблема общих учетных записей	536
Использование роли	538
Удаление ролей	538
Конструкция WITH ADMIN OPTION	540
Объединение CREATE USER с GRANT	545
Оглушительный успех!	546
Новые инструменты	547
Присоединяйтесь!	548
Используйте SQL в своих проектах... и возможно, вас тоже ожидает успех!	548



root



bashful



doc



dopey



grumpy



happy



sleepy



sneezy



Десять важнейших тем (о которых мы не рассказали)

Но даже после всего сказанного беседа еще ченал! Есть еще кое-что, о чем вы должны знать. Мы рел неправильно просто проигнорировать эти темы — они за бы краткого упоминания. Итак, прежде чем откладывать к тесь с этими короткими, но важными разделами. А ког те и эту главу, останется еще пара приложений... и может рекламы... и ничего больше. Честное слово!

1. Используйте графический интерфейс к сво
2. Зарезервированные слова и специальные си
3. ALL, ANY и SOME
4. Подробнее о типах данных
5. Временные таблицы
6. Преобразование типа
7. Имя пользователя и текущее время
8. Полезные числовые функции
8. Полезные числовые функции (продолжение)
9. Индексирование для ускорения операций
10. PHP/MySQL за две минуты

ABSOLUTE ACTION AID ADMIN AFTER AGGREGATE ALIAS ALL ALLOCATE ALTER AND ANY ARE ARRAYS AS
ASC ASSERTION AT AUTHORIZATION
BEFORE BEGIN BINARY BIT BLOB BOOLEAN BOTH BREADTH BY
CALL CASCADES CASE CAST CATALOG CHAR CHARACTER CHECKER CLASS CLOB CLOSE COLLATE
COLLATION COLLUMS COMMIT COMPLETION CONNECT CONNECTION CONSTRAINT CONSTRAINTS
CONSTRUCTOR CONTINUE CORRESPONDING CREATE CROSS CURSOR CURRENT CURRENT DATE
CURRENT PATH CURRENT ROLE CURRENT TIME CURRENT TIMESTAMP CURRENT USER CURSOR CYCLE
DATA DATE DAY DEALLOCATE DEC DECIMAL DECLARE DEFAULT DEFERRABLE DEFERRED DELETE DEPTHS
DEREF DESC DESCRIBE DESCRIPTION DESTROY DESTRUCTOR DETERMINISTIC DICTIONARY DIAGNOSTICS
DISCONNECT DISTINCT DOMAIN DOUBLE DROP DYNAMIC
EACH ELSE END END EXEC EQUALS ESCAPE EVERY EXCEPT EXCEPTION EXEC EXECUTE EXTERNAL
FALSE FETCH FIRST FLOAT FOR FOREIGN FOUND FROM FREE FULL FUNCTION
GENERAL GET GLOBAL GO GOVD GRANT GROUP GROUPS
HAVING HOST HOUR
IDENTITY IGNORE IMMEDIATE IN INDICATOR INITIALIZE INITIALLY INHERIT INOUT INPUT INSERT
INT INTERVAL INTERSECT INTERVAL INTO IS ISOLATION ITERATE
JOIN
KEY
LANGUAGE LARGE LAST LATERAL LEADING LEFT LESS LEVEL LINK LIMIT LOCAL LOCALTIME
LOCALTIMESTAMP LOCATOR
MAP MATCH MINUTE MODIFIES MODIFY MIDDLE MONTH
NAMES NATIONAL NATURAL NCHAR NCOL NEW NEXT NO NONE NOT NULL NUMERIC
OBJECT OF OFF OLD ON ONLY OPEN OPERATION OPTION OR ORDER ORDINALITY OUT OUTER OUTPUT
PAD PARAMETER PARAMETERS PARTIAL PATH POSITION PRECISION PREFIX PREPARED PREPARE
PRESERVE PRIMARY PRIOR PRIVILEGES PROCEDURE PUBLIC
READ READS REAL RECURSIVE REF REFERENCES REFERENCING RELATIVE RESTRICT RESULT RETURN
RECURS REVOKE RIGHT ROLE ROLLBACK ROLLUP ROUTINE ROW ROWS
SAVEPOINT SCHEMA SCROLL SCOPE SEARCH SECOND SECTION SELECT SEQUENCE SESSION
SESSION USER SET SETS SIZE SMALLINT SOME SPACE SPECIFIC SPECIFICTYPE SQL SQLDEFINITION
SQLSTATE SQLWARNING START STATE STATEMENT STATIC STRUCTURE SYTIME SYS
TABLE TEMPORARY TERMINATE THEN TIME TIMEZONE TIMEZONE_HOUR TIMEZONE_MINUTE TO
TRAILING TRANSACTION TRANSLATION TREAT TRIGGER TRIGGER
UNDER UNION UNIQUE UNKNOWN UNKNOWN UPDATE USAGE USER USING
VALUE VALUES VARCHAR VARIABLE VARYING VIEW
WHEN WHENEVER WHERE WITH WITHOUT WORK WRITE
YEAR
ZONE

```

> SELECT CURRENT_DATE,
+-----+
| CURRENT_DATE |
+-----+
| 2007-07-26   |
+-----+
1 row in set (0.00 sec)

```

```

> SELECT CURRENT_TIME,
+-----+
| CURRENT_TIME |
+-----+
| 11:26:49     |
+-----+
1 row in set (0.00 sec)

```

```

> SELECT CURRENT_USER,
+-----+
| CURRENT_USER |
+-----+
| root@localhost |
+-----+
1 row in set (0.00 sec)

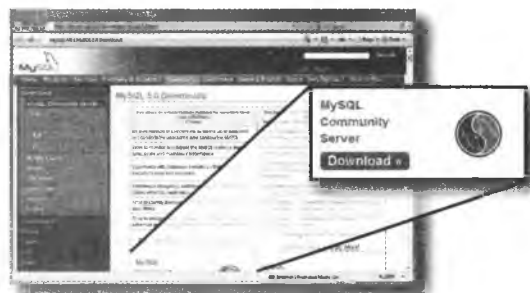
```

II

Попробуйте сами

Ваши новые знания принесут пользу только в том случае, если вы сможете применить их на практике. В этом приложении содержатся инструкции по установке РСУБД MySQL.

За дело!	568
Инструкции и устранение проблем	568
Установка MySQL в системе Windows	569
Установка MySQL в Mac OS X	572



III

Список инструментов

Полный инструментарий SQL

В этом приложении перечислены все инструменты SQL, упоминавшиеся в книге. Не жалейте времени, *просмотрите весь список и возрадуйтесь* — ведь вы изучили их все!



Введение

Не могу поверить, что
они включили такое
в книгу об SQL!



В этом разделе мы ответим
на насущный вопрос:
«Так почему они включили
ТАКОЕ в книгу об SQL?»

Для кого написана эта книга?

Если вы ответите «да» на все следующие вопросы...

- 1 У вас имеется доступ к компьютеру с установленной РСУБД — Oracle, MS SQL или MySQL? Или компьютер, на котором вы ее сможете установить?
- 2 Вы хотите **изучить, понять и запомнить принципы создания таблиц, баз данных и написания запросов по самым лучшим и современным стандартам?**
- 3 Вы **предпочитаете оживленную беседу сухим, скучным академическим лекциям?**

Мы поможем вам изучить концепции и синтаксис SQL так, чтобы по возможности упростить его понимание и практическое применение.

...то эта книга для вас.

Кому эта книга не пойдёт?

Если вы ответите «да» на любой из следующих вопросов...

- 1 Вы **абсолютно уверенно** разбираетесь в начальном синтаксисе SQL и ищете книгу о проектировании баз данных для экспертов?
- 2 Вы уже являетесь опытным программистом и ищете *справочник по SQL?*
- 3 Вы **боитесь попробовать что-нибудь новое?** Скорее пойдете к зубному врачу, чем наденете полосатое с клетчатым? Считаете, что техническая книга, в которой концепции SQL изображены в виде человечков, серьезной быть не может?

Но если вам нужно повторить материал или вы никогда толком не понимали суть нормальных форм, связей «один-ко-многим» и левых внешних соединений, — эта книга вам поможет.

...эта книга не для вас.



*[Заметка от отдела продаж:
вообще-то эта книга для любого,
у кого есть деньги.]*

Мы знаем, о чем вы думаете

«Разве серьезные книги по SQL такие?»

«И почему здесь столько рисунков?»

«Можно ли так чему-нибудь научиться?»

И мы знаем, о чем думает ваш мозг

Мозг жаждет новых впечатлений. Он постоянно ищет, анализирует, *ожидает* чего-то необычного. Он так устроен, и это помогает нам выжить.

Как наш мозг поступает со всеми обычными, повседневными вещами? Он всеми силами пытается оградиться от них, чтобы они не мешали его *настоящей* работе — сохранению того, что действительно *важно*. Мозг не считает нужным сохранять скучную информацию. Она не проходит фильтр, отсекающий «очевидно несущественное».

Но как же мозг *знает*, что важно? Представьте, что вы выехали на прогулку и вдруг прямо перед вами появляется тигр. Что происходит в вашей голове и теле?

Активизируются нейроны. Вспыхивают эмоции. Происходят химические реакции.

И тогда ваш мозг понимает...

Конечно, это важно! Не забывать!

А теперь представьте, что вы находитесь дома или в библиотеке в теплом, уютном месте, где тигры не водятся. Вы учитесь — готовитесь к экзамену. Или пытаетесь освоить сложную техническую тему, на которую вам выделили неделю... максимум десять дней.

И тут возникает проблема: ваш мозг пытается оказать вам услугу. Он старается сделать так, чтобы на эту *очевидно* несущественную информацию не тратились драгоценные ресурсы. Их лучше потратить на что-нибудь важное. На тигров, например. Или на то, что к огню лучше не прикасаться. Или что на лыжах не стоит кататься в футболке и шортах.

Нет простого способа сказать своему мозгу: «Послушай, мозг, я тебе, конечно, благодарен, но какой бы скучной ни была эта книга и пусть мой датчик эмоций сейчас на нуле, я *хочу* запомнить то, что здесь написано».

Ваш мозг считает, что ЭТО важно.



Замечательно. Еще 592 сухие, скучные страницы.

Ваш мозг полагает, что ЭТО можно не запоминать.



Эта книга для тех, кто хочет учиться.

Как мы что-то *узнаем*? Сначала нужно это «что-то» *понять*, а потом *не забыть*. Затоптать в голову побольше фактов недостаточно. Согласно новейшим исследованиям в области когнитивистики, нейробиологии и психологии обучения, для *усвоения материала* требуется что-то большее, чем простой текст на странице. Мы знаем, как заставить ваш мозг работать.

Основные принципы серии «Head First»:



Наглядность. Графика запоминается гораздо лучше, чем обычный текст, и значительно повышает эффективность восприятия информации (до 89% по данным исследований). Кроме того, материал становится более понятным. **Текст размещается на рисунках**, к которым он относится, а не под ними или на соседней странице.

Разговорный стиль изложения. Недавние исследования показали, что при личном разговорном стиле изложения материала (вместо формальных лекций) улучшение результатов на итоговом тестировании составляло до 40%. Рассказывайте историю, вместо того чтобы читать лекцию. Не относитесь к себе слишком серьезно. Что скорее привлечет ваше внимание: занимательная беседа за столом или лекция?

Вам смешно, надеюсь? Ведь я же клоун... Смешно, спрашиваю?



Активное участие читателя. Пока вы не начнете напрягать извилины, в вашей голове ничего не произойдет. Читатель должен быть заинтересован в результате; он должен решать задачи, формулировать выводы и овладевать новыми знаниями. А для этого необходимы упражнения и каверзные вопросы, в решении которых задействованы оба полушария мозга и разные чувства.

Привлечение (и сохранение) внимания читателя. Ситуация, знакомая каждому: «Я очень хочу изучить это, но засыпаю на первой странице». Мозг обращает внимание на интересное, странное, притягательное, неожиданное. Изучение сложной технической темы не обязано быть скучным. Интересное узнается намного быстрее.

Один момент... Вы говорите: «Проверьте эти запросы!» Значит, предполагается, что все они работают. И я вам говорю! Но один запрос вообще не работает, а еще несколько выглядят сомнительно.



Обращение к эмоциям. Известно, что наша способность запоминать в значительной мере зависит от эмоционального сопереживания. Мы запоминаем то, что нам безразлично.

Мы запоминаем, когда что-то чувствуем. Нет, сантименты здесь ни при чем: речь идет о таких эмоциях, как удивление, любопытство, интерес, и чувство «Да я крут!» при решении задачи, которую окружающие считают сложной — или когда вы понимаете, что разбираетесь в теме лучше, чем всезнайка-Боб из технического отдела.



Метапознание: наука о мышлении

Если вы действительно хотите быстрее и глубже усваивать новые знания — задумайтесь над тем, как вы задумываетесь. Учитесь учиться.

Мало кто из нас изучает теорию метапознания во время учебы. Нам *положено* учиться, но нас редко этому *учат*.

Но раз вы читаете эту книгу, то, вероятно, вы хотите изучить SQL, и по возможности быстрее. Вы хотите *запомнить* прочитанное, а для этого абсолютно необходимо сначала *понять* прочитанное.

Чтобы извлечь максимум пользы из учебного процесса, нужно заставить ваш мозг воспринимать новый материал как Нечто Важное. Критичное для вашего существования. Такое же важное, как тигр. Иначе вам предостойт бесконечная борьба с вашим мозгом, который всеми силами уклоняется от запоминания новой информации.

Как же УБЕДИТЬ мозг, что язык SQL не менее важен, чем тигр?

Есть способ медленный и скучный, а есть быстрый и эффективный. Первый основан на тупом повторении. Всем известно, что даже самую скучную информацию *можно* запомнить, если повторять ее снова и снова. При достаточном количестве повторений ваш мозг прикидывает: «*Вроде бы несущественно, но раз одно и то же повторяется столько раз... Ладно, уговорил*».

Быстрый способ основан на **повышении активности мозга** и особенно сочетании разных ее *видов*. Доказано, что все факторы, перечисленные на предыдущей странице, помогают вашему мозгу работать на вас. Например, исследования показали, что размещение слов *внутри* рисунков (а не в подписях, в основном тексте и т. д.) заставляет мозг анализировать связи между текстом и графикой, а это приводит к активизации большего количества нейронов. Больше нейронов — выше вероятность того, что информация будет сочтена важной и достойной запоминания.

Разговорный стиль тоже важен: обычно люди проявляют больше внимания, когда они участвуют в разговоре, так как им приходится следить за ходом беседы и высказывать свое мнение. Причем мозг совершенно не интересуется, что вы «разговариваете» с книгой! С другой стороны, если текст сух и формален, то мозг чувствует то же, что чувствуете вы на скучной лекции в роли пассивного участника. Его клонит в сон.

Но рисунки и разговорный стиль — это только начало.



Вот что сделали Мы:

Мы использовали **рисунки**, потому что мозг лучше приспособлен для восприятия графики, чем текста. С точки зрения мозга рисунок стоит 1024 слов. А когда текст комбинируется с графикой, мы внедряем текст прямо в рисунки, потому что мозг при этом работает эффективнее.

Мы используем **избыточность**: повторяем одно и то же несколько раз, применяя *разные* средства передачи информации, обращаемся к разным чувствам — и все для повышения вероятности того, что материал будет закодирован в нескольких областях вашего мозга.

Мы используем концепции и рисунки несколько **неожиданным** образом, потому что мозг лучше воспринимает новую информацию. Кроме того, рисунки и идеи обычно имеют **эмоциональное содержание**, потому что мозг обращает внимание на биохимию эмоций. То, что заставляет нас *чувствовать*, лучше запоминается — будь то *шутка*, *удивление* или *интерес*.

Мы используем **разговорный стиль**, потому что мозг лучше воспринимает информацию, когда вы участвуете в разговоре, а не пассивно слушаете лекцию. Это происходит и при *чтении*.

В книгу включены многочисленные упражнения, потому что мозг лучше запоминает, когда вы что-то делаете. Мы постарались сделать их непростыми, но интересными — то, что предпочитает большинство читателей.

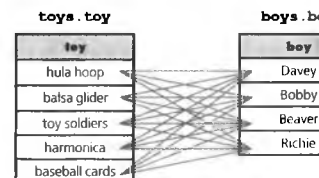
Мы совместили **несколько стилей обучения**, потому что одни читатели предпочитают пошаговые описания, другие стремятся сначала представить «общую картину», а третьим хватает фрагмента кода. Независимо от ваших личных предпочтений полезно видеть несколько вариантов представления одного материала.

Мы постарались задействовать **оба полушария вашего мозга**; это повышает вероятность усвоения материала. Пока одна сторона мозга работает, другая часто имеет возможность отдохнуть; это повышает эффективность обучения в течение продолжительного времени.

А еще в книгу включены **истории** и упражнения, отражающие другие точки зрения. Мозг глубже усваивает информацию, когда ему приходится оценивать и выносить суждения.

В книге часто встречаются **вопросы**, на которые не всегда можно дать простой ответ, потому что мозг быстрее учится и запоминает, когда ему приходится что-то делать. Невозможно накачать *мышцы*, наблюдая за тем, как занимаются *другие*. Однако мы позаботились о том, чтобы усилия читателей были приложены в *верном* направлении. Вам не придется ломать голову над невразумительными примерами или разбираться в сложном, перенасыщенном техническим жаргоном или слишком лаконичном тексте.

В историях, примерах, на картинках используются **люди** — потому что вы тоже **человек**. И ваш мозг обращает на людей больше внимания, чем на неодушевленные **предметы**.



Беседа у камина



КЛЮЧЕВЫЕ МОМЕНТЫ





Вырежьте
и прикрепите
на холодильник.

Что можете сделать ВЫ, чтобы заставить свой мозг повиноваться

Мы свое дело сделали. Остальное за вами. Эти советы станут отправной точкой; прислушайтесь к своему мозгу и определите, что вам подходит, а что не подходит. Пробуйте новое.

- **Не торопитесь. Чем больше вы поймете, тем меньше придется запоминать.**

Просто читать недостаточно. Когда книга задает вам вопрос, не переходите к ответу. Представьте, что кто-то *действительно* задает вам вопрос. Чем глубже ваш мозг будет мыслить, тем скорее вы поймете и запомните материал.

- **Выполняйте упражнения, делайте заметки.**

Мы включили упражнения в книгу, но выполнять их за вас не собираемся. И не *разглядывайте* упражнения. **Берите карандаш и пишите** Физические действия *во время* учения повышают его эффективность.

- **Читайте врезки.**

Это значит: читайте все. *Врезки – часть основного материала!* Не пропускайте их.

- **Не читайте другие книги после этой перед сном.**

Часть обучения (особенно перенос информации в долгосрочную память) происходит *после* того, как вы откладываете книгу. Ваш мозг не сразу усваивает информацию. Если во время обработки поступит новая информация, часть того, что вы узнали ранее, может быть потеряна.

- **Пейте воду. И побольше.**

Мозг лучше всего работает в условиях высокой влажности. Дегидратация (которая может наступить еще до того, как вы почувствуете жажду) снижает когнитивные функции.

- **Говорите вслух.**

Речь активизирует другие участки мозга. Если вы пытаетесь что-то понять или лучше запомнить, произнесите вслух. А еще лучше – попробуйте объяснить кому-нибудь другому. Вы будете быстрее усваивать материал и, возможно, откроете для себя что-то новое.

- **Прислушивайтесь к своему мозгу.**

Следите за тем, когда ваш мозг начинает уставать. Если вы начинаете поверхностно воспринимать материал или забываете только что прочитанное – пора сделать перерыв.

- **Чувствуйте!**

Ваш мозг должен знать, что материал книги действительно *важен*. Переживайте за героев наших историй. Придумывайте собственные подписи к фотографиям. Поморщиться над неудачной шуткой все равно лучше, чем не почувствовать ничего.

- **Творите!**

Попробуйте применить новые знания в своей повседневной работе. Просто сделайте *хоть что-нибудь*, чтобы приобрести практический опыт за рамками упражнений. Все, что для этого нужно – это карандаш и подходящая задача... задача, в которой изучаемые методы и инструменты могут принести пользу.

Примите к сведению

Это учебник, а не справочник. Мы намеренно убрали из книги все, что могло бы помешать изучению материала, над которым вы работаете. И при первом чтении книги начинать следует с самого начала, потому что книга предполагает наличие у читателя определенных знаний и опыта.

Мы начинаем с основного синтаксиса SQL, а затем переходим к проектированию баз данных SQL и построению нетривиальных запросов.

Правильное проектирование таблиц и баз данных — дело, конечно, важное, но сначала необходимо понять синтаксис SQL. Поэтому наш учебный курс начинается с команд SQL, которые вы можете опробовать сами. А когда это у вас получится, вы начнете испытывать интерес к теме. Затем, в более поздних главах книги, мы опишем правила проектирования таблиц. К этому времени вы уже будете хорошо понимать необходимый синтаксис и сможете сосредоточиться на *изучении концепций*.

Мы не пытаемся описать все без исключения команды, функции и ключевые слова SQL.

Теоретически мы могли бы включить в книгу описание всех команд, функций и ключевых слов, но читатель, вероятно, предпочтет иметь дело с книгой, которую можно сдвинуть с места. Поэтому мы приводим лишь тот материал, который действительно абсолютно необходим — то, чем вы будете пользоваться в 95% случаев. А после прочтения книги вы сможете без проблем найти описание нужной функции в справочнике.

Мы не описываем все тонкости синтаксиса для всех разновидностей SQL.

Существует много разных РСУБД: Standard SQL, MySQL, Oracle, MS SQL Server, PostgreSQL, DB2... и это еще не все. Если бы мы стали описывать все различия в синтаксисе всех команд, то книга стала бы во много раз толще, поэтому в книге основное внимание уделяется стандартному синтаксису SQL с небольшим уклоном в сторону MySQL. Все примеры книги будут работать в MySQL. Кроме того, большинство из них будет работать во всех перечисленных выше РСУБД. Помните, ранее мы советовали купить справочник? Так вот, купите специализированный справочник для той РСУБД, с которой вы работаете.

Упражнения ОБЯЗАТЕЛЬНЫ.

Упражнения являются частью основного материала книги. Одни упражнения способствуют запоминанию материала, другие помогают лучше понять его, третьи ориентированы на его практическое применение. **Не пропускайте упражнения.**

Повторение применяется намеренно.

У книг этой серии есть одна принципиальная особенность: мы хотим, чтобы вы *действительно хорошо* усвоили материал. И чтобы вы запомнили все, что узнали. Большинство справочников не ставит своей целью успешное запоминание, но это не справочник, а учебник, поэтому некоторые концепции излагаются в книге по несколько раз.

Примеры кода были сделаны по возможности компактными.

Наши читатели не любят просматривать по 200 строк кода, чтобы найти две нужные строки. Большинство примеров книги приводится в минимальном контексте, чтобы та часть, которую вы непосредственно изучаете, была понятной и простой. Не ждите, что весь код будет стопроцентно устойчивым или даже просто завершенным — примеры написаны в учебных целях и не всегда являются полнофункциональными.

Многие команды доступны в Интернете, чтобы вы могли скопировать их для выполнения в своем терминале или рабочей программе. Их можно загрузить по адресу <http://www.headfirstlabs.com/books/hfsql/>

Упражнения «Игры разума» не имеют ответов.

В некоторых из них правильного ответа вообще нет, в других вы должны сами решить, насколько правильны ваши ответы (это является частью процесса обучения). В некоторых упражнениях «Игры разума» приводятся подсказки, которые помогут вам найти нужное направление.

Установка сервера SQL.

Чтобы создавать базы данных и таблицы средствами SQL, а также работать с ними, вам понадобится доступ к серверу SQL. Возможно, на вашем веб-сервере уже имеется установленная и настроенная РСУБД, а если нет — установите ее на своем домашнем компьютере. В приложении ii приведены инструкции по установке MySQL (популярная бесплатная реализация SQL) для Mac и Windows.

SQL для ленивых.

Если вам не хочется возиться с установкой РСУБД на своем компьютере и вы просто хотите самостоятельно проверить примеры, приведенные в книге, вам повезло! Мы создали специальную «SQL-песочницу», в которой вы можете поиграть и поэкспериментировать с большинством примеров. Посетите страницу по адресу http://www.headfirstlabs.com/sql_hands_on/

Технические рецензенты

Кэри Коллетт



Чосер тоже помогал.

Стив Милано



Шелли Римс



Джейми Хендерсон



Лу-Энн Мазза



Наши замечательные рецензенты

Мы бесконечно благодарны нашим техническим рецензентам, вылавливавшим бесчисленные грубые ляпсусы, коварные ошибки и банальные опечатки. Без них материал книги не был бы и наполовину таким четким и правильным. Они отлично справились со своей работой.

Кэри Коллетт применил 15 лет практического опыта, полученного во время работы в начинающих фирмах, правительственных лабораториях и финансовом секторе, в ходе рецензирования этой книги. Сейчас он надеется вернуться к другим приятным занятиям, не связанным с работой, — кулинарии, туризму, чтению книг и развлечениям с собаками.

Лу-Энн Мазза выкроила немного времени из своей занятой профессиональной жизни программиста и аналитика для написания исключительно точных и актуальных рецензий. Мы рады, что теперь она сможет уделять больше времени своим увлечениям — велоспорту, фотографии, компьютерам, музыке и теннису.

Когда **Стив Милано** не программирует на полдюжине разных языков, не пишет рецензии по книгам и не играет в группе панк-рока Onion

Flavored Rings в плохо проветриваемых подвалах по всей стране, обычно он сидит дома со своими котами Ральфом и Сквиком.

«Шелли» **Мойра Мишель Римс**, магистр педагогических наук, обладательница сертификатов MCP и MCSE, ведет образовательные программы для детей младшего возраста в общественном колледже Дельгадо (Нью-Орлеан). В настоящее время она переводит свои учебные курсы в интернет-формат, чтобы обеспечить изменившиеся потребности населения Нью-Орлеана после урагана «Катрина». Мы благодарны ей за то, что она смогла выбрать для нас время в своем плотном графике.

Джейми Хендерсон — ведущий специалист по системным архитектурам. Ее волосы выкрашены в фиолетовый цвет, а свободное время делится между виолончелью, книгами, видеоиграми и фильмами на DVD.

Благодаря этой замечательной группе код и упражнения в книге делают именно то, что им положено делать, а читатель, перевернув последнюю страницу, начнет вполне уверенно программировать на SQL.

Благодарности

Редакторы

Прежде всего хочу поблагодарить своего редактора **Бретта Маклафлина** за целых два учебных курса по серии Head First. Бретт не просто редактор — умение выслушать собеседника сочетается с готовностью помочь. Без его руководства, поддержки и интереса эта книга никогда не была бы написана. Он не только понял меня с первого собеседования, но и благожелательно относился к моему (порой чрезмерному) юмору. В результате эта книга стала самым лучшим из всех моих авторских проектов. Бретт дал мне много полезных советов, и благодаря ему я узнала много полезного в ходе работы. Спасибо, Бретт!



Бретт
Маклафлин



Кэтрин Нолан

Редактор **Кэтрин Нолан** наверняка потеряла немало нервных клеток из-за того, что мне очень сильно не повезло в конце процесса редактирования. Возможно, только благодаря ей эта книга все-таки вышла в свет. Она управлялась с проектом виртуозно, словно жонглер, и ухитрилась ничего не уронить. Мне был отчаянно необходим четко расписанный план, а Кэтрин — лучший планировщик из всех, кого я когда-либо встречала. Вероятно, я создала ей больше всего проблем. Будем надеяться, что следующий проект Кэтрин пройдет более гладко — она это заслужила!

Сотрудники издательства O'Reilly

Художественный редактор **Луиза Барр** была для нас хорошим другом и талантливым дизайнером. Каким-то образом ей удалось преобразовать мои безумные идеи во впечатляющие художественные образы, которые предельно ясно передают самые сложные концепции. Вся работа по дизайну была выполнена ей, и я уверена, что на многих страницах книги вам захочется поблагодарить ее вместе со мной.

Книга наверняка содержала бы массу ошибок, если бы не процесс технического рецензирования. **Сандерс Клейнфилд** отлично справился с работой редактора по производству и подготовил книгу к печати. Кроме того, он вышел далеко за рамки своих прямых обязанностей и указал целый ряд концептуальных пробелов, которые действительно стоило заполнить. Спасибо, Сандерс!

Наконец, я хочу поблагодарить **Кэти Сьерра** и **Берта Бэйтса** за создание этой замечательной серии и за самый лучший и напряженный учебный курс, который мне довелось пройти за свою жизнь. Если бы не эти три дня — даже не хочу думать, насколько сложнее мне было бы работать над книгой. Завершающие комментарии Берта отличались беспощадной точностью и значительно улучшили эту книгу.



Лу Барр

Всему свое место

Раньше я хранила истории болезни на бумаге, но они постоянно терялись. А теперь я изучила SQL, и у меня ничего не потеряется! Вот как полезно знать, что такое таблицы!



Разве не обидно потерять? Что угодно — ключи от машины, купон на скидку в 25%, данные приложения... Нет ничего хуже, чем невозможность **найти то, что вам нужно...** и именно тогда, когда нужно. А в том, что касается приложений, для хранения важной информации не найти места лучше, чем **таблица**. Так что переверните страницу и присоединяйтесь к нашей прогулке по миру **реляционных баз данных**.

Определение данных

У Грега много друзей. Он обожает знакомить их друг с другом, но, конечно, для этого ему нужно помнить, кто чем увлекается. Грег аккуратно записывает подробную информацию на листочках:



Грег уже очень давно использует свою систему. А на прошлой неделе он решил включить в нее людей, занимающихся поисками работы, так что его каталог быстро растет. Очень быстро...



Неудобно
записок и пром



МОЗГОВОЙ ШТУРМ

Нет ли более разумного способа хранения этой информации? А что бы сделали Вы на его месте?



Как насчет базы данных?
Ведь книга написана о базах
данных, верно?

**Абсолютно верно. База данных —
именно то, что нам нужно.**

Но прежде чем браться за создание ба-
зы данных, необходимо получше разо-
браться в том, какие виды данных будут
в ней храниться и на какие *категории*
они будут разделены.

Возьми в руку карандаш



Перед вами несколько карточек из каталога Грега. Найдите сходные данные, собранные Грегом о каждом человеке. Присвойте таким данным метку, описывающую категорию информации, и запишите эти метки в отведенных полях.

Энн Бренсон
Дата рождения: 1/7/1962
Программист

Не замужем,
но есть планы

Маунтин-Вью, СА
annie@boards-r-us.com

Увлечения: книги, пешие
прогулки, домашнее пиво

Ищет: новая работа

Джейми Гамильтон
Дата рождения: 10/9/1964
Системный администратор

Не женат

Саннивейл, СА
dontbother@breakneckpizza.com

Увлечения: туризм, литера-
тура

Ищет: друзья, женщины

Алан Скуп
Дата рождения: 1/7/1966
Инженер

Женат

Сан-Антонио, ТХс
soukip@breakneckpizza.com

Увлечения: ролевые игры,
программирование

Ищет: ничего

Анджелина Мендоса
Дата рождения:
19/8/1979
Системный
администратор

Замужем

Сан-Франциско, СА
angelina@starbuzzcoffee.com

Увлечения: театр, танцы

Ищет: новая работа

Ищет

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Возьми в руку карандаш



Решение

Перед вами несколько карточек из каталога Грегa. Найдите сходные данные, собранные Грегом о каждом человеке. Присвойте таким данным метку, описывающую категорию информации, и запишите эти метки в отведенных полях.

Имя → Энн Бренсон
 Дата рождения: 1/7/1962
 Программист
 Статус → Не замужем, но есть планы
 Маунтин-Вью, СА
 annie@boards-r-us.com
 Увлечения: книги, пешие прогулки, домашнее пиво
 Ищет: новая работа

Фамилия → Джеймс Гамильтон
 Дата рождения: 10/9/1964
 Системный администратор
 Не женат
 Саннивейл, СА
 dontbother@breakneckpizza.com
 Увлечения: туризм, литература
 Ищет: друзья, женщины

Категории, которые мы определим, будут использоваться для упорядочения данных.

Мы разделили полное имя на имя и фамилию. Такое деление позднее пригодится нам при сортировке данных.

- Имя
- Фамилия
- Дата рождения
- Профессия
- Статус
- Место жительства
- Электронная почта
- Увлечения
- Ищет

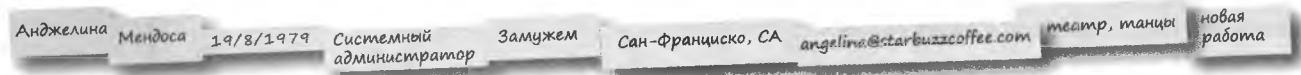
Профессия → Алан Скуп
 Дата рождения: 1/7/1966
 Инженер
 Женат
 Место жительства → Сан-Антонио, TX
 soukup@breakneckpizza.com
 Электронная почта → увлечения: ролевые игры, программирование
 Ищет: ничего

Анджелина Мендоса
 Дата рождения: 19/8/1979
 Системный администратор
 Замужем
 Сан-Франциско, СА
 angelina@starbuzzcoffee.com
 Увлечения: театр, танцы
 Ищет: новая работа

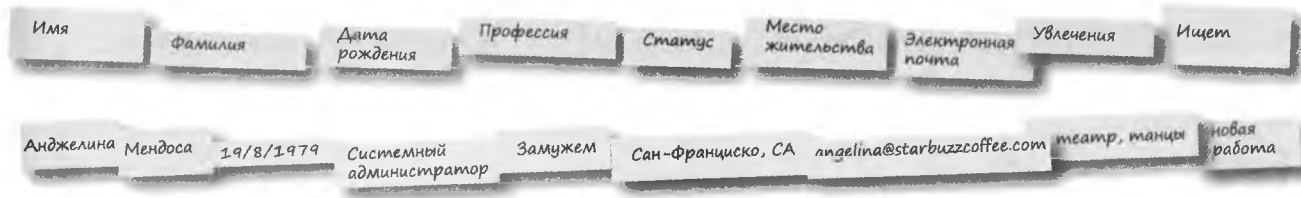
Грег уже записал на своих карточках названия некоторых категорий: «Дата рождения», «Увлечения» и «Ищет».

Рассматриваем данные по категориям

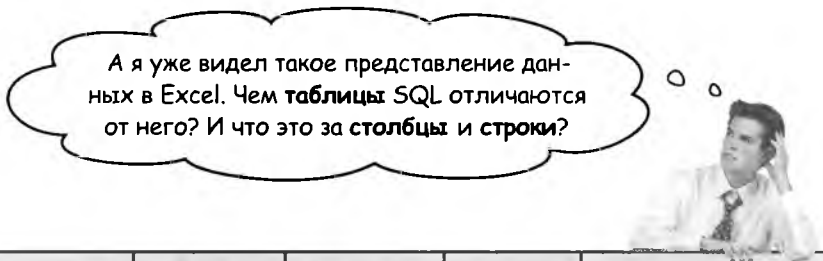
Давайте взглянем на данные с новой точки зрения. Если разрезать каждый листок на полоски, а затем разложить их по горизонтали, вот что у вас получится:



Если теперь разрезать еще один листок с названиями этих категорий и разложить полоски над соответствующими данными, результат будет выглядеть примерно так:



А вот как выглядит та же информация в виде **ТАБЛИЦЫ** из строк и столбцов.



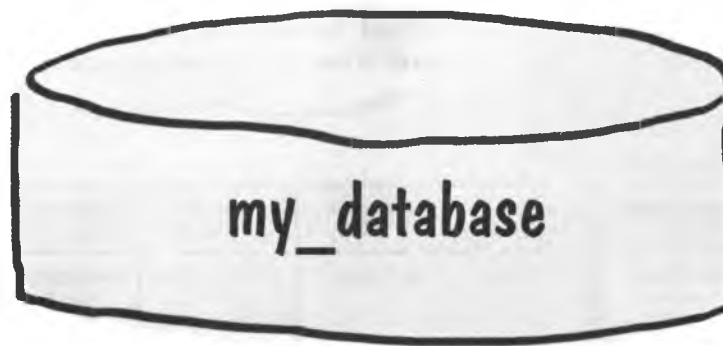
фамилия	имя	электронная почта	дата рождения	профессия	место жительства	статус	увлечения	ищет
Бренсон	Энн	annie@boards-r-us.com	1-7-1962	программист	Маунтин-Вью, CA	не замужем, но есть планы	ролевые игры, программирование	новая работа
Гамильтон	Джейми	dontbother@breakneckpizza.com	10-9-1964	системный администратор	Саннивейл, CA	не женат	туризм, литература	друзья, женщины
Скуп	Алан	soukup@breakneckpizza.com	1-7-1966	инженер	Сан-Антонио, TX	женат	ролевые игры, программирование	ничего
Мендоса	Анджелина	angelina@starbuzzcoffee.com	19-8-1979	системный администратор	Сан-Франциско, CA	замужем	театр, танцы	новая работа

Что такое «база данных»?

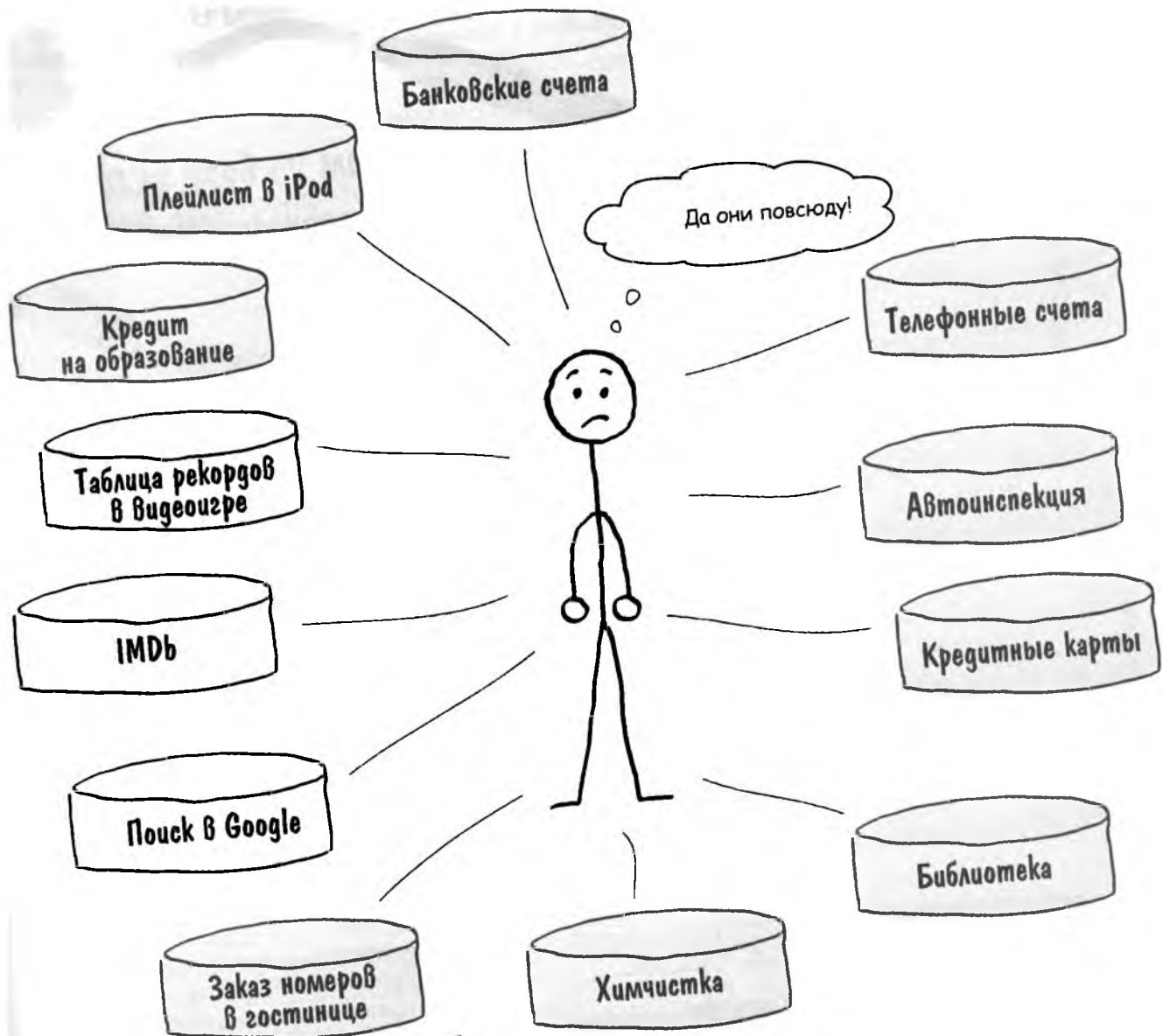
Прежде чем переходить к подробному рассмотрению таблиц, строк и столбцов, давайте сделаем шаг назад и попробуем представить общую картину. Первая структура SQL, о которой вы должны знать, — контейнер, в котором хранятся все ваши таблицы. Она-то и называется *базой данных*.

Базой данных называется контейнер, в котором хранятся таблицы и другие структуры SQL для работы с ними.

Каждый раз, когда вы выполняете поиск в Интернете, обращаетесь за справкой, используете TiVo, заказываете билеты, получаете штраф за превышение скорости или покупаете что-нибудь в магазине, необходимая информация *запрашивается из базы данных*.



На диаграммах и блок-схемах базы данных изображаются в виде цилиндров. Когда вы видите такое изображение, значит, перед вами база данных.



Вы и лишь немногие из окружающих вас баз данных.



Посмотрим на базу данных через волшебные очки...

Думайте о базе данных как о контейнере, в котором хранится информация.

Таблица

столбец1	столбец2	столбец3	столбец4	столбец5	столбец6
данные	данные	данные	данные	данные	данные
данные	данные	данные	данные	данные	данные
данные	данные	данные	данные	данные	данные

Это столбцы.

Другая таблица.

столбец1	столбец2	столбец3	столбец4
данные	данные	данные	данные
данные	данные	данные	данные
данные	данные	данные	данные
данные	данные	данные	данные

А это строки.

Тоже таблица.

столбец1	столбец2	столбец3
данные	данные	данные
данные	данные	данные
данные	данные	данные
данные	данные	данные

И еще одна таблица.

Информация в базе данных делится на таблицы.

База данных состоит из таблиц.

Таблицей называется структура, в которой хранятся данные, упорядоченные по **столбцам** и **строкам**.

Помните категории из предыдущего примера? Каждая категория соответствует столбцу таблицы. Например, столбец может содержать одно из значений: Не женат, Женат, Разведен.

Строка таблицы содержит всю информацию об одном объекте таблицы. В новой таблице Грега строка содержит полное описание одного человека. Например, в одной строке могут храниться следующие данные: Джон, Джексон, не женат, писатель, jj@boards-r-us.com.

СТАНЬ таблицей



Ниже вы найдете несколько карточек и таблицу. Ваша задача — представить себя на месте частично заполненной таблицы, заполнить пустые места и достичь просветления. Когда вы справитесь с упражнением, переверните страницу и проверьте, удалось ли вам достичь духовного единения с таблицей.



Duncan's Donuts
5
25/4
с вареньем
8:56
жирноваты

Starbuzz Coffee
23/4
с вареньем
9
7:43
почти идеально

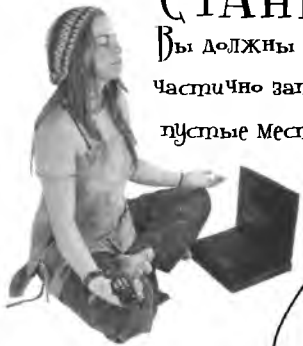
Duncan's Donuts
7
24/4
мало варенья
22:35
с вареньем

с вареньем
вчерашие, но вкусные
6
Krispy King
26/4
21:39

Присвойте полям таблицы осмысленные имена.

shop				
			9	
		25/4	5	
				мало варенья

СТАТЬ таблицей. Ответ



Если ваши имена столбцов не совпали с нашими — ничего страшного.

Вы должны были представить себя на месте частично заполненной таблицы и заполнить пустые места.

По содержанию карточек понятно, что речь идет о пончиках с вареньем.

jelly_doughnuts

shop	time	date	rating	comments
Starbuzz Coffee	7:43	23/4	9	почти идеально
Duncan's Donuts	8:56	4/25	5	жирноваты
Krispy King	9:39 pm	4/26	6	вчерашние, но вкусные
Duncan's Donuts	10:35 pm	4/24	7	мало варенья

В базах данных хранится логически связанная информация

Все таблицы в базе данных должны быть так или иначе **связаны** между собой. Например, база данных с информацией о съеденных пончиках может состоять из следующих таблиц:

База данных с именем 'my_snacks' состоит из трех таблиц.

Имена базы данных и таблиц обычно записываются символами нижнего регистра.

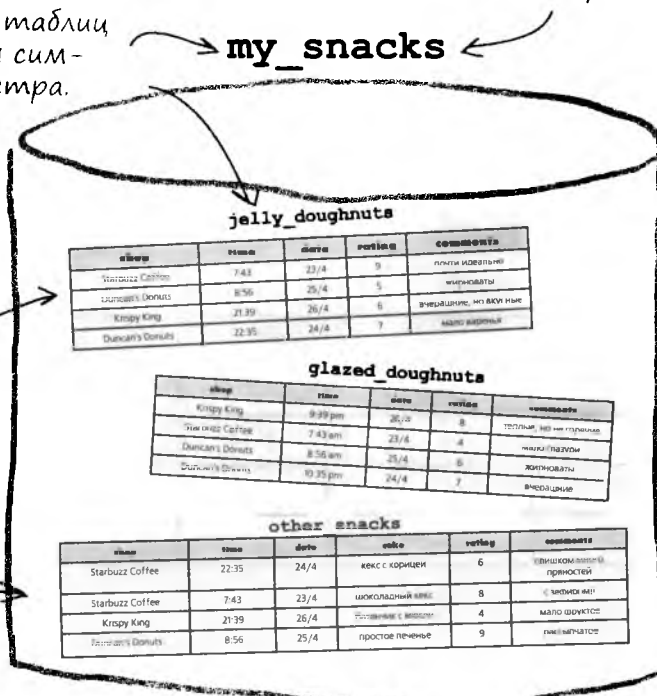


Таблица с информацией о пончиках с вареньем.

Таблица с информацией о пончиках с глазурью.

Таблица с информацией о других блюдах.



Столбец – фрагмент данных, хранящихся в таблице. **Строка** (или **запись**) – набор столбцов, описывающих атрибуты одного объекта. Столбцы и строки образуют таблицу.

Перед вами пример таблицы для хранения данных адресной книги. **Столбцы** также часто называются **полями** – эти два термина означают одно и то же. Кроме того, термины **строка** и **запись** тоже считаются синонимами.



Объединяем столбцы со строками – и получаем таблицу.

first_name	last_name	address	city	state	id_num
Джо	Эппс	данные	данные	данные	данные
Эл	Джонс	данные	данные	данные	данные
Мэри	Моррис	данные	данные	данные	данные
Лу	Грин	данные	данные	данные	данные



Выходит, данные с моих карточек можно преобразовать в таблицу?

Вот именно. Сначала информация о каждом человеке разделяется на категории.

Категории становятся столбцами. Каждая карточка преобразуется в запись. Вы можете взять всю информацию с карточек и преобразовать ее в таблицу.



Теперь вы знаете, что категории называются столбцами...

Данные одной карточки образуют строку.

last_name	first_name	email	birthday	profession	location	status	interests	seeking
Бренсон	Энн	annie@boards-r-us.com	1-7-1962	Программист	Маунтин-Вью, CA	Не замужем, но есть планы	Книги, пешие прогулки, домашнее пиво	Новая работа
Гамильтон	Джейми	dontbother@yahoo.com	10-9-1966	Системный администратор	Саннивейл, CA	Не женат	Туризм, литература	Друзья, женщины
Скуп	Алан	fprose@yahoo.com	2-12-1975	Инженер	Сан-Антонио, TX	Женат	Ролевые игры	Ничего
Мендоса	Анджелина	angel79@gmail.com	19-8-1979	Системный администратор	Сан-Франциско, CA	Замужем	Театр, танцы	Новая работа

...и что данные каждой карточки размещаются в одной строке, которая также может называться записью.

Наконец-то. Но как я буду создавать свою таблицу?



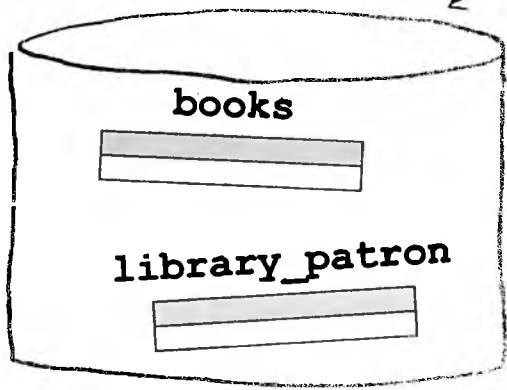


Упражнение

Перед вами несколько баз данных и таблиц. Подумайте, какие категории данных вы бы включили в каждую таблицу.

library_db

База данных библиотеки.



books (книги) :

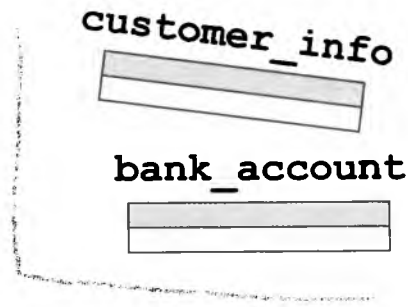
library_patron (посетители) :

bank_db

База данных банка.

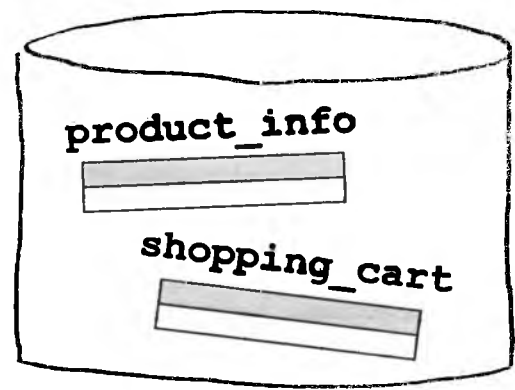
customer_info (клиенты) :

bank_account (счета) :




onlinestore_db

База данных интернет-магазина.



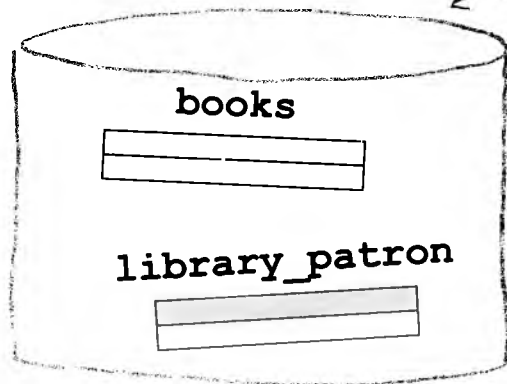
product_info (товары) :

shopping_cart (корзина) :


Упражнение
Решение

Перед вами несколько баз данных и таблиц. Подумайте, какие категории данных вы бы включили в каждую таблицу.

library_db

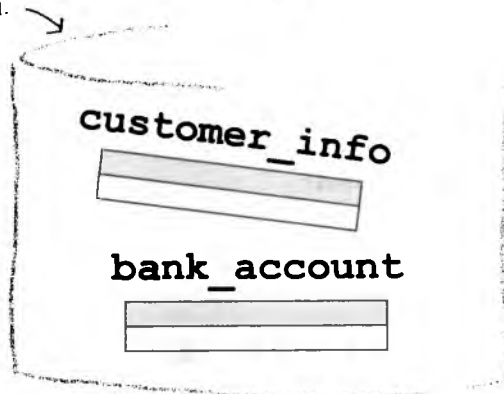


← База данных библиотеки.

books (книги): название, автор, цена,
номер по каталогу

library_patron
(посетители): имя, фамилия, адрес

bank_db



← База данных банка.

customer_info
(клиенты): имя, фамилия, адрес,
номер счета

bank_account
(счета): баланс, депозиты, снятие
средств

onlinestore_db



← База данных интернет-магазина.

product_info
(товары): название, размер, цена

shopping_cart
(корзина): сумма, код клиента

Командуйте!

Запустите свою систему управления реляционной базой данных SQL (РСУБД). Откройте окно командной строки или графическую оболочку для работы с РСУБД. Наше окно терминала после запуска MySQL выглядит так:

```
File Edit Window Help CommandMeBaby
Welcome to the SQL monitor. Commands end with ; or \g.

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

>
```

Угловая скобка — приглашение командной строки. После нее вводятся команды управления.

Пробелы в именах баз данных и таблиц SQL запрещены. Вместо них обычно используются символы подчеркивания.

Прежде всего необходимо создать базу данных, в которой будут храниться таблицы с данными.

- 1 Введите следующую команду для создания базы данных с именем **gregs_list**.

CREATE DATABASE gregs_list;

CREATE DATABASE команда.

gregs_list — имя базы данных.

Команда должна завершаться символом «;» (точка с запятой).

```
File Edit Window Help CommandMeBaby
> CREATE DATABASE gregs_list;
Query OK, 1 row affected (0.01 sec)
```

Сообщение от РСУБД показывает, что запрос был успешно выполнен.



Будьте осторожны!

А вы прочитали Введение?

Мы используем MySQL для управления своими базами данных, поэтому команды вашей системы управления базами данных (СУБД) могут выглядеть немного иначе. Инструкции по установке MySQL на сервере приведены в приложении II.

- 2 Теперь необходимо приказать РСУБД использовать только что созданную базу данных:

```
USE gregs_list;
```

Теперь все последующие операции будут выполняться с базой данных gregs_list!

```
File Edit Window Help USEful
> USE gregs_list;
Database changed
```

Чаще Задаваемые Вопросы

В: Зачем создавать базу данных, если таблица всего одна?

О: Язык SQL требует, чтобы все таблицы находились в базах данных. Для такого требования имеются веские причины. В частности SQL позволяет управлять доступом к таблицам со стороны нескольких пользователей. Предоставить или запретить доступ ко всей базе данных обычно проще, чем управлять доступом для каждой из множества таблиц.

В: Команда CREATE DATABASE записана символами верхнего регистра. Это обязательно?

О: В некоторых системах ключевые слова должны записываться в верхнем регистре, но в SQL регистр игнорируется. Это означает, что записывать команды в верхнем регистре не обязательно, но это считается признаком хорошего стиля. Взгляните на только что введенную нами команду:
CREATE DATABASE gregs_list;
Регистр символов позволяет легко отделить команду (CREATE DATABASE) от имени базы данных (gregs_list).

В: Как выбираются имена баз данных, таблиц и столбцов?

О: Старайтесь выбирать содержательные имена. Иногда для этого приходится строить имя из нескольких слов. Пробелы в именах запрещены, но они обычно заменяются символами подчеркивания. Несколько распространенных вариантов:

```
gregs_list
gregslist
Gregslist
gregsList
```

В: А если я предпочитаю имя «gregsList» без подчеркиваний?

О: Нет проблем. Здесь важно выбрать один стиль и придерживаться его. Если вы присвоили базе данных имя gregsList без подчеркиваний, а второе слово начинается с буквы верхнего регистра, применяйте эту схему ко всем таблицам базы данных — например, назовите другую таблицу myContacts, чтобы не нарушать выбранную схему.

В: Разве не правильнее было бы называть базу данных greg's_list? Почему в имени нет апострофа?

О: Апострофы имеют особый смысл в SQL. Теоретически апостроф можно включить в имя таблицы, но проще обойтись без него.

В: Я также заметил символ «;» в конце команды CREATE DATABASE. Для чего он нужен?

О: Символ «;» является признаком конца команды.

Регистр символов и подчеркивания упрощают работу программиста (хотя для SQL они не нужны!).

Создание таблицы: команда CREATE TABLE

Давайте посмотрим, как создаются таблицы, на примере данных с описаниями пончиков. Предположим, вы часто забываете, что означает то или иное название. Просто *создайте таблицу*, и вам не придется ничего запоминать. Введите приведенную ниже команду в консольном окне, а потом нажмите RETURN, чтобы РСУБД выполнила эту команду.

doughnut_list

doughnut_name	doughnut_type
Blooberry	с начинкой
Cinnamondo	пышки
Rockstar	хворост
Carameller	хворост
Appleblush	с начинкой

Команда SQL для создания таблицы — обратите внимание на регистр символов.

Имя таблицы записывается в нижнем регистре и может содержать символы подчеркивания.

Просто нажмите RETURN, чтобы продолжить команду с новой строки — так вам будет проще разобратся в ее синтаксисе.

Левая круглая скобка открывает список столбцов в создаваемой таблице.

CREATE TABLE doughnut_list

Имена столбцов разделяются запятыми.

Имя первого столбца таблицы.

doughnut_name VARCHAR(10),

Имя второго столбца.

doughnut_type VARCHAR(6)

Правая круглая скобка завершает список столбцов.

Символ «;» сообщает РСУБД о завершении команды.

А это ТИП ДАННЫХ. В столбцах типа VARCHAR (VARIABLE CHARACTER) хранится текстовая информация. Запись (6) означает, что длина текста не превышает 6 символов.

Эй, а как же я? Как создать таблицу для моей базы данных `gregs_list`?



Создание более сложных таблиц

Помните столбцы таблицы Грега? Мы записали их на карточке. Эта информация понадобится вам для построения команды **CREATE TABLE**.

Команда `CREATE TABLE` превратит это...

...вот в это



<code>last_name</code>	<code>first_name</code>	<code>email</code>	<code>birthday</code>	<code>profession</code>	<code>location</code>	<code>status</code>	<code>interests</code>	<code>seeking</code>

МОЗГОВОЙ ШТУРМ

Найдите два важных отличия между именами столбцов на карточке и в таблице. Почему они важны?

Посмотрите, как просто пишется код SQL

Вы уже знаете, что для создания таблицы данные необходимо разбить на категории. Затем вы подбираете подходящий тип данных и длину каждого столбца. После этого написать код SQL будет совсем несложно.

Возьми в руку карандаш

Слева записан код команды CREATE TABLE для новой базы данных Грегга. Попробуйте догадаться, что делает каждая строка команды CREATE TABLE. Также запишите пример данных, которые будут храниться в каждом столбце.

```
CREATE TABLE my_contacts
(
  last_name VARCHAR(30),
  first_name VARCHAR(20),
  email VARCHAR(50),
  birthday DATE,
  profession VARCHAR(50),
  location VARCHAR(50),
  status VARCHAR(20),
  interests VARCHAR(100),
  seeking VARCHAR(100)
);
```


Возьми в руку карандаш

Решение

Ниже приведены описания каждой строки команды CREATE TABLE и примеры данных для каждого типа столбца.

```
CREATE TABLE my_contacts
```

```
(
```

```
  last_name VARCHAR(30),
```

```
  first_name VARCHAR(20),
```

```
  email VARCHAR(50),
```

```
  birthday DATE,
```

```
  profession VARCHAR(50),
```

```
  location VARCHAR(50),
```

```
  status VARCHAR(20),
```

```
  interests VARCHAR(100),
```

```
  seeking VARCHAR(100)
```

```
);
```

Создает таблицу с именем 'my_contacts'	
Начинает список столбцов	
Добавляет столбец с именем 'last_name', вмещающий до 30 символов	'Андерсон'
Добавляет столбец с именем 'first_name', вмещающий до 20 символов	'Джиллиан'
Добавляет столбец с именем 'email', вмещающий до 50 символов	'jill_anderson@breakneckpizza.com'
Добавляет столбец с именем 'birthday', в котором хранится дата	'1980-05-09'
Добавляет столбец с именем 'profession', вмещающий до 50 символов	'Писатель'
Добавляет столбец с именем 'location', вмещающий до 50 символов	'Пало-Альто, Калифорния'
Добавляет столбец с именем 'status', вмещающий до 20 символов	'Не замужем'
Добавляет столбец с именем 'interests', вмещающий до 100 символов	'Каяк, террапиум'
Добавляет столбец с именем 'seeking', вмещающий до 100 символов	'Друзья'
Завершает список столбцов, а «>» завершает команду	

Наконец-то создаем таблицу my_contacts

Теперь вы точно знаете, что делает каждая строка, и можете ввести команду **CREATE TABLE** для создания таблицы. Команду можно вводить по строкам, копируя ее из приведенного выше определения.

А можно ввести все в одной длинной строке:

```
CREATE TABLE my_contacts(last_name VARCHAR(30), first_name VARCHAR(20), email VARCHAR(50), birthday DATE, profession VARCHAR(50), location VARCHAR(50), status VARCHAR(20), interests VARCHAR(100), seeking
```

Какой бы способ вы ни выбрали, прежде чем нажимать RETURN после «;», убедитесь в том, что вы не пропустили ни одного символа:

last_name VARCHAR(3) — совсем не то же самое, что lastname VARCHAR(30)!

Поверьте, это та самая команда. Просто мы записали ее **ОЧЕНЬ МЕЛКИМ** шрифтом, чтобы она поместилась на странице!

Таблица готова

Обратите внимание: нажатие RETURN после символа «;» завершает ввод команды и приказывает РСУБД выполнить ее.

```
File Edit Window Help AllDone
> CREATE TABLE my_contacts
-> (
->   last_name VARCHAR(30),
->   first_name VARCHAR(20),
->   email VARCHAR(50),
->   birthday DATE,
->   profession VARCHAR(50),
->   location VARCHAR(50),
->   status VARCHAR(20),
->   interests VARCHAR(100),
->   seeking VARCHAR(100)
-> );
Query OK, 0 rows affected (0.07 sec)
```

И что же, все данные должны храниться в столбцах VARCHAR или DATE?



Вообще-то вам понадобится еще несколько типов для других видов данных — например для чисел.

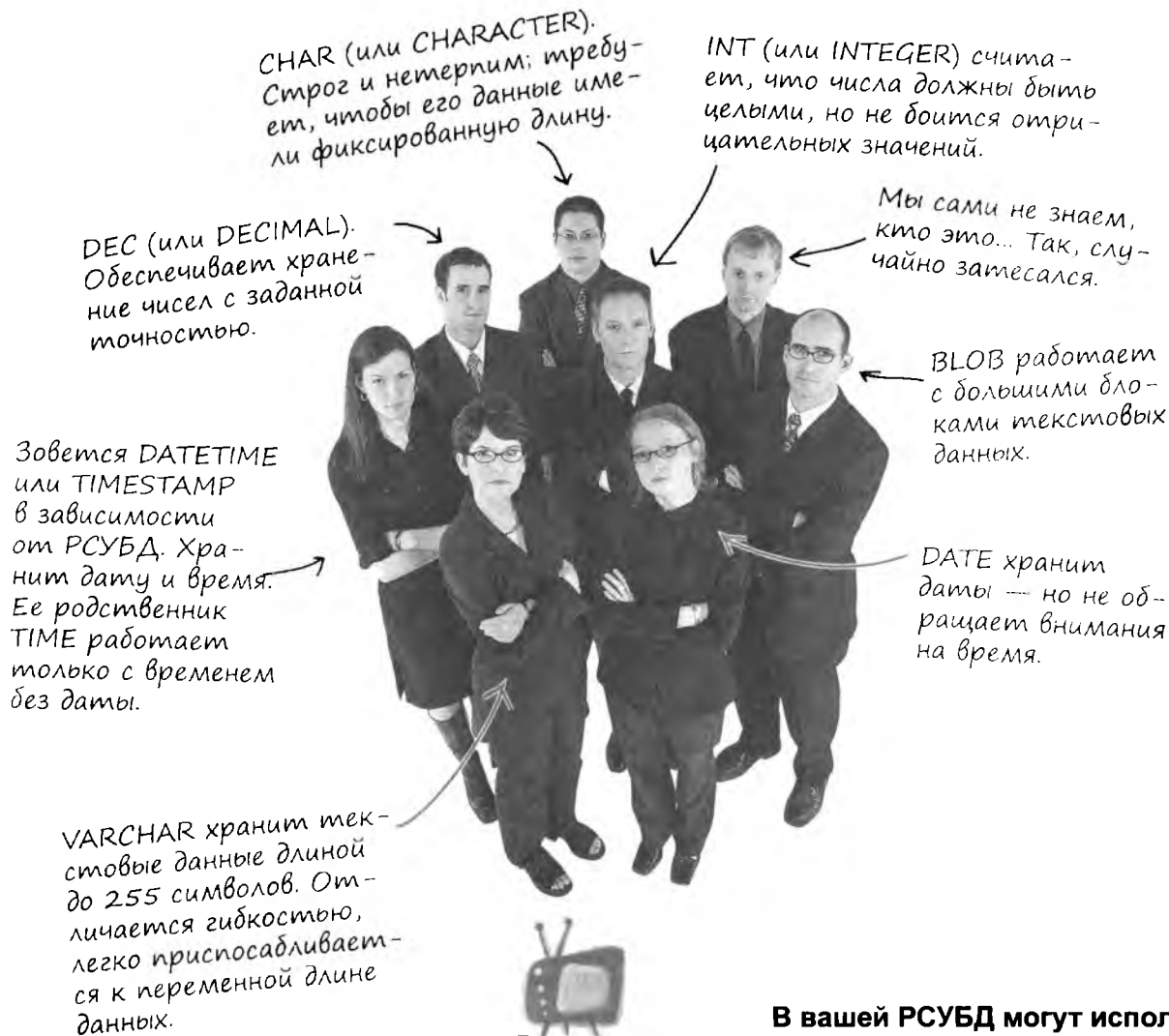
Предположим, в нашу таблицу добавляется столбец с ценой пончиков. Хранить числовые данные в столбце VARCHAR неудобно. Значения таких столбцов интерпретируются как текст, и с ними нельзя будет выполнять математические операции. Однако существуют и другие типы данных, с которыми мы пока не сталкивались...

МОЗГОВОЙ ШТУРМ

Прежде чем двигаться дальше, приведите пару примеров данных, для хранения которых недостаточно типов VARCHAR или DATE.

Знакомство с типами данных

Перед вами еще несколько полезных типов данных. Их работа — хранить ваши данные без искажения. С VARCHAR и DATE вы уже знакомы; пора познакомиться с остальными.



Будьте осторожны!

В вашей РСУБД могут использоваться другие имена типов!

К сожалению, общепринятой системы имен типов не существует.

В вашей конкретной РСУБД некоторые типы могут называться по-другому. За информацией о правильных именах обращайтесь к документации РСУБД.

Выберите наиболее подходящий тип данных для каждого столбца... А заодно заполните другие недостающие данные.

Количество цифр в целой и дробной части.

Имя столбца	Описание	Пример	Наиболее подходящий тип
price	Цена продаваемого товара	5678.39	DEC (6, 2) ←
zip_code	От 5 до 10 символов		
atomic_weight	Атомный вес элемента (с точностью до 6 цифр)		
comments	Большой блок текста (длиннее 255 символов)	Джо, я на собрании акционеров. В демOVERсии по экрану летают резиновые утки. Это что, штука такая? На премию можешь не рассчитывать...	
quantity	Количество единиц товара на складе		
tax_rate	Ставка налога	3.755	
book_title	Название книги	Head First SQL	
gender	Один символ — М или Ж		CHAR (1)
phone_number	Десять цифр без знаков препинания	2105552367	
state	Двухсимвольное сокращение штата	TX, CA	
anniversary	Дата рождения	11/22/2006	DATE
games_won	Количество побед в игре		INT
meeting_time	Время (и день) встречи	10:30 a.m. 4/12/2020	

Часть Задаваемые Вопросы

В: Почему мы не храним все текстовые данные в столбцах типа BLOB?

О: По соображениям эффективности. Столбец VARCHAR или CHAR имеет фиксированный размер, не более 256 символов. Столбец BLOB занимает намного больше памяти. С увеличением объема базы данных может кончиться место на жестком диске. Кроме того, со значениями BLOB нельзя выполнять некоторые важные строковые операции,

доступные для VARCHAR и CHAR (но об этом позднее).

В: Зачем нужны разные числовые типы INT и DEC?

О: Опять же по соображениям эффективности. Оптимальный выбор типа данных для каждого столбца таблицы уменьшает ее размер и ускоряет работу с данными.

В: Это все? Других типов нет?

О: Есть, но эти типы самые важные. Конкретный набор поддерживаемых типов данных также зависит от РСУБД, поэтому за дополнительной информацией следует обращаться к документации. Также рекомендуем книгу «SQL in a Nutshell» (O'Reilly) — это отличный справочник, в котором описаны основные различия между разными РСУБД.

* КТО И ЧТО ДЕЛАЕТ? *

В индексе может быть меньше 10 цифр, поэтому мы использовали VARCHAR для экономии места. Так можно было выбрать тип CHAR с фиксированной длиной значения.

Выберите наиболее подходящий тип данных для каждой строки... А заодно заполните другие недостающие данные.

Имя столбца	Описание	Пример	Наиболее подходящий тип
price	Цена продаваемого товара	5678.39	DEC(5,2)
zip_code	От 5 до 10 символов	90210-0010	VARCHAR(10) ←
atomic_weight	Атомный вес элемента (с точностью до 6 цифр)	4.002602	DEC(10, 6)
comments	Большой блок текста (более 255 символов)	Джо, я на собрании акционеров. В демо-версии по экрану летают резиновые утки. Это что, шутка такая? На премию можешь не рассчитывать...	BLOB
quantity	Количество единиц товара на складе	239	INT
tax_rate	Ставка налога	3.755	DEC(4, 2)
book_title	Название книги	Head First SQL	VARCHAR(50)
gender	Один символ – М или Ж	M	CHAR(1)
phone_number	Десять цифр без знаков препинания	2105552367	CHAR(10) ←
state	Двухсимвольное сокращение штата	TX, CA	CHAR(2)
anniversary	Дата рождения	22/11/2006	DATE
games_won	Количество побед в игре	15	INT
meeting_time	Время (и день) встречи	10:30 4/12/2020	DATETIME

↑
Тип TIMESTAMP обычно используется для сохранения текущего времени. Тип DATETIME лучше подходит для будущих событий.

↑
Телефонный номер имеет фиксированную длину. И мы рассматриваем его как строку текста, потому что с номером не нужно выполнять математические вычисления.



- Прежде чем создавать таблицу, разбейте данные на категории. Уделите особое внимание выбору типа каждого столбца.
- Создайте базу данных, в которой будут храниться все ваши таблицы, командой **CREATE DATABASE**.
- Используйте команду **USE DATABASE**, чтобы получить доступ к базе данных и перейти к созданию таблиц.
- Таблицы создаются командой **CREATE TABLE** с указанием имен столбцов и их типов данных.
- Основные типы данных столбцов: **CHAR**, **VARCHAR**, **BLOB**, **INT**, **DEC**, **DATE** и **DATETIME**. Каждый тип устанавливает свои ограничения для хранящихся в нем данных.



Минутку, а где таблица, которую я только что создала в базе данных `gregs_list`? Хочу убедиться в том, что все было сделано верно.

Отличная мысль, проверять себя необходимо.

Чтобы посмотреть, как выглядит созданная вами таблица `my_contacts`, воспользуйтесь командой **DESC** для вывода ее описания:

DESC my_contacts;

DESC — сокращение от DESCRIBE (вывести описание)

Попробуйте выполнить эту команду.

```
File Edit Window Help DescTidy
> DESC my_contacts;
```

Описание таблицы

Результат выполнения команды DESC выглядит примерно так:

На эти атрибуты пока не обращайте внимания; вскоре мы вернемся к ним.

```
File Edit Window Help DescTidy
> DESC my_contacts;
+-----+-----+-----+-----+-----+-----+
| Column      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| last_name   | varchar(30)   | YES  |     | NULL    |      |
| first_name  | varchar(20)   | YES  |     | NULL    |      |
| email       | varchar(50)   | YES  |     | NULL    |      |
| birthday    | date          | YES  |     | NULL    |      |
| profession  | varchar(50)   | YES  |     | NULL    |      |
| location    | varchar(50)   | YES  |     | NULL    |      |
| status      | varchar(20)   | YES  |     | NULL    |      |
| interests   | varchar(100)  | YES  |     | NULL    |      |
| seeking     | varchar(100)  | YES  |     | NULL    |      |
+-----+-----+-----+-----+-----+-----+
9 rows in set (0.07 sec)
```



Я забыл включить в таблицу один важный столбец. Это еще не поздно сделать?

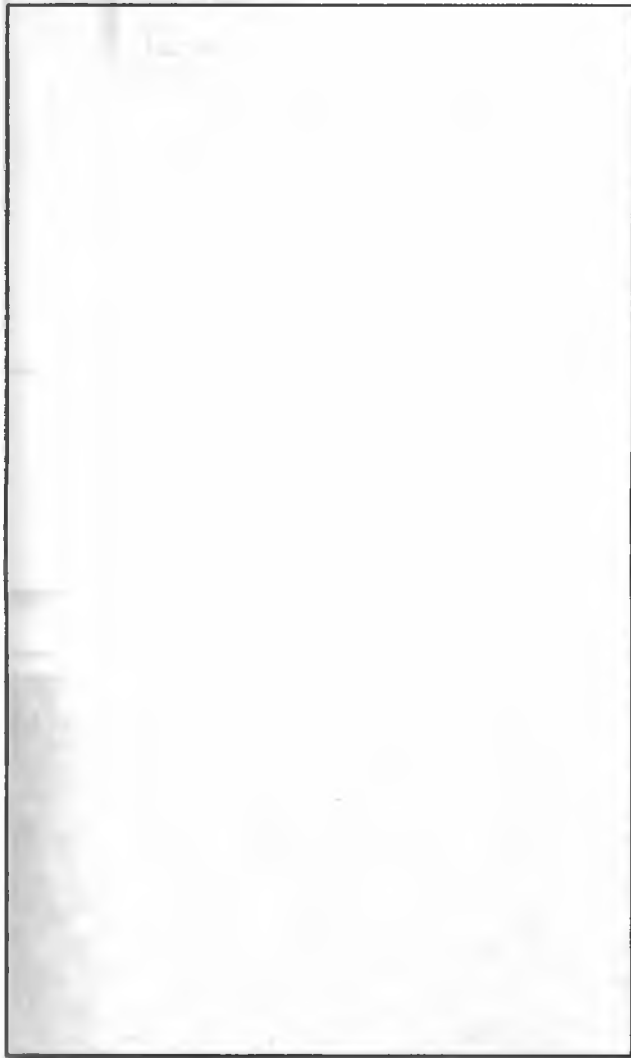
МОЗГОВОЙ ШТУРМ

А что думаете об этом вы? Какие проблемы могут возникнуть при добавлении нового столбца?



Магниты с Кодами

Код создания базы данных и таблицы с новым столбцом `gender` полностью перепутаны. Сможете ли вы расставить фрагменты в правильном порядке? Некоторые круглые скобки и символы «;» упали на пол. Они слишком малы, чтобы их подбирать — добавьте их по своему усмотрению!



`email VARCHAR(50)`

`birthday DATE`

`USE gregs_list`

`first_name VARCHAR(20)`

`last_name VARCHAR(30)`

`interests VARCHAR(100)`

`seeking VARCHAR(100)`

`status VARCHAR(20)`

`CREATE DATABASE gregs_list`

`profession VARCHAR(50)`

`location VARCHAR(50)`

`CREATE TABLE my_contacts`

`gender CHAR(1)`

Когда работа будет закончена, попробуйте ввести новую команду `CREATE TABLE` в консоли SQL для создания таблицы с новым столбцом `gender`!



Магниты с Кодами

Вы должны были восстановить код создания базы данных и таблицы с новым столбцом gender.

gregs_list уже существует.

Восстановленный код команды. Проверьте свой ответ и читайте дальше...

Нельзя заново создать существующую таблицу или базу данных!

А вы попытались ввести команду CREATE TABLE? Тогда вы, наверное, уже знаете, что команда из этого упражнения не позволит создать новый столбец.

Если вы ввели команду в своей консоли, вероятно, результат выглядел примерно так:

```
CREATE DATABASE gregs_list
USE gregs_list
CREATE TABLE my_contacts
(
  last_name VARCHAR(20),
  first_name VARCHAR(30),
  email VARCHAR(50),
  birthday DATE,
  gender CHAR(1),
  profession VARCHAR(50),
  location VARCHAR(50),
  status VARCHAR(20),
  interests VARCHAR(100),
  seeking VARCHAR(100)
);
```

Новый столбец gender.

Ой! Получаем сообщение об ошибке. Похоже, таблица не создалась.

```
File Edit Window Help OhCrap!
> CREATE TABLE my_contacts
-> (
->   last_name VARCHAR(30),
->   first_name VARCHAR(20),
->   email VARCHAR(50),
->   gender CHAR(1),
->   birthday DATE,
->   profession VARCHAR(50),
->   location VARCHAR(50),
->   status VARCHAR(20),
->   interests VARCHAR(100),
->   seeking VARCHAR(100)
-> );
ERROR 1050 (42S01): Table 'my_contacts' already exists
```

Задаваемые Вопросы

В: Почему в упражнении «Развлечения с магнитами» я получаю ошибку?

О: Нельзя создать заново уже существующую таблицу. И после того как база данных будет создана, ее не нужно создавать повторно. Еще одна часто встречающаяся ошибка — пропущенный завершитель «;». Также проверьте, нет ли опечаток в ключевых словах SQL.

В: Почему после определения столбца «seeking VARCHAR(100)» нет запятой, как у других столбцов?

О: Сразу за определением «seeking» следует закрывающая круглая скобка. Она сообщает РСУБД о том, что список завершен, поэтому запятая здесь не нужна.

В: Можно ли добавить в таблицу забытый столбец или все придется делать заново?

О: Придется делать заново, но прежде чем создавать таблицу с добавленным столбцом gender, необходимо сначала удалить старую таблицу. Так как в таблице пока нет данных, просто удалите старую таблицу и начните заново.

В: А если у меня есть таблица с данными и в нее нужно добавить столбец? Можно ли сделать это без удаления всей таблицы и повторения ввода?

О: Хороший вопрос! Да, таблицу можно изменить без уничтожения хранящихся в ней данных. Мы еще рассмотрим эту возможность, а пока наша таблица пуста, проще удалить ее и создать заново.

Так команду CREATE TABLE придется набирать повторно? Если бы команды SQL можно было вводить в текстовом редакторе (скажем, Блокнот или TextEdit), это сэкономило бы нам немало времени и сил.



Очень правильная мысль. Мы рекомендуем почаще использовать текстовый редактор во время чтения книги.

Это позволит вам копировать и вставлять команды в консоли SQL, и вам не придется вводить всю команду заново. Кроме того, вы можете копировать и редактировать старые команды SQL для создания новых команд.

Долой старые таблицы!

- 1 Уничтожить старую таблицу куда проще, чем создать новую. Введите простую команду:

Команда удаления
таблицы...

...и имя удаляемой
таблицы.

И не забудьте
завершитель «;».

```
DROP TABLE my_contacts;
```

```
File Edit Window Help ByeByeTable
> DROP TABLE my_contacts;
Query OK, 0 rows affected (0.12 sec)
```

Команда **DROP TABLE** работает независимо от того, есть в таблице данные или нет, поэтому использовать ее следует **ОЧЕНЬ ВНИМАТЕЛЬНО**. Удаленная таблица пропадает навсегда вместе со всеми данными, которые в ней были.

Команда DROP TABLE удаляет таблицу со всеми данными!

- 2 Теперь можно ввести новую команду **CREATE TABLE**:

```
File Edit Window Help Success
> CREATE TABLE my_contacts
  -> (
  ->   last_name VARCHAR(30),
  ->   first_name VARCHAR(20),
  ->   email VARCHAR(50),
  ->   gender CHAR(1),
  ->   birthday DATE,
  ->   profession VARCHAR(50),
  ->   location VARCHAR(50),
  ->   status VARCHAR(20),
  ->   interests VARCHAR(100),
  ->   seeking VARCHAR(100)
  -> );
Query OK, 0 rows affected (0.05 sec)
```

На этот раз
все получилось.

Компания ключевых слов и типов данных SQL, облаченных в маскарадные костюмы, развлекается игрой «Кто я?». Игрок дает подсказку, а остальные на основании сказанного им пытаются угадать, кого он изображает. Будем считать, что игроки всегда говорят правду о себе. Если сказанное ими может относиться сразу к нескольким персонажам, перечислите всех, к кому может относиться их высказывание. Заполните пропуски справа именами одного или нескольких участников.

Сегодняшние участники:

CREATE DATABASE, USE DATABASE, CREATE TABLE, DESC, DROP TABLE, CHAR, VARCHAR, BLOB, DATE, DATETIME, DEC, INT

Поможем с хранением чисел.

Занимаюсь устранением нежелательных таблиц.

Специализируюсь на вопросах Да/Нет.

Помогу запомнить день рождения знакомого.

Во мне хранятся все таблицы.

Числа — это хорошо, но я ненавижу дроби.

Люблю длинные, подробные объяснения.

Место для хранения ВСЕГО.

Без меня таблица не могла бы существовать.

Точно знаю, когда вам нужно явиться к врачу на следующей неделе.

Сохранить денежную сумму? Без проблем.

Вывожу описание формата таблицы.

Без нас вы вообще не сможете создать таблицу.

Кто я?



Имя

→ Ответы на с. 86.

дальше ▶ 69



Код под микроскопом

Ладно, моя таблица готова. И как теперь перенести данные с карточек в таблицу?



Для добавления данных в таблицу используется команда INSERT.

На приведенной ниже схеме показано, что делает каждая из частей команды. Значения во второй группе скобок должны следовать *в том же порядке, что и имена столбцов*.

Ниже приведена не реальная команда, а «заготовка» — условный шаблон, демонстрирующий формат команды INSERT.

Команда начинается с ключевого слов `INSERT INTO`.

Имя таблицы (в базе данных Грега — `my_contacts`).

Список имен столбцов, разделенных запятыми. Как вы уже знаете, в списке Грега содержатся столбцы с именами `first_name`, `last_name`, `email` и т. д.

Имена других столбцов (запятая после последнего столбца не нужна).

INSERT INTO имя_таблицы (столбец1, столбец2, ...)

VALUES ('значение1', 'значение2', ...)

Еще одно ключевое слово; сообщает, что дальше следует список значений столбцов.

Список значений, разделенных запятыми. В базе данных Грега список содержит данные с карточек.

Текстовые данные всегда заключаются в апострофы, даже отдельные символы (например, 'M').

Другие значения (запятая после последнего значения не нужна).

Как обычно, завершается символом

ВАЖНО: значения должны следовать в том же порядке, что и имена столбцов.

КТО И ЧТО ДЕЛАЕТ?

Прежде чем составлять команду INSERT, необходимо установить соответствие между именами столбцов и значениями.

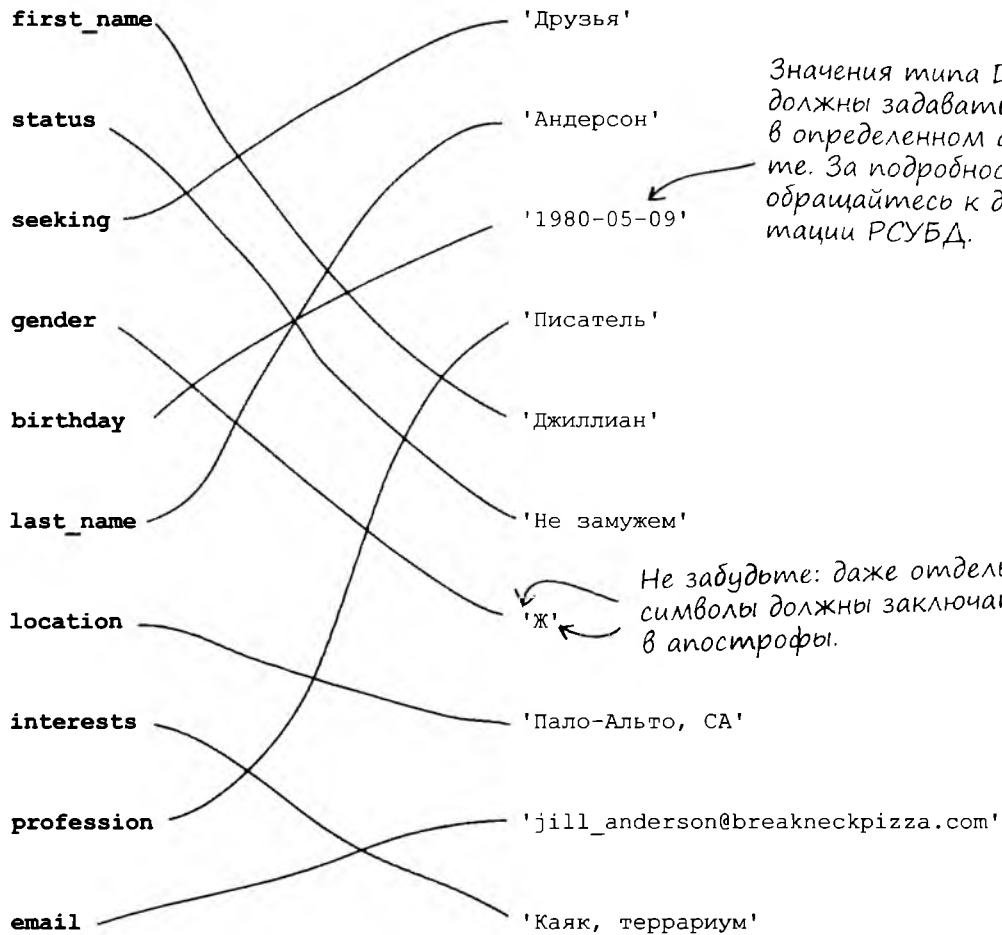
СТОЛБЦЫ	ЗНАЧЕНИЯ
<code>first_name</code>	'Друзья'
<code>status</code>	'Андерсон'
<code>seeking</code>	'1980-05-09'
<code>gender</code>	'Писатель'
<code>birthday</code>	'Джиллиан'
<code>last_name</code>	'Не замужем'
<code>location</code>	'Ж'
<code>interests</code>	'Пало-Альто, Калифорния'
<code>profession</code>	'jill_anderson@breakneckpizza.com'
<code>email</code>	'Каяк, террариум'

* КТО И ЧТО ДЕЛАЕТ? *

Прежде чем составлять команду INSERT, необходимо установить соответствие между именами столбцов и значениями.

СТОЛБЦЫ

ЗНАЧЕНИЯ



Значения типа DATE должны задаваться в определенном формате. За подробностями обращайтесь к документации РСУБД.

Не забудьте: даже отдельные символы должны заключаться в апострофы.

Создание команды INSERT

Имена столбцов перечисляются в первой паре скобок и разделяются запятыми.

Нажмите RETURN перед открывающей скобкой — это упростит чтение кода в окне консоли.

```
INSERT INTO my_contacts
```

```
(last_name, first_name, email, gender, birthday,  
profession, location, status, interests,  
seeking)
```

```
VALUES
```

```
('Андерсон', 'Джиллиан', 'jill_anderson@  
breakneckpizza.com', 'Ж', '1980-05-09',  
'Писатель', 'Пало-Альто, СА', 'Не замужем',  
'Каяк, террариум', 'Друзья');
```

Нажмите RETURN после закрывающей скобки списка столбцов, а потом после ключевого слова VALUES — код, разбитый на строки, лучше читается.

Значения столбцов перечисляются во второй паре скобок и разделяются запятыми.

Значения столбцов VARCHAR, CHAR, DATE или BLOB заключаются в апострофы.



Будьте осторожны!

Порядок важен!

Порядок перечисления значений должен точно совпадать с порядком перечисления столбцов.



Попробуйте сами

Мы рассмотрели один из способов добавления строк в таблицу. Попробуйте выполнить эту команду в РСУБД. Сначала введите ее в текстовом редакторе, чтобы вам не пришлось набирать весь текст заново, если вы ошибетесь при вводе. Будьте особенно внимательны с апострофами и запятыми. Запишите ответ РСУБД в этой строке:



Вы только что сказали, что значения **CHAR, VARCHAR, DATE** и **BLOB** заключаются в **апострофы** в командах **INSERT**. Значит, для **числовых значений** (таких, как **DEC** и **INT**) **апострофы не нужны?**

Совершенно верно.

Ниже приведена команда **INSERT** для таблицы с данными о покупке пончиков. Обратите внимание: числовые значения столбцов **dozens** и **price** записываются без апострофов.

*Столбец **dozens** имеет тип данных **INT**.*

*Столбец **price** имеет тип **DEC(4,2)**; это означает, что его значения состоят из четырех цифр с двумя цифрами в дробной части.*

```
INSERT INTO doughnut_purchases  
(donut_type, dozens, topping, price)  
VALUES  
('с вареньем', 3, 'sprinkles', 3.50);
```

*Значения столбцов **dozens** и **price** записываются без апострофов!*



Ваша РСУБД сообщает об ошибках в командах, но эти сообщения порой выглядят весьма туманно. Взгляните на каждую из приведенных ниже команд INSERT. Попробуйте найти ошибку в каждой из команд, затем введите ее и посмотрите, что скажет РСУБД.

```
INSERT INTO my_contacts
```

```
(last_name, first_name, email, gender, birthday, profession, location, status, interests, seeking) VALUES ('Андерсон', 'Джиллиан', 'jill_anderson@breakneckpizza.com', 'Ж', '1980-05-09', 'Писатель', 'Не замужем', 'Каяк, террариум', 'Друзья');
```

Где ошибка?

Что говорит РСУБД:

```
INSERT INTO my_contacts
```

```
(last_name, first_name, gender, birthday, profession, location, status, interests, seeking) VALUES ('Андерсон', 'Джиллиан', 'jill_anderson@breakneckpizza.com', 'Ж', '1980-05-09', 'Писатель', 'Пало-Альто, СА', 'Не замужем', 'Каяк, террариум', 'Друзья');
```

Где ошибка?

Что говорит РСУБД:

```
INSERT INTO my_contacts
```

```
(last_name, first_name, email, gender, birthday, profession, location, status, interests, seeking) VALUES ('Андерсон', 'Джиллиан', 'jill_anderson@breakneckpizza.com', 'Ж', '1980-05-09', 'Писатель', 'Пало-Альто, СА', 'Не замужем', 'Каяк, рептилия', 'Друзья');
```

Где ошибка?

Что говорит РСУБД:

```
INSERT INTO my_contacts
```

```
(last_name, first_name, email, gender, birthday, profession, location, status, interests, seeking) VALUES ('Андерсон', 'Джиллиан', 'jill_anderson@breakneckpizza.com', 'Ж', '1980-05-09', 'Писатель', 'Пало-Альто, СА', 'Не замужем', 'Каяк, террариум', 'Друзья');
```

Где ошибка? Если на этой команде РСУБД «зависнет», попробуйте ввести

Что говорит РСУБД: сти апостроф и символ «;» после основного кода команды.

Возьми в руку карандаш



Решение

Ваша РСУБД сообщает об ошибках в командах, но эти сообщения порой выглядят весьма туманно. Взгляните на каждую из приведенных ниже команд INSERT. Попробуйте найти ошибку в каждой из команд, затем введите ее и посмотрите, что скажет РСУБД.

```
INSERT INTO my_contacts
```

```
(last_name, first_name, email, gender, birthday, profession, location, status, interests, seeking) VALUES ('Андерсон', 'Джиллиан', 'jill_anderson@breakneckpizza.com', 'Ж', '1980-05-09', 'Писатель', 'Не замужем', 'Каяк, террариум', 'Друзья');
```

Где ошибка? Нет значения location.

← В первом списке есть столбец location, а во втором списке нет значения этого столбца.

Что говорит РСУБД: ОШИБКА 1136 (21501): Количество столбцов не соответствует количеству значений (строка 1).

← Для разных ошибок выдаются одинаковые сообщения. Будьте внимательны: найти опечатку бывает очень сложно.

```
INSERT INTO my_contacts
```

```
(last_name, first_name, gender, birthday, profession, location, status, interests, seeking) VALUES ('Андерсон', 'Джиллиан', 'jill_anderson@breakneckpizza.com', 'Ж', '1980-05-09', 'Писатель', 'Пало-Альто, СА', 'Не замужем', 'Каяк, террариум', 'Друзья');
```

Где ошибка? Нет столбца email.

← На этот раз все значения заданы, но пропущен столбец email в списке столбцов.

Что говорит РСУБД: ОШИБКА 1136 (21501): Количество столбцов не соответствует количеству значений (строка 1).

```
INSERT INTO my_contacts
```

```
(last_name, first_name, email, gender, birthday, profession, location, status, interests, seeking) VALUES ('Андерсон', 'Джиллиан', 'jill_anderson@breakneckpizza.com', 'Ж', '1980-05-09', 'Писатель', 'Пало-Альто, СА', 'Не замужем', 'Каяк, рептилии', 'Друзья');
```

Где ошибка? Пропущена запятая.

← Нет запятой между значениями столбцов profession и location.

Что говорит РСУБД: ОШИБКА 1136 (21501): Количество столбцов не соответствует количеству значений (строка 1).

```
INSERT INTO my_contacts
```

```
(last_name, first_name, email, gender, birthday, profession, location, status, interests, seeking) VALUES ('Андерсон', 'Джиллиан', 'jill_anderson@breakneckpizza.com', 'Ж', '1980-05-09', 'Писатель', 'Пало-Альто, СА', 'Не замужем', 'Каяк, террариум', 'Друзья');
```

Где ошибка? Последнее значение не завершается апострофом.

Что говорит РСУБД: ОШИБКА 1064 (42000): Ошибка в синтаксисе SQL. Обратитесь к документации по вашей версии сервера MySQL за описанием синтаксиса, который должен использоваться рядом с " в строке 4

Модификации команды INSERT

У синтаксиса **INSERT** существует ряд модификаций, о которых следует знать.

1 Изменение порядка столбцов

Порядок столбцов можно изменить — при условии, что значения будут перечисляться в соответствующем порядке!

```
INSERT INTO my_contacts
(interests, first_name, last_name, gender, email, birthday,
profession, location, status, seeking)
VALUES
('Каяк, террариум', 'Джиллиан', 'Андерсон', 'Ж',
'jill_anderson@breakneckpizza.com', '1980-05-09', 'Писатель',
'Пало-Альто, СА', 'Не замужем', 'Друзья');
```

Порядок перечисления столбцов изменился. А теперь посмотрите на значения: они перечисляются в том же порядке.

2 Не указаны имена столбцов

Список столбцов можно опустить, но тогда **все** значения должны быть указаны **в порядке перечисления столбцов при создании таблицы**. (Если не уверены, проверьте порядок в команде на с. 73.)

```
INSERT INTO my_contacts
VALUES
('Андерсон', 'Джиллиан', 'jill_anderson@breakneckpizza.com', 'Ж', '1980-05-09', 'Писатель', 'Пало-Альто, СА', 'Не замужем', 'Каяк, террариум', 'Друзья');
```

Имена столбцов можно не указывать, но тогда необходимо задать ВСЕ значения в ТОЧНОМ ПОРЯДКЕ их следования в таблице!

3 Не указаны некоторые значения

В списке значений отсутствуют данные некоторых столбцов.

```
INSERT INTO my_contacts
(last_name, first_name, email)
VALUES
('Андерсон', 'Джиллиан', 'jill_anderson@breakneckpizza.com');
```

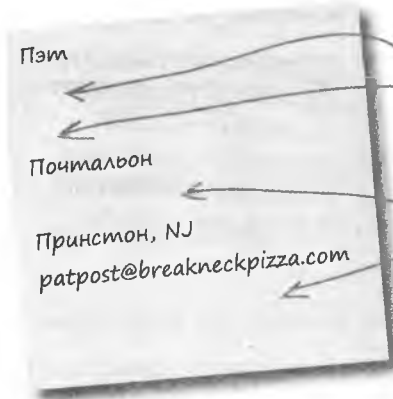
Команда заполняет лишь часть данных записи. Так как РСУБД не знает, какие именно значения пропущены, для всех заданных значений необходимо явно указать имена столбцов.



Как вы думаете, что хранится в столбцах, которым не были присвоены значения?

Столбцы без значений

Давайте вставим в базу данных my_contacts данные из неполной карточки:



Нет данных о фамилии и дате рождения, да и с полом нет особой уверенности.

Также отсутствуют значения столбцов status, interests и seeking.

COLUMNS:	VALUES:
last_name	?
first_name	'Пэт'
email	'patpost@breakneckpizza'
gender	?
birthday	?
profession	'Почтальон'
location	'Принстон, NJ'
status	?
interests	?
seeking	?

Так как на карточке часть данных отсутствует, Грегу придется ввести неполную информацию. Но в этом нет ничего страшного, недостающие сведения можно будет добавить позже.

Здесь используется версия INSERT с неполным набором значений, позволяющая задать только известные значения столбцов.

```
INSERT INTO my_contacts
(first_name, email, profession, location)
VALUES
('Пэт', 'patpost@breakneckpizza.com',
'Почтальон', 'Принстон, NJ');
```

```
File Edit Window Help MoreDataPlease
> INSERT INTO my_contacts (first_name, email, profession,
location) VALUES ('Пэт', 'patpost@breakneckpizza.com',
'Почтальон', 'Принстон, NJ');
Query OK, 1 row affected (0.02 sec)
```

Столбцы, значения которых
не были заданы, содержат NULL.

last_name	first_name	email	gender	birthday	profession	location	status	interests	seeking
Андрерсон	Джиллиан	jill_anderson@breakneckerpizza.com	Ж	1980-05-09	Писатель	Пало-Альто, CA	Не замужем	Каяк, террариум	Да
NULL	Пэт	patrost@breakneckerpizza.com	NULL	NULL	Почтальон	Принстон, NJ	NULL	NULL	NULL

Команда SELECT читает данные из таблицы

Хотите посмотреть, что у вас получилось? Команда DESC уже не подойдет — она выводит только *структуру* таблицы, но не хранящуюся в ней информацию. А для вывода данных, хранящихся в таблице, используется команда выборки SELECT.

Мы хотим вы-
брать все дан-
ные из таблицы...

... звездочка именно
это и означает:
выбрать ВСЕ.

Имя
таблицы.

```
SELECT * FROM my_contacts;
```



РАССЛАБЬТЕСЬ

Для нас пока несуще-
ственно, как работает
команда SELECT.

Эта команда будет подробно рассмотрена в гла-
ве 2. А пока просто расположитесь поудобнее и
насладитесь красотой созданной вами таблицы.

Попробуйте сами. Вероятно, для просмотра результатов
окно консоли придется растянуть по горизонтали.

МОЗГОВОЙ ШТУРМ

Итак, в столбцах, которым не присвоены
значения, выводится NULL. Как вы думаете,
что это *значит*?

дальше ▶



ОТКРОВЕННО ОБ SQL

Интервью недели:

Признания NULL

Head First: Добро пожаловать, NULL. Честно говоря, немного странно видеть вас здесь. Мы даже не думали, что вы действительно существуете. По слухам, вы — просто ноль, то есть вообще ничто.

NULL: И вы поверили этой грязной лжи? Да, я здесь, и я абсолютно реален! А вы, значит, думаете, что я — ничто, пустое место?

Head First: Спокойно, спокойно... Просто вы всегда появляетесь там, где нет значения...

NULL: Еще бы — уж лучше я, чем ноль или, скажем, пустая строка.

Head First: Пустая строка?..

NULL: Значение из двух апострофов, между которыми нет ни одного символа. Оно все равно рассматривается как текстовая строка, но имеющая нулевую длину. Скажем, как если бы столбцу `first_name` в таблице `my_contacts` было присвоено значение "".

Head First: Выходит, вы не просто хитроумный синоним для «ничто»?

NULL: Говорю же, я не «ничто»! Я есть! Просто мое состояние немного... неопределенное.

Head First: Выходит, если сравнить вас с нулем или пустой строкой, вы не будете им равны?

NULL: Нет! Я никогда не равен нулю. Более того, я даже не равен какому-нибудь другому NULL. Сравнить два NULL между собой нельзя. Столбец может содержать NULL, но его значение никогда не **равно** NULL, потому что NULL — неопределенное значение! Понимаете?

Head First: Успокойтесь и давайте разберемся. Вы не равны нулю, вы не равны пустой строке. И вы даже не равны самому себе! Как-то это странно, вы не находите?

NULL: Знаю, это нелегко понять. Считайте, что я не определен. Я — как содержимое закрытой коробки, в которой может лежать все что угодно. Сравнить одну закрытую коробку с другой нельзя, потому что вы не знаете, что лежит в этих коробках. Они вообще могут быть пустыми. Что внутри — неизвестно.

Head First: Говорят, иногда ваше присутствие нежелательно. В некоторых ситуациях NULL создает проблемы.

NULL: Признаю, бывали и неприятные ситуации. Некоторые столбцы всегда должны иметь определенное значение — например, столбец фамилии. Бессмысленно создавать в таблице описание человека с неопределенной фамилией.

Head First: Так вы не будете появляться там, где вас не должно быть?

NULL: Точно! Только скажите! При создании таблицы и определении столбцов укажите: в этом столбце NULL быть не должно. И я там не появлюсь.

Head First: Вообще-то вы не похожи на закрытую коробку.

NULL: С меня хватит. Извините, мне пора — у меня куча дел.

Управление NULL в таблицах

Некоторые столбцы в таблице всегда должны иметь определенное значение. Помните неполную карточку, в которой стояло только имя Пэт без фамилии? Когда в вашей таблице еще два десятка записей с NULL вместо фамилии, найти такую запись будет трудно. К счастью, вы можете легко запретить присваивание NULL столбцам таблицы.

```
CREATE TABLE my_contacts
(
  last_name VARCHAR (30) NOT NULL,
  first_name VARCHAR (20) NOT NULL
);
```

Просто добавьте слова NOT NULL после типа данных.

Значение такого столбца должно быть указано в команде INSERT. В противном случае РСУБД выдаст сообщение об ошибке.

Возьми в руку карандаш



```
CREATE TABLE my_contacts
(
  last_name VARCHAR(30) NOT NULL,
  first_name VARCHAR(20) NOT NULL,
  email VARCHAR(50),
  gender CHAR(1),
  birthday DATE,
  profession VARCHAR(50),
  location VARCHAR(50),
  status VARCHAR(20),
  interests VARCHAR(100),
  seeking VARCHAR(100)
);
```

Взгляните на столбцы таблицы my_contacts в команде CREATE TABLE. Какие из них следует снабдить условием NOT NULL? Подумайте, какие столбцы никогда не должны содержать NULL, и обведите их кружком.

Мы уже выделили два таких столбца; разберитесь с остальными. Обращайте особое внимание на столбцы, которые позднее будут использоваться для поиска или содержащие уникальные значения.

Возьми в руку карандаш

Решение

```
CREATE TABLE my_contacts
(
  last_name VARCHAR(30) NOT NULL,
  first_name VARCHAR(20) NOT NULL,
  email VARCHAR(50),
  gender CHAR(1),
  birthday DATE,
  profession VARCHAR(50),
  location VARCHAR(50),
  status VARCHAR(20),
  interests VARCHAR(100),
  seeking VARCHAR(100)
);
```

Взгляните на столбцы таблицы `my_contacts` в команде `CREATE TABLE`. Какие из них следует снабдить условием `NOT NULL`? Подумайте, какие столбцы никогда не должны содержать `NULL`, и обведите их кружком.

Мы уже выделили два таких столбца; разберитесь с остальными. Обращайте особое внимание на столбцы, которые позднее будут использоваться для поиска или содержащие уникальные значения.

Все столбцы должны быть объявлены с ключевыми словами `NOT NULL`.

ВСЕ столбцы таблицы будут использоваться для поиска. С самого начала стоит позаботиться о том, чтобы все записи содержали полную информацию...

но если в таблице есть столбцы, которые будут заполняться позднее, для таких столбцов стоит разрешить значения `NULL`.

NOT NULL В Выходных данных DESC

А вот как будет выглядеть таблица my_contacts, если объявить все столбцы с ключевыми словами NOT NULL:

Команда создает таблицу, у которой все столбцы объявлены с NOT NULL.

Описание таблицы. Обратите внимание на слово NO в столбце NULL.

```
File Edit Window Help NoMoreNULLs
CREATE TABLE my_contacts
(
  last_name VARCHAR(30) NOT NULL,
  first_name VARCHAR(20) NOT NULL,
  email VARCHAR(50) NOT NULL,
  gender CHAR(1) NOT NULL,
  birthday DATE NOT NULL,
  profession VARCHAR(50) NOT NULL,
  location VARCHAR(50) NOT NULL,
  status VARCHAR(20) NOT NULL,
  interests VARCHAR(100) NOT NULL,
  seeking VARCHAR(100) NOT NULL
);
Query OK, 0 rows affected (0.01 sec)

> DESC my_contacts;
+-----+-----+-----+-----+-----+-----+
| Column      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| last_name   | varchar(30)   | NO   |     |         |       |
| first_name  | varchar(20)   | NO   |     |         |       |
| email       | varchar(50)   | NO   |     |         |       |
| gender      | char(1)       | NO   |     |         |       |
| birthday    | date          | NO   |     |         |       |
| profession  | varchar(50)   | NO   |     |         |       |
| location    | varchar(50)   | NO   |     |         |       |
| status      | varchar(20)   | NO   |     |         |       |
| interests   | varchar(100)  | NO   |     |         |       |
| seeking     | varchar(100)  | NO   |     |         |       |
+-----+-----+-----+-----+-----+-----+
10 rows in set (0.02 sec)
```

DEFAULT и значения по умолчанию

Если в столбце часто хранится какое-то одно конкретное значение, ему можно присвоить значение по умолчанию при помощи ключевого слова **DEFAULT**. Значение, следующее за **DEFAULT**, автоматически заносится в таблицу при каждом добавлении новой записи — *если не задано другое значение*. Значение по умолчанию должно соответствовать типу данных столбца.

Этот столбец ВСЕГДА должен содержать значение. Для этого мы не только объявляем его с ключевыми словами **NOT NULL**, но и присваиваем значение по умолчанию 1.

```
CREATE TABLE doughnut_list
(
  doughnut_name VARCHAR(10) NOT NULL,
  doughnut_type VARCHAR(6) NOT NULL,
  doughnut_cost DEC(3,2) NOT NULL DEFAULT 1.00
);
```

Значение может содержать до 3 цифр: одна до и две после запятой.

Это значение сохраняется в столбце `doughnut_cost`, если в команде `INSERT` не указано другое значение.

doughnut_list

doughnut_name	doughnut_type	doughnut_cost
Blooberry	с начинкой	2.00
Cinnamondo	пышки	1.00
Rockstar	хворост	1.00
Carameller	хворост	1.00
Appleblush	с начинкой	1.40

Так будет выглядеть таблица, если при вставке данных пончиков `Cinnamondo`, `Rockstar` и `Carameller` не указывать значение столбца `doughnut_cost`.

Условие DEFAULT заполняет пустые столбцы заданным значением.



- Для вывода описания структуры таблицы используется команда **DESC**.
- Команда **DROP TABLE** уничтожает таблицу со всем содержимым. Будьте внимательны!
- Для сохранения данных в таблице используется команда **INSERT**, которая существует в нескольких вариантах.
- **NULL** — неопределенное значение, которое не равно нулю или пустой строке. Для столбца, содержащего **NULL**, выполняется условие **IS NULL**, но при этом он не равен **NULL**.
- Столбцы, значение которых не указано в команде **INSERT**, по умолчанию инициализируются **NULL**.
- Чтобы запретить хранение **NULL** в столбце, используйте ключевые слова **NOT NULL** при создании таблицы.
- Условие **DEFAULT** определяет значение по умолчанию — если при заполнении таблицы значение столбца не указано, он автоматически заполняется этим значением.

NULL и NOT NULL

При создании базы данных следует знать, какие столбцы не должны принимать значение **NULL** — это упростит сортировку и поиск данных. Условие **NOT NULL** задается для столбцов при создании таблицы.

DEFAULT

Определяет значение по умолчанию для столбца; оно используется в том случае, если значение столбца не указано при вставке строки.



Новые инструменты

Мы подошли к концу главы 1. Вы научились создавать базы данных и таблицы и вставлять в них данные некоторых распространенных типов. Кроме того, вы знаете, как обеспечить наличие определенного значения у столбца.

CREATE TABLE

Команда создает таблицу, но для ее выполнения необходимо знать **ИМЕНА** и **ТИПЫ ДАННЫХ** столбцов. Они определяются на основе анализа информации, которая будет храниться в таблице.

DROP TABLE

Команда удаляет таблицу, при создании которой была допущена ошибка — но это следует делать до выполнения команд **INSERT**, заполняющих таблицу данными.

CREATE DATABASE

Команда создает базу данных, в которой хранятся все таблицы с данными.

USE DATABASE

Команда открывает базу данных для создания таблиц.

Компания ключевых слов и типов данных SQL, облаченных в маскарадные костюмы, развлекается игрой «Кто я?». Игрок дает подсказку, а остальные на основании сказанного им пытаются угадать, кого он изображает. Будем считать, что игроки всегда говорят правду о себе. Если сказанное ими может относиться сразу к нескольким персонажам, перечислите всех, к кому может относиться их высказывание. Заполните пропуски справа именами одного или нескольких участников.

Сегодняшние участники:

CREATE DATABASE, USE DATABASE, CREATE TABLE, DESC, DROP TABLE, CHAR, VARCHAR, BLOB, DATE, DATETIME, DEC, INT

Поможем с хранением чисел.

Занимаюсь устранением нежелательных таблиц.

Специализируюсь на вопросах Да/Нет.

Помогу запомнить день рождения знакомого.

Во мне хранятся все таблицы.

Числа — это хорошо, но я ненавижу дроби.

Люблю длинные, подробные объяснения.

Место для хранения ВСЕГО.

Без меня таблица не могла бы существовать.

Точно знаю, когда вам нужно явиться к врачу на следующей неделе.

Сохранить денежную сумму? Без проблем.

Вывожу описание формата таблицы.

Без нас вы вообще не сможете создать таблицу.



Имя

DEC, INT

DROP TABLE

CHAR(1)

DATE

CREATE DATABASE

INT

BLOB

CREATE TABLE

CREATE DATABASE

DATETIME

DEC

DESC

CREATE DATABASE,
USE DATABASE
DROP TABLE

Дополнил очки тем, добавил (1)

Выборка данных



```
SELECT * FROM gifts  
WHERE contents = «дорого»;
```

При работе с базами данных операция **выборки** обычно выполняется чаще, чем операция **вставки** данных в базу. В этой главе вы познакомитесь с могущественной командой **SELECT** и узнаете, как **получить доступ к важной информации**, которую вы сохранили в своих таблицах. Также вы научитесь использовать условия **WHERE**, **AND** и **OR** для избирательной выборки данных и предотвращения вывода *ненужных* данных.

Трудный поиск

Грег наконец-то перенес все данные из своей картотеки в таблицу `my_contacts`. Теперь ему хочется отдохнуть. Он раздобыл два билета на концерт и хочет пригласить одну из своих знакомых — девушку из Сан-Франциско.

Чтобы найти ее адрес электронной почты, Грег просматривает содержимое таблицы командой **SELECT** из главы 1.

```
SELECT * from my_contacts;
```



↗
Данные хранятся
в таблице Грега...
где-то.

СТАТЬ ГРЕГОМ



Теперь поставьте себя на место Грега. Просмотрите начало таблицы `my_contacts` на следующей странице и попробуйте найти в ней Энн из Сан-Франциско.

Таблица my_contacts состоит из многих столбцов. Здесь показано лишь начало таблицы.

last_name	first_name	email	gender
Андерсон	Джиллиан	jill_anderson@breakneckpizza.com	Ж
Иоффе	Кевин	joffe@simuduck.com	М
Ньюсам	Аманда	aman2luv@breakneckpizza.com	Ж
Гарсиа	Эд	ed99@b0tt0msup.com	М
Раундтри	Джо-Энн	jojoround@breakneckpizza.com	Ж
Вриггс	Крис	cbriggs@boards-r-us.com	М
Харт	Ллойд	hovercraft@breakneckpizza.com	М
Тот	Энн	Anne_Toht@leapinlimos.com	Ж
Уайли	Эндрю	andrewwiley@objectville.net	М
Паламбо	Том	palofmine@mightygumball.net	М
Райан	Аланна	angrypizate@breakneckpizza.com	Ж
Маккинни	Клей	clay@starbuzzcoffee.com	М
Микер	Энн	ann_mackar@chocoholic-inc.com	Ж
Пауэрс	Брайан	bp_honey_doit.com	М
Мэнсон	Энн	am66@objectville.net	М
Мендел	Дебра	dehmonster@breakneckpizza.com	Ж
Тедеско	Дженис	janistedesco@starbuzzcoffee.com	Ж
Талвар	Викрам	vikt@starbuzzcoffee.com	М
Швед	Джо	szwed_joe@objectville.net	М
Шеридан	Диана	sheridi@mightygumball.net	Ж
Сноу	Эдвард	snowman@tikibeanlounge.com	М
Отто	Глен	glenn0098@objectville.net	М
Харди	Энн	anneh@b0tt0msup.com	Ж
Дил	Мэри	nobigdeal@starbuzzcoffee.com	Ж
Джейгел	Энн	dreamgirl@breakneckpizza.com	Ж
Мелфи	Джеймс	drmelfi@b0tt0msup.com	М
Оливер	Ли	lee_oliver@weatherorama.com	М
Паркер	Энн	annepe@starbuzzcoffee.com	Ж
Риччи	Питер	ricciman@tikibeanlounge.com	М
Рено	Грейс	grace23@objectville.net	Ж
Мосс	Зелда	zelda@weatherorama.com	Ж
Дэй	Клиффорд	cliffnight@breakneckpizza.com	М
Волгер	Джойс	joyce@chocoholic-inc.com	Ж
Блант	Энн	anneblunt@breakneckpizza.com	Ж
Воллинг	Линди	lindy@tikibeanlounge.com	Ж
Гарес	Фред	fgares@objectville.net	М
Джейкобс	Энн	anne99@objectville.net	Ж

location
Пало-Альто
Сан-Хосе, CA
Сан-Франциско
Сан-Матео
Сан-Франциско
Остин, TX
Сан-Хосе, CA
Сан-Франциско
Нью-Йорк, NY
Принстон, NJ
Сан-Франциско
Нью-Йорк, NY
Сан-Франциско
Напа, CA
Сиэтл, WA
Нагез, MS
Лас-Вегас, NV
Пало-Альто
Нью-Йорк, NY
Финикс, AZ
Фарго, ND
Боулдер, CO
Сан-Франциско
Бостон, MA
Сан-Франциско
Даллас, TX
Сент-Луис, MO
Сан-Франциско
Рено, NV
Пало-Альто
Саннивейл, CA
Честер, NJ
Остин, TX
Сан-Франциско
Сан-Диего, CA
Сан-Хосе, CA
Сан-Хосе, CA

|| это еще не конец! У Грега МЕНЬ МНОГО карточек.

Стань Грегом. Ответ



Вы должны были представить себя на месте Грега, посмотреть начало таблицы `my_contacts`, найти всех Энн из Сан-Франциско и записать их имена, фамилии и адреса электронной почты.

Тот, Энн: Anne_Toht@leapinlimos.co

Харди, Энн: anneh@bottomsup.com

Паркер, Энн: annep@starbuzzcoffee.com

Блант, Энн: anneblunt@Bbreakneckpizza.com

Разные Энн и адреса их электронной почты.

Ищем контакт

Поиск занял слишком много времени и был исключительно нудным. Также существует весьма реальная опасность того, что Грег пропустил пару-тройку подходящих Энн, включая ту, которую он ищет.

Зная адреса электронной почты, Грег рассылает сообщения и получает ответы...

To: Тот, Энн <Anne_Toht@leapinlimos.com>
From: Грег <greg@gregsl.com>
Subject: Мы встречались в Starbuzz?

Я сейчас встречаюсь с отличным парнем, его зовут Тим Вудман. Мы встретились на студенческой вечеринке.

To: Блант, Энн <anneblunt@breakneckpizza.com>
From: Грег <greg@gregsl.com>
Subject: Мы встречались в Starbuzz?

Ковбой, ты-то мне и нужен! Заедь за мной в пять, зайдём куда-нибудь перекусить.

To: Харди, Энн <anneh@bottomsup.com>
From: Грег <greg@gregsl.com>
Subject: Мы встречались в Starbuzz?

Я не та Энн, которую ты ищешь, но я уверена, что она того стоит. Если что-то не срастется — напиши мне.

To: Parker, Anne <annep@starbuzzcoffee.com>
From: Грег <greg@gregsl.com>
Subject: Мы встречались в Starbuzz?

Конечно, я тебя помню! Жаль, что ты не написал раньше. Я уже договорилась с моим бывшим парнем, который хочет начать все заново.

МОЗГОВОЙ ШТУРМ

Сможете ли вы написать запрос SQL для выборки только тех записей, у которых столбец `first_name` содержит строку «Энн»?

Улучшенная команда SELECT

Следующая команда SELECT поможет Грегу отыскать данные Энн намного быстрее, чем при дотошном просмотре всей огромной таблицы. В этой команде мы используем условие **WHERE**, которое уточняет критерий отбора записей для РСУБД. Условие сужает результаты поиска, а команда возвращает только те записи, для которых это условие выполняется.

Знак = в условии WHERE означает, что каждое значение столбца first_name проверяется на равенство с текстом 'Энн'. Если два значения равны, то вся запись включается в результат выборки. Если нет — запись пропускается.

```
SELECT * FROM my_contacts
WHERE first_name = 'Энн';
```

Имя таблицы

Поставьте символ «;» и нажмите RETURN, чтобы выполнить запрос: «Если столбец first_name содержит имя «Энн», покажите запись».

Ключевое слово WHERE сообщает РСУБД, что вы хотите отобразить из таблицы подмножество записей.

В условии WHERE это означает, что команда проверяет значения столбца с именем first_name.

= на языке SQL это означает «равно».

Значение столбца first_name. Не забудьте, что текстовые значения должны заключаться в апострофы.

В этом окне консоли показан результат запроса — подмножество записей, у которых столбец first_name содержит значение 'Энн'.

```
File Edit Window Help NoDate
> SELECT * FROM my_contacts WHERE first_name = 'Anne';
```

last_name	first_name	email	gender	birthday	location
Тот	Энн	Anne Toth@leapinlimos.com	Ж	NULL	Сан-Франциско, CA
Мэнсон	Энн	am86@objectville.net	Ж	NULL	Сиэтл, WA
Харди	Энн	anneh@b0tt0msup.com	Ж	NULL	Сан-Франциско, CA
Паркер	Энн	annep@starbuzzcoffee.com	Ж	NULL	Сан-Франциско, CA
Влант	Энн	anneblunt@breakneckpizza.com	Ж	NULL	Сан-Франциско, CA
Джейкобс	Энн	anne99@objectville.net	Ж	NULL	Сан-Хосе, CA

6 rows in set (3.67 sec)

Результат выполнения команды SELECT.



Минутку, вы же не думали, что я не замечу знак * ? Что он здесь делает?

Что это за * ?

Звездочка (*) приказывает РСУБД вернуть значения всех столбцов таблицы.

```
SELECT * FROM my_contacts  
WHERE first_name = 'Энн';
```

Когда вы видите `SELECT *`, считайте, что эта конструкция приказывает РСУБД вернуть **ВСЕ СТОЛБЦЫ**.



Я — звезда!

Звездочка используется для выборки **всех** столбцов таблицы.

Часто задаваемые вопросы

В: А если я не хочу включать в выборку все столбцы? Можно ли использовать что-то другое вместо звездочки?

О: Да, можно. Звездочка выбирает все столбцы, но через несколько страниц вы

узнаете, как ограничить выборку частью столбцов, чтобы с результатом было проще работать.

В: Есть ли другие символы, которые, как и звездочка, имеют специальное значение?

О: В SQL есть и другие специальные (зарезервированные) символы. Мы еще встретимся с ними позднее в этой книге. А пока вам достаточно знать только один специальный символ — звездочку. Тогда этот специальный символ используется в части `SELECT` команды SQL.



Бар Head First Lounge включает в свое меню фруктовые коктейли. Используя то, что вы узнали в главе 1, создайте таблицу и вставьте в нее приведенные ниже данные.

Таблица входит в базу данных с именем `drinks`. База данных содержит таблицы `easy_drinks` с рецептами напитков, состоящих всего из двух ингредиентов.

`easy_drinks`

<code>drink_name</code>	<code>main</code>	<code>amount1</code>	<code>second</code>	<code>amount2</code>	<code>directions</code>
Терновник	тоник	1.5	ананасовый сок	1	взболтать со льдом, разлить по бокалам, украсить лимонной цедрой
Голубая луна	содовая	1.5	черничный сок	0.75	взболтать со льдом, разлить по бокалам, украсить лимонной цедрой
Вот тебе на	персиковый нектар	1	ананасовый сок	1	взболтать со льдом, разлить по стаканам
Лаймовый физз	Спрайт	1.5	сок лайма	0.75	взболтать со льдом, разлить по бокалам
Поцелуй	вишневый сок	2	абрикосовый нектар	7	подавать со льдом и соломинкой
Горячее золото	персиковый нектар	3	апельсиновый сок	6	вливать в кружку горячий апельсиновый сок, добавить персиковый нектар
Одинокое дерево	содовая	1.5	вишневый сок	0.75	взболтать со льдом, разлить по бокалам
Борзая	содовая	1.5	грейпфрутовый сок	5	подавать со льдом, тщательно взболтать
Бабье лето	яблочный сок	2	горячий чай	6	налить сок в кружку, добавить горячий чай
Лягушка	холодный чай	1.5	лимонад	5	подавать на льду с ломтиком лайма
Сода плюс	содовая	2	виноградный сок	1	взболтать в бокале, подавать без льда

Значения `amount1` и `amount2` задаются в унциях.

→ Ответ на с. 151.



Не жалейте времени на планирование.

Тщательно выбирайте типы данных и не забывайте о `NULL`. Затем **проверьте свой код на с. 151.**



Если в запросах попадают незнакомые символы — не беспокойтесь. Просто введите их и посмотрите, как будет выполнен запрос.

Получите свой напиток



Воспользуйтесь только что созданной таблицей `easy_drinks` и проверьте эти запросы на своем компьютере. Запишите, какие напитки вернет каждый запрос.

```
SELECT * FROM easy_drinks WHERE main = 'Спрайт';
```

Какие напитки?

```
SELECT * FROM easy_drinks WHERE main = содовая;
```

Какие напитки?

```
SELECT * FROM easy_drinks WHERE amount2 = 6;
```

Какие напитки?

```
SELECT * FROM easy_drinks WHERE second = "апельсиновый сок";
```

Какие напитки?

```
SELECT * FROM easy_drinks WHERE amount1 < 1.5;
```

Какие напитки?

```
SELECT * FROM easy_drinks WHERE amount2 < '1';
```

Какие напитки?

```
SELECT * FROM easy_drinks WHERE main > 'содовая';
```

Какие напитки?

```
SELECT * FROM easy_drinks WHERE amount1 = '1.5';
```

Какие напитки?



Один момент... Вы говорите: «Проверьте эти запросы». Значит, предполагается, что все они работают. И я вам поверила! Но один запрос вообще не работает, а еще несколько выглядят сомнительно.

Да, вы абсолютно правы.

Один запрос вообще не работает.
А несколько других работают, хотя вроде бы и не должны.

Вопрос на повышенную оценку: напишите, какой запрос не работает...

...и какие запросы работают, хотя, казалось бы, работать не должны.

Получите свой напиток

Итак, вы воспользовались таблицей `easy_drinks`, проверили запросы на своем компьютере и записали, какие напитки вернет каждый запрос.



```
SELECT * FROM easy_drinks WHERE main = 'Спрайт';
```

Какие напитки? Лаймовый физз.....

Обратите внимание на апострофы.

```
SELECT * FROM easy_drinks WHERE main = содовая;
```

Хм... Похоже, этот запрос

Какие напитки? Ошибка.....

выполняться не будет.

```
SELECT * FROM easy_drinks WHERE amount2 = 6;
```

Какие напитки? Горячее золото, Бабье лето.....

А этот запрос работает. Переменная относится к типу DEC поэтому апострофы не нужны

```
SELECT * FROM easy_drinks WHERE second = "апельсиновый сок";
```

Какие напитки? Горячее золото.....

```
SELECT * FROM easy_drinks WHERE amount1 < 1.5;
```

Какие напитки? Вот тебе на.....

```
SELECT * FROM easy_drinks WHERE amount2 < '1';
```

Какие напитки? Голубая луна, Лаймовый физз, Одинокое дерево.....

```
SELECT * FROM easy_drinks WHERE main > 'содовая';
```

Какие напитки? Терновник, Лаймовый физз.....

Еще одно правильно сформулированное условие WHERE.

```
SELECT * FROM easy_drinks WHERE amount1 = '1.5';
```

Какие напитки? Терновник, Голубая луна, Лаймовый физз, Одинокое дерево, Борзая, Лягу.....

Вопрос на повышенную оценку: напишите, какой запрос не работает...

Некорректное условие WHERE.
Значение столбца VARCHAR
должно быть заключено
в апострофы.

WHERE main = содовая;

...и какие запросы работают, хотя, казалось бы, работать не должны.

WHERE second = "апельсиновый сок";

Запрос выполняется без ошибок, хотя
при вставке значения использовались
апострофы вместо кавычек..

WHERE amount2 < '1';

Запрос работает, хотя
вроде бы не должен, по-
тому что переменные
DEC не нужно заключать
в апострофы.

WHERE amount1 = '1.5';

То же самое!

Последние два запроса работают, потому что РСУБД обычно прощают незначительные ошибки пользователей. Они игнорируют апострофы и рассматривают значения DEC и INT как числа, хотя апострофы определяют их как текст. Эти запросы НЕПРАВИЛЬНЫ, но РСУБД исправляет вашу ошибку.

Как запрашивать разные типы данных

Чтобы написать правильное условие WHERE, необходимо правильно отформатировать каждый из входящих в него типов данных. Ниже представлены правила форматирования для всех основных типов.



Типы данных VARCHAR, CHAR, BLOB, DATE и TIME записываются в апострофах. Числовые типы DEC и INT записываются без апострофов.

Мы ♥ апострофы	А мы нет
CHAR	DEC
VARCHAR	INT
DATE	
DATE TIME, TIME, TIMESTAMP	
BLOB	

Проблемы со знаками препинания

Грег нашел еще нескольких знакомых. Он пытается включить одного из них в свою таблицу:

Стив Фанион
Дата рождения: 1/4/1970
Панк

Не женат

Гровер' Милл, Нью-Джерси
steve@onionflavoredrings.com

Увлечения: бунтарство

Ищет: единомышленники,
гитаристы

```
INSERT INTO my_contacts
```

```
VALUES
```

```
('Фанион', 'Стив', 'steve@onionflavoredrings.com', 'М',  
'1970-01-04', 'Панк', 'Гровер' Милл, NJ', 'Не женат',  
'бунтарство', 'единомышленники, гитаристы');
```

Но его PCУБД почему-то не отвечает. Грег вводит несколько символов «;», пытаясь завершить обработку запроса. Безуспешно.

```
File Edit Window Help Aliens!  
> INSERT INTO my_contacts VALUES ('Фанион', 'Стив', 'steve@  
onionflavoredrings.com', 'М', '1970-04-01', 'Панк', 'Гровер'  
Милл, NJ', 'Не женат', 'Бунтарство', 'Единомышленники,  
гитаристы');  
  
>  
> ;  
> ;  
>
```

При каждом нажатии
RETURN выдается
приглашение: '>'.
→



Как вы думаете, что происходит?



Хм, а почему перед приглашением постоянно выводится апостроф? Наверняка какие-то проблемы с апострофами в команде INSERT...

Непарный апостроф

Точно! Когда Грег попытался вставить запись, РСУБД ожидала увидеть четное число апострофов — по одному до и после каждого значения VARCHAR, CHAR и DATE. Название города Гровер' Милл породило проблему, потому что в нем содержится лишний апостроф. РСУБД хочет получить еще один закрывающий апостроф.



РАССЛАБЬТЕСЬ

Как вернуть консоль под свой контроль

Завершите команду — введите апостроф и точку с запятой. РСУБД получает дополнительный апостроф, который ожидает получить.

РСУБД выдаст сообщение об ошибке, но попробовать все равно стоит.

Правда, при этом будет выдано сообщение об ошибке, и вам придется вводить команду **INSERT** заново.

Апостроф с символом «;» завершает некорректную команду INSERT.

Сообщение довольно четко объясняет суть происходящего. В нем цитируется часть запроса, начинающаяся с лишнего апострофа.

```
File Edit Window Help TakeTwo
> INSERT INTO my_contacts VALUES ('фанион', 'Стив', 'steve@onionflavoredrings.com', 'М', '1970-01-04', 'Панк', 'Гровер' Милл, NJ', 'Не женат', 'Бунтарство', 'Единомышленники, гитаристы');
>
> ;
> ;
>
> ' ;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your SQL server version for the right syntax to use near ' Милл, Нью-Джерси', 'Не женат', 'Бунтарство', 'Единомышленники, гитаристы');
' at line 1
>
```

Запись не была вставлена, но РСУБД хотя бы снова реагирует на ваши действия.

Апострофы как специальные символы

Если вы вставляете в таблицу значение VARCHAR, CHAR или BLOB, содержащее внутренний апостроф, необходимо сообщить РСУБД, что этот апостроф не завершает текст, и является его частью и его необходимо включить в строку. Для этого можно поставить перед апострофом обратную косую черту.

```
INSERT INTO my_contacts  
(location)  
VALUES  
('Гровер\' Милл');
```

Апостроф входит в число «за-резервированных» символов SQL. Это означает, что в языке он выполняет особые функции.

Он сообщает РСУБД, где начинается и заканчивается фрагмент текста.



Часть задаваемые вопросы

В: Какие типы данных должны заключаться в апострофы?

О: Текстовые типы данных — а проще говоря, значения VARCHAR, CHAR, BLOB и TIME/DATE. Короче, все, что не является числом.

В: Нужны ли апострофы в столбцах DEC и INT?

О: Нет. Числовые столбцы не содержат пробелов; РСУБД понимает, где кончается число и начинается следующее слово в команде.

В: Значит, апострофы используются только в текстовых столбцах?

О: Да. Проблема в том, что текстовые столбцы могут содержать пробелы. При наличии апострофов в данных это создает проблемы. SQL не знает, как отличить внутренний апостроф в тексте от апострофа, начинающего или завершающего значение столбца.

В: А не проще различать их, заключая текстовые значения в кавычки вместо апострофов?

О: Нет, не проще. Дело в том, что команды SQL позднее могут использоваться в языках программирования (например, PHP). В языке программирования кавычки означают «здесь приводится команда SQL» — таким образом апострофы распознаются как часть команды SQL, а не как часть конструкций языка программирования.

Команда INSERT с Внутренним апострофом

Вы должны сообщить РСУБД, что апостроф не обозначает начало или конец строки, а является *частью* текста.

Экранирование обратной косой чертой

Чтобы решить эту проблему (и одновременно исправить команду INSERT), поставьте перед апострофом в тексте обратную косую черту:

```
INSERT INTO my_contacts
```

```
VALUES
```

```
('Фанион', 'Стив', 'steve@onionflavoredrings.com', 'М', '1970-01-04', 'Панк', 'Гровер\' Милл, NJ', 'Не женат', 'Бунтарство', 'Единомышленники, гитаристы');
```

Когда вы ставите перед апострофом префикс \, указывающий, что апостроф является частью текста, это называется «экранированием».

Экранирование удвоением апострофа

Апостроф также можно «экранировать» другим способом — поставив перед ним дополнительный апостроф.

```
INSERT INTO my_contacts
```

```
VALUES
```

```
('Фанион', 'Стив', 'steve@onionflavoredrings.com', 'М', '1970-01-04', 'Панк', 'Гровер' Милл, NJ', 'Не женат', 'Бунтарство', 'Единомышленники, гитаристы');
```

Апострофы также можно «экранировать» удвоением, то есть заменой одного апострофа двумя.



МОЗГОВОЙ ШТУРМ

С какими еще символами могут возникнуть аналогичные проблемы?



Упражнение

Если таблица содержит данные с апострофами, вероятно, в какой-то момент вам придется искать их с условием `WHERE`. Чтобы включить в выборку данные с апострофами, экранируйте их, как это делалось при вставке.

Перепишите следующую команду с использованием двух разных способов экранирования внутреннего апострофа.

```
SELECT * FROM my_contacts
WHERE
location = 'Гровер' Милл, NJ';
```

1

2

Какой способ вы предпочитаете?



Упражнение

Решение

Если таблица содержит данные с апострофами, вероятно, в какой-то момент вам придется искать их с условием `WHERE`. Чтобы включить в выборку данные с апострофами, экранируйте их, как это делалось при вставке.

Перепишите следующую команду с использованием двух разных способов экранирования внутреннего апострофа.

```
SELECT * FROM my_contacts
WHERE
location = 'Гровер' Милл';
```

1

```
SELECT * FROM my_contacts
```

```
WHERE
```

```
location = 'Гровер \ ' Милл, NJ';
```

*Первый способ: обратная
косая черта.*

2

```
SELECT * FROM my_contacts
```

```
WHERE
```

```
location = 'Гровер '' Милл, NJ';
```

*Второй способ: удвоение
апострофа*

Выборка ограниченного набора столбцов

Итак, вы знаете, как написать команду `SELECT` для выборки любых типов данных — в том числе и содержащих апострофы.

Выход `SELECT *` получается слишком длинным. А если меня интересует только адрес электронной почты? Нельзя ли скрыть лишние столбцы?



Команда `SELECT` может включить в выборку только те столбцы, которые вам нужны.

Чтобы с результатами было удобно работать, их нужно немного ограничить. Иначе говоря, выходные данные таблицы должны содержать меньшее количество столбцов — только те столбцы таблицы, которые нас интересуют.

Попробуйте сами



Упражнение

Прежде чем вводить следующий запрос `SELECT`, прикиньте, как будет выглядеть таблица результатов.

(Структура таблицы `easy_drinks` показана на с. 93)

Символ `*` заменяется именами столбцов.

```
SELECT drink_name, main, second
FROM easy_drinks
WHERE main = 'содовая';
```



Упражнение

Попробуйте сами

Прежде чем вводить следующий запрос SELECT, прикиньте, как будет выглядеть таблица результатов.

drink_name	main	second
Голубая луна	содовая	черничный сок
Одинокое дерево	содовая	вишневый сок
Борзая	содовая	грейпфрутовый сок
Сода плюс	содовая	виноградный сок

Старый способ

SELECT * FROM easy_drinks;

При выводе всех столбцов результаты не помещаются в окне терминала. Данные переносятся на следующую строку и разобраться в них довольно сложно.

```
File Edit Window Help MessyDisplay
> SELECT * FROM easy_drinks;
+-----+-----+-----+-----+-----+-----+
| drink_name | main | amount1 | second | amount2 | directions |
+-----+-----+-----+-----+-----+-----+
| Поцелуй | вишневый сок | 2.0 | абрикосовый нектар | 7.00 | подавать со льдом и соломинкой |
| Горячее золото | персиковый нектар | 3.0 | апельсиновый сок | 6.00 | влить в кружку горячий апельсиновый сок, добавить персиковый нектар |
| Одинокое дерево | содовая | 1.5 | вишневый сок | 0.75 | взболтать со льдом, разлить по бокалам |
| Борзая | содовая | 1.5 | грейпфрутовый сок | 5.00 | подавать со льдом, тщательно взболтать |
| Бабье лето | яблочный сок | 2.0 | горячий чай | 6.00 | налить сок в кружку, добавить горячий чай |
| Лягушка | холодный чай | 1.5 | лимонад | 5.00 | подавать на льду с ломтиком лайма |
| Сода плюс | содовая | 2.0 | виноградный сок | 1.00 | взболтать в бокале, подавать без льда |
| Терновник | тоник | 1.5 | ананасовый сок | 1.00 | взболтать со льдом, разлить по бокалам, украсить лимонной цедрой |
| Голубая луна | содовая | 1.5 | черничный сок | 0.75 | взболтать со льдом, разлить по бокалам, украсить лимонной цедрой |
| Вот тебе на | персиковый нектар | 1.0 | ананасовый сок | 1.00 | взболтать со льдом, разлить по стаканам |
| Лаймовый физз | Спрайт | 1.5 | лаймовый сок | 0.75 | взболтать со льдом, разлить по бокалам |
+-----+-----+-----+-----+-----+-----+
11 rows in set (0.00 sec)
```

Отбор конкретных столбцов

Указывая, какие столбцы должны возвращаться запросом, мы отбираем из полных результатов интересующую нас информацию. По аналогии с тем, как условие WHERE ограничивает количество возвращаемых записей, конструкция отбора столбцов ограничивает количество возвращаемых столбцов. По сути, вы поручаете работу по отбору информации SQL.

```
SELECT drink_name, main, second
FROM easy_drinks;
```

...Мы можем сузить результаты выборки, включая в них только интересующие нас столбцы.

```
File Edit Window Help JustEnough
> SELECT drink_name, main, second FROM easy_drinks;
+-----+-----+-----+
| drink_name | main | second |
+-----+-----+-----+
| Поцелуй | вишневый сок | абрикосовый нектар |
| Горячее золото | персиковый нектар | апельсиновый сок |
| Одинокое дерево | содовая | вишневый сок |
| Борзая | содовая | грейпфрутовый сок |
| Бабье лето | яблочный сок | горячий чай |
| Лягушка | холодный чай | лимонад |
| Сода плюс | содовая | виноградный сок |
| Терновник | тоник | ананасовый сок |
| Голубая луна | содовая | черничный сок |
| Вот тебе на | персиковый нектар | ананасовый сок |
| Лаймовый физз | Спрайт | сок лайма |
+-----+-----+-----+
11 rows in set (0.00 sec)
```

Отбор столбцов ускоряет получение результатов

Отбор столбцов полезен и удобен, но у него есть и другие преимущества. С увеличением объема данных в таблице отбор столбцов ускоряет получение результатов. Ускорение проявляется и при использовании кода SQL в других языках программирования, например PHP.

Возьми в руку карандаш



Несколько способов получить «Поцелуй»

Помните нашу таблицу `easy_drinks`? Следующая команда `SELECT` вернет коктейль «Поцелуй»:

```
SELECT drink_name FROM easy_drinks
WHERE
main = 'вишневый сок' ;
```

Допишите четыре команды `SELECT` на следующей странице, чтобы они тоже возвращали «Поцелуй».

easy_drinks

drink_name	main	amount1	second	amount2	directions
Терновник	тоник	1.5	ананасовый сок	1	взболтать со льдом, разлить по бокалам, украсить лимонной цедрой
Голубая луна	содовая	1.5	черничный сок	0.75	взболтать со льдом, разлить по бокалам, украсить лимонной цедрой
Вот тебе на	персиковый нектар	1	ананасовый сок	1	взболтать со льдом, разлить по стаканам
Лаймовый физз	Спрайт	1.5	сок лайма	0.75	взболтать со льдом, разлить по бокалам
Поцелуй	вишневый сок	2	абрикосовый нектар	7	подавать со льдом и соломинкой
Горячее золото	персиковый нектар	3	апельсиновый сок	6	вливать в кружку горячий апельсиновый сок, добавить персиковый нектар
Одинокое дерево	содовая	1.5	вишневый сок	0.75	взболтать со льдом, разлить по бокалам
Борзая	содовая	1.5	грейпфрутовый сок	5	подавать со льдом, тщательно взболтать
Бабье лето	яблочный сок	2	горячий чай	6	налить сок в кружку, добавить горячий чай
Лягушка	холодный чай	1.5	лимонад	5	подавать на льду с ломтиком лайма
Сода плюс	содовая	2	виноградный сок	1	взболтать в бокале, подавать без льда

SELECT

WHERE

SELECT

WHERE

SELECT

WHERE

SELECT

WHERE

Теперь запишите три команды SELECT, которые возвращают коктейль «Лягушка».

1

2

3

Возьми в руку карандаш



Решение

Допишите четыре команды SELECT на следующей странице, чтобы они тоже возвращали «Поцелуй».

SELECT *drink_name FROM easy_drinks*

WHERE *second = 'абрикосовый нектар';*

SELECT *drink_name FROM easy_drinks*

WHERE *amount2 = 7;*

SELECT *drink_name FROM easy_drinks*

WHERE *directions = 'подавать со льдом и соломинкой';*

SELECT *drink_name FROM easy_drinks*

WHERE *drink_name = 'Поцелуй';*

Эта форма встречается редко. Используйте ее, если вы полностью уверены в том, что столбец *drink_name* записан без ошибок.

Теперь запишите три команды SELECT, которые возвращают коктейль «Лягушка».

1 SELECT *drink_name FROM easy_drinks*

WHERE *main = 'холодный чай';*

2 SELECT *drink_name FROM easy_drinks*

WHERE *second = 'лимонад';*

3 SELECT *drink_name FROM easy_drinks*

WHERE *directions = 'подавать на льду с ломтиком лайма';*

КЛЮЧЕВЫЕ МОМЕНТЫ



- Используйте апострофы в условии WHERE при ссылке на значения текстовых столбцов.
- Не используйте апострофы при ссылке на значения числовых столбцов.
- Используйте * в команде SELECT для выборки всех столбцов таблицы.
- Если вы ввели запрос, а РСУБД не может завершить его обработку, проверьте, нет ли в нем непарных апострофов.
- По возможности используйте выборку конкретных столбцов таблицы (вместо конструкции SELECT *, включающей все столбцы).

Часть Задаваемые Вопросы

В: А если запрос должен вернуть все столбцы таблицы? Перечислять их в SELECT или использовать * ?

О: Если вам действительно нужны все столбцы — конечно, используйте *. Перечисление столбцов хорошо работает только тогда, когда вас интересует ограниченное подмножество столбцов.

В: Я скопировал запрос из Интернета, но когда пытаюсь выполнить его на своем компьютере — происходит ошибка. Я делаю что-то не так?

О: Запросы, вставленные из браузера, часто содержат невидимые символы, внешне неотличимые от пробелов, но имеющие

другой смысл для SQL. Вставка в текстовый редактор — один из способов выявления и удаления символов-«невидимок». Так что в подобных ситуациях лучше всего вставить запрос в текстовый редактор и повнимательнее присмотреться к нему.

В: Например, в Microsoft Word?

О: Нет, Word — не лучший вариант. Эта программа не показывает скрытое форматирование, которое может присутствовать в тексте. Попробуйте использовать Блокнот (PC) или TextEdit в режиме простого текста (Mac).

Пончики и таблицы...

Чтобы найти в таблице лучшие пончики с глазурью, нам понадобятся минимум две команды SELECT. Первая выбирает записи с пончиками нужного типа, а вторая – записи с оценкой 10.



doughnut_ratings

location	time	date	type	rating	comments
Starbuzz Coffee	7:43	23/4	с корицей	6	слишком много пряностей
Duncan's Donuts	8:56	25/8	с глазурью	5	жирноваты
Duncan's Donuts	19:58	26/4	с вареньем	6	вчерашние, но вкусные
Starbuzz Coffee	22:35	24/4	с глазурью	7	теплые, но не горячие
Krispy King	21:39	26/9	с вареньем	6	мало варенья
Starbuzz Coffee	7:48	23/4	шоколадный кекс	10	с зефиром!
Krispy King	20:56	25/11	с глазурью	8	кленовый сироп

Представьте, что таблица содержит 10 000 записей.

1 Можно провести поиск по типу пончиков:

В выборку включается столбец rating для поиска высшей оценки, а также столбец location с названием заведения.

```
SELECT location, rating FROM doughnut_ratings
WHERE
type = 'с глазурью';
```

Все выбранные записи будут иметь нужное значение type.

Результаты первого запроса... Представьте, что дальше идут еще несколько сотен записей.

location	rating
Duncan's Donuts	5
Starbuzz Coffee	7
Krispy King	8
Starbuzz Coffee	10
Duncan's Donuts	8

...Таблицы и пончики

2

А можно выполнить поиск по оценке:

```
SELECT location, type FROM doughnut_ratings
WHERE rating = 10;
```

Все выбранные записи имеют высшую оценку.

В выборку включается столбец type для проверки типа, а также столбец location с названием заведения.

location	type
Starbuzz Coffee	шоколадный кекс
Krispy King	с глазурью
Starbuzz Coffee	с глазурью
Duncan's Donuts	шоколадный кекс

Второй запрос снова возвращает сотни записей.

Маловато пользы от таких запросов. Я могу выбрать любой из двух вариантов и перерыть результаты, но в таблице многие тысячи записей... А я хочу пончик **прямо сейчас!**



МОЗГОВОЙ ШТУРМ

На какой вопрос мы хотим получить ответ в этих запросах?



Объединение условий

Два условия поиска — тип «с глазурью» и оценка 10 — можно объединить в один запрос при помощи ключевого слова AND. Результаты такого запроса будут удовлетворять обоим условиям.

```

SELECT location
FROM doughnut_ratings
WHERE type = 'с глазурью'
AND
rating = 10;
    
```

Теперь достаточно выбрать только столбец location.

Ключевое слово AND объединяет два условия WHERE.

Результат запроса AND. Даже если запрос вернет несколько записей, мы будем знать, что во всех этих заведениях есть глазированные пончики с оценкой 10, так что пойти можно в любое из них. Или во все поочередно.

location	rating
Duncan's Donuts	5
Starbuzz Coffee	7
Krispy King	8
Starbuzz Coffee	10
Duncan's Donuts	8

Запрос объединяет результаты выборки по условиям «с глазурью» и оценкой 10 и находит записи, удовлетворяющие обоим условиям.

AND

location	type
Starbuzz Coffee	шоколадный кекс
Krispy King	с глазурью
Starbuzz Coffee	с глазурью
Duncan's Donuts	шоколадный кекс

location
Starbuzz Coffee

Мама, пойдём в Starbuzz? Ну пожааалуйста!?





Упражнение

Так значит, я мог найти Энн при помощи AND?

Используя таблицу `my_contacts`, напишите несколько запросов для Грегга. Включите в выборку только те столбцы, которые необходимы для получения ответа. Обратите особое внимание на апострофы.

Напишите запрос для получения адресов электронной почты всех программистов.

.....

Напишите запрос для получения имени и места жительства всех людей, у которых дата рождения совпадает с вашей.

.....

Напишите запрос, при помощи которого Грег мог бы найти всех Энн из Сан-Франциско.





Упражнение Решение

Используя таблицу `my_contacts`, напишите несколько запросов для Грега. Включите в выборку только те столбцы, которые необходимы для получения ответа. Обратите особое внимание на апострофы.

Напишите запрос для получения адресов электронной почты всех программистов.

Нам нужен
столбец `email`.

```
.....  
SELECT email FROM my_contacts  
.....  
WHERE profession = 'computer programmer';  
.....
```

Условие выбора — значение столбца `profession` равно 'программист'.

Напишите запрос для получения имени и места жительства всех людей, у которых дата рождения совпадает с вашей.

```
.....  
SELECT last_name, first_name, location  
.....  
FROM my_contacts  
.....  
WHERE birthday = '1975-09-05';  
.....
```

Здесь должна быть
ваша дата рождения.

Напишите запрос, при помощи которого Грег мог бы найти всех Энн из Сан-Франциско.

```
.....  
SELECT last_name, first_name, email  
.....  
FROM my_contacts  
.....  
WHERE location = 'Сан-Франциско, CA'  
.....  
AND first_name = 'Энн';  
.....
```

Поиск числовых значений

Предположим, вы хотите найти в таблице `easy_drinks` все напитки, содержащие более одной унции содовой, и сделать это *в одном запросе*. Сложное решение с двумя запросами выглядит так:

Нам нужны названия напитков.

Напитки, содержащие 1,5 унции содовой.

```
SELECT drink_name FROM easy_drinks
WHERE
  main = 'содовая'
AND
  amount1 = 1.5;
```

```
File Edit Window Help MoreSoda
> SELECT drink_name FROM easy_drinks WHERE main = 'содовая' AND
amount1 = 1.5;
+-----+
| drink_name |
+-----+
| Голубая луна |
| Одинокое дерево |
| Борзая |
+-----+
3 rows in set (0.00 sec)
```

Напитки, содержащие 2 унции содовой.

```
SELECT drink_name FROM easy_drinks
WHERE
  main = 'содовая'
AND
  amount1 = 2;
```

```
File Edit Window Help EvenMoreSoda
> SELECT drink_name FROM easy_drinks WHERE main = 'содовая' AND
amount1 = 2;
+-----+
| drink_name |
+-----+
| Сода плюс |
+-----+
1 row in set (0.00 sec)
```



А как было бы здорово, если бы в одном запросе можно было найти все напитки из таблицы `easy_drinks`, содержащие **более 1 унции содовой**... Но я знаю, что это всего лишь мечты...

easy_drinks

drink_name	main	amount1	second	amount2	directions
Терновник	тоник	1.5	ананасовый сок	1	взболтать со льдом, разлить по бокалам, украсить лимонной цедрой
Голубая луна	содовая	1.5	черничный сок	0.75	взболтать со льдом, разлить по бокалам, украсить лимонной цедрой
Вот тебе на	персиковый нектар	1	ананасовый сок	1	взболтать со льдом, разлить по стаканам
Лаймовый физз	Спрайт	1.5	сок лайма	0.75	взболтать со льдом, разлить по бокалам
Поцелуй	вишневый сок	2	абрикосовый нектар	7	подавать со льдом и соломинкой
Горячее золото	персиковый нектар	3	апельсиновый сок	6	вливать в кружку горячий апельсиновый сок, добавить персиковый нектар
Одинокое дерево	содовая	1.5	вишневый сок	0.75	взболтать со льдом, разлить по бокалам
Борзая	содовая	1.5	грейпфрутовый сок	5	подавать со льдом, тщательно взболтать
Бабье лето	яблочный сок	2	горячий чай	6	налить сок в кружку, добавить горячий чай
Лягушка	холодный чай	1.5	лимонад	5	подавать на льду с ломтиком лайма
Сода плюс	содовая	2	виноградный сок	1	взболтать в бокале, подавать без льда

Одного достаточно

Однако использовать два запроса вместо одного неэффективно; к тому же вы рискуете упустить напитки, в которые входит 1.75 или 3 унции содовой. Лучше воспользоваться оператором сравнения «больше»:



```
SELECT drink_name FROM easy_drinks
```

```
WHERE
```

```
main = 'содовая'
```

```
AND
```

```
amount1 > 1;
```

С оператором > это условие вернет все напитки, содержащие более 1 унции содовой.

```
File Edit Window Help DoltOnce
> SELECT drink_name FROM easy_drinks WHERE main = 'содовая' AND
amount1 > 1;
+-----+
| drink_name |
+-----+
| Голубая луна |
| Одинокое дерево |
| Борзая |
| Сода плюс |
+-----+
4 rows in set (0.00 sec)
```



А почему первые два запроса нельзя объединить дополнительным оператором AND?

Операторы сравнения

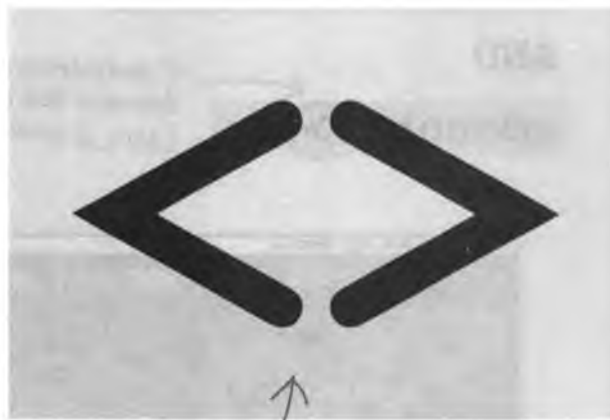
Ранее в наших условиях WHERE использовался только оператор =. Вы только что увидели пример использования оператора >, сравнивающего одно значение с другим. Ниже приведена полная сводка операторов сравнения.

Оператор = проверяет только точные совпадения. Он не поможет, если вы хотите проверить, что некоторое значение меньше или больше другого.

Этот странный знак означает «не равно». Его результат прямо противоположен результату знака =. Два значения либо равны, либо не равны — третьего не дано.



Всем известный знак равенства.



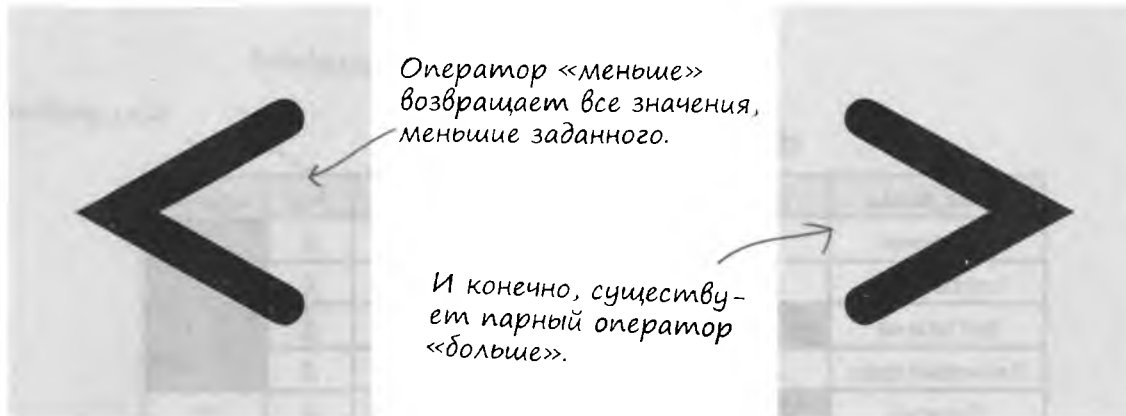
Означает «НЕ РАВНО». Условие возвращает все записи, у которых два значения не совпадают.



А вы заметили, что в каждом рассмотренном нами условии WHERE имя столбца располагалось слева? Будет ли условие работать, если имя столбца будет указано справа?

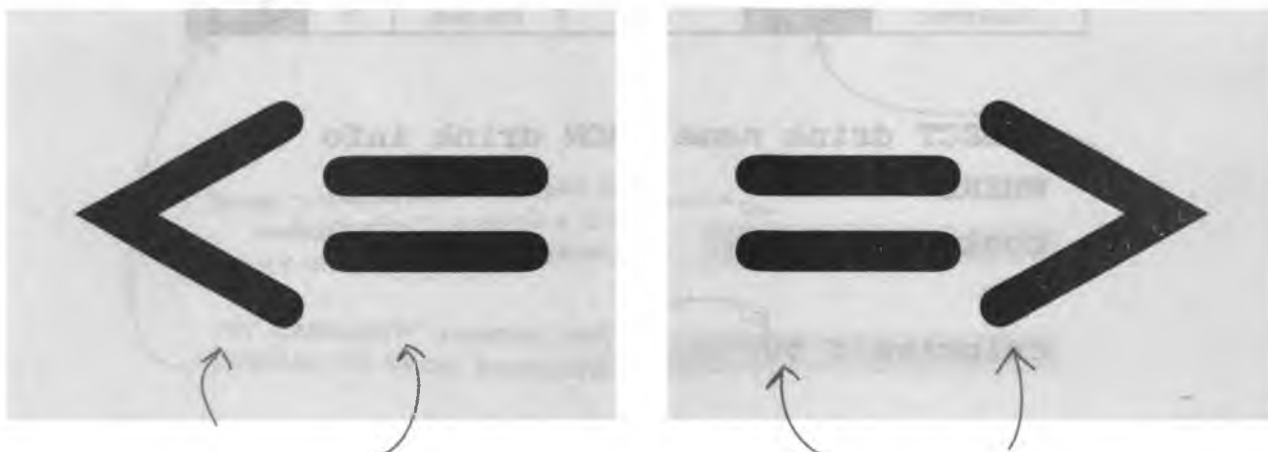
Оператор «**меньше**» сравнивает значение столбца, указанного слева, со значением в правой части. Если значение столбца меньше, то запись включается в возвращаемый набор.

Оператор «**больше**» по смыслу противоположен знаку «меньше». Он сравнивает значение столбца со значением в правой части. Если значение столбца больше, то запись включается в возвращаемый набор.



Оператор «**меньше или равно**» отличается от «меньше» только одним: столбцы, значение которых равно заданному, тоже включаются в результат.

То же и с оператором «**больше или равно**». Если значение столбца больше заданного значения или равно ему, то запись включается в возвращаемый набор.



Возвращаются все записи со значением столбца, МЕНЬШИМ ИЛИ РАВНЫМ заданному.

Оператор БОЛЬШЕ ИЛИ РАВНО.

Операторы сравнения при поиске числовых данных

В баре хранится таблица с ценами и данными о калорийности напитков. Владелец хочет отобразить напитки с высокой ценой и низкой калорийностью для проведения рекламной акции.

При помощи операторов сравнения он ищет в таблице `drink_info` напитки с ценой более \$3.50, содержащие не более 50 калорий.

drink_info

drink_name	cost	carbs	color	ice	calories
Терновник	3	8.4	желтый	Д	33
Голубая луна	2.5	3.2	синий	Д	12
Вот тебе на	3.5	8.6	оранжевый	Д	35
Лаймовый физз	2.5	5.4	зеленый	Д	24
Поцелуй	5.5	42.5	фиолетовый	Д	171
Горячее золото	3.2	32.1	оранжевый	Н	135
Одинокое дерево	3.6	4.2	красный	Д	17
Борзая	4	14	желтый	Д	50
Бабье лето	2.8	7.2	коричневый	Н	30
Лягушка	2.6	21.5	бронзовый	Д	80
Сода плюс	3.8	4.7	красный	Н	19

Количество углеводов (в граммах). Лед Калорийность

```
SELECT drink_name FROM drink_info
```

```
WHERE
```

```
cost >= 3.5
```

```
AND
```

```
calories < 50;
```

Это значит: «Напитки с ценой \$3.50 и более». Сюда входят напитки, стоящие ровно \$3.50.

Это значит: «Напитки, содержащие менее 50 калорий».

Запрос возвращает только напитки, удовлетворяющие **обоим** условиям — потому что два результата объединяются ключевым словом **AND**. Запрос возвращает напитки «Вот тебе на», «Одинокое дерево» и «Сода плюс».

Возьми в руку карандаш



А теперь ваша очередь. Напишите запросы, которые возвращают указанную информацию. Также запишите результат каждого запроса.

Цены желтых напитков со льдом, содержащих более 33 калорий.

.....

.....

Результат:

Названия и цвета напитков, содержащих не более 4 граммов углеводов, в которые кладется лед.

Результат:

Цены напитков, содежащих 80 и более калорий.

Результат:

Напитки «Борзая» и «Поцелуй», с цветом и информацией об использовании льда, но без указания названий напитков в запросе!

.....

Результат:



Возьми в руку карандаш

Решение

А теперь ваша очередь. Напишите запросы, которые возвращают указанную информацию. Также запишите результат каждого запроса.

Цены желтых напитков со льдом, содержащих более 33 калорий.

```
SELECT cost FROM drink_info
WHERE ice = 'Y'
AND
color = 'yellow'
AND
calories > 33;
```

Результат: \$4.00

Названия и цвета напитков, содержащих не более 4 граммов углеводов, в которые кладется лед.

```
SELECT drink_name, color FROM drink_info
WHERE
carbs <= 4
AND
ice = 'Y';
```

Результат: Голубая луна, синий

Цены напитков, содежащих 80 и более калорий.

```
SELECT cost FROM drink_info
WHERE
calories >= 80;
```

Результат: \$5.50, \$3.20, \$2.60

Напитки «Борзая» и «Поцелуй», с цветом и информацией об использовании льда, но без указания названий напитков в запросе!

```
SELECT drink_name, color, ice FROM drink_info
WHERE
cost >= 3.8;
```

Результат: Поцелуй, фиолетовый, Д Борзая, желтый, Д

Хитрый вопрос... Вы должны были просмотреть таблицу и найти столбец, по которому можно было бы отобразить эти — и только эти! — напитки.

Но ведь это работает только с числами, верно? А если мне понадобится найти все напитки, названия которых начинаются с конкретной буквы?



Операторы сравнения при поиске текстовых данных

Сравнение данных текстовых столбцов (CHAR и VARCHAR) происходит аналогичным образом. Операторы сравнивают значения **в алфавитном порядке**. Допустим, вас интересуют все напитки, названия которых начинаются с буквы «Г»; следующий запрос выбирает напитки, удовлетворяющие этому критерию.

drink_info

drink_name	cost	carbs	color	ice	calories
Терновник	3	8.4	желтый	Д	33
Голубая луна	2.5	3.2	синий	Д	12
Вот тебе на	3.5	8.6	оранжевый	Д	35
Лаймовый физз	2.5	5.4	зеленый	Д	24
Поцелуй	5.5	42.5	фиолетовый	Д	171
Горячее золото	3.2	32.1	оранжевый	Н	135
Одинокое дерево	3.6	4.2	красный	Д	17
Борзая	4	14	желтый	Д	50
Бабье лето	2.8	7.2	коричневый	Н	30
Лягушка	2.6	21.5	бронзовый	Д	80
Сода плюс	3.8	4.7	красный	Н	19

```
SELECT drink_name
FROM drink_info
WHERE
drink_name >= 'Г'
AND
drink_name < 'Д';
```

Запрос возвращает напитки, начинающиеся с буквы Г и следующих букв, но при этом первая буква предшествует Д.



РАССЛАБЬТЕСЬ

Пока не беспокойтесь о порядке следования записей в результатах.

В следующей главе вы узнаете, как отсортировать результаты по алфавиту.

Выбор ингредиентов

Бармена попросили сделать коктейль с вишневым соком. Для поиска рецептов можно воспользоваться двумя запросами.

Два запроса для проверки первого и второго ингредиента.

```
File Edit Window Help..
> SELECT drink_name FROM easy_drinks WHERE main = 'вишневый сок';
+-----+
| drink_name |
+-----+
| Поцелуй    |
+-----+
1 row in set (0.02 sec)

> SELECT drink_name FROM easy_drinks WHERE second = 'вишневый сок';
+-----+
| drink_name |
+-----+
| Одинокое дерево |
+-----+
1 row in set (0.01 sec)
```

Два запроса? Неэффективно. Наверняка их можно как-нибудь объединить.

drink_info

drink_name	cost	carbs	color	lco	calories
Терновник	3	8.4	желтый	Д	33
Голубая луна	2.5	3.2	синий	Д	12
Вот тебе на	3.5	8.6	оранжевый	Д	35
Лаймовый физз	2.5	5.4	зеленый	Д	24
Поцелуй	5.5	42.5	фиолетовый	Д	171
Горячее золото	3.2	32.1	оранжевый	Н	135
Одинокое дерево	3.6	4.2	красный	Д	17
Борзая	4	14	желтый	Д	50
Бабье лето	2.8	7.2	коричневый	Н	30
Лягушка	2.6	21.5	бронзовый	Д	80
Сода плюс	3.8	4.7	красный	Н	19



Быть ЦЛБ не быть

Для объединения двух запросов используется связка OR. С этим условием запрос возвращает записи, у которых выполняется *любое* из указанных условий. Таким образом, из двух отдельных запросов строится один комбинированный запрос.

```
File Edit Window Help SweetCherryPie
> SELECT drink_name from easy_drinks
WHERE main = 'вишневый сок'
OR
second = 'вишневый сок';
+-----+
| drink_name |
+-----+
| Поцелуй   |
| Одинокое дерево |
+-----+
2 rows in set (0.02 sec)
```

Возьми в руку карандаш



Вычеркните лишние части двух команд SELECT и добавьте связку OR, чтобы превратить их в одну команду SELECT:

```
SELECT drink_name FROM easy_drinks WHERE  
main = 'апельсиновый сок';
```

```
SELECT drink_name FROM easy_drinks WHERE  
main = 'яблочный сок';
```

Запишите здесь полученную команду SELECT.

Возьми в руку карандаш



Решение

Вычеркните лишние части двух команд SELECT и добавьте связку OR, чтобы превратить их в одну команду SELECT.

```
SELECT drink_name FROM easy_drinks WHERE
```

```
main = 'апельсиновый сок' <>
```

OR

```
SELECT drink_name FROM easy_drinks WHERE
```

```
main = 'яблочный сок';
```

Символ «>» нужно убрать, потому что команда еще не завершена.

Со связкой OR запрос вернет названия напитков, главным ингредиентом которых является апельсиновый ИЛИ яблочный сок.

Эту строку можно просто вычеркнуть, все необходимое есть в первой части запроса (присоединенной ключевым словом OR).

Запишите здесь полученную команду SELECT.

```
SELECT drink_name FROM easy_drinks
```

```
WHERE
```

```
main = 'апельсиновый сок'
```

```
OR
```

```
main = 'яблочный сок';
```

Запрос, который у нас получился.

Оператор OR действительно полезен, но я не понимаю, почему мы не воспользовались AND?



Не путайте AND с OR!

Если истинными должны быть **ВСЕ** условия, используйте **AND**.

Если истинным должно быть **ХОТЯ БЫ ОДНО** из условий, используйте **OR**.

Так и не разобрались? Переверните страницу.

AND

OR

Часто задаваемые вопросы

В: Можно ли использовать более одной связки AND или OR в одном условии WHERE?

О: Конечно, связок может быть сколько угодно. Также в одном условии AND может использоваться вместе с OR.

Чем AND отличается от OR

Следующие примеры демонстрируют возможные комбинации двух условий, объединенных связками AND и OR.

doughnut_ratings

location	time	date	type	rating	comments
Krispy King	8:50	27/9	с глазурью	10	почти идеально
Duncan's Donuts	8:59	25/8	NULL	6	жирноваты
Starbuzz Coffee	19:35	24/5	с корицей	5	вчерашние, но вкусные
Duncan's Donuts	19:03	26/4	с вареньем	7	мало варенья

```
SELECT type FROM doughnut_ratings
```

РЕЗУЛЬТАТЫ

Да, есть совпадение.

```
WHERE location = 'Krispy King' AND rating = 10;
```

Да

с глазурью

```
WHERE location = 'Krispy King' OR rating = 10;
```

с глазурью

Нет совпадений.

```
WHERE location = 'Krispy King' AND rating = 3;
```

нет

```
WHERE location = 'Krispy King' OR rating = 3;
```

с глазурью

Нет совпадений.

```
WHERE location = 'Snappy Bagel' AND rating = 10;
```

нет

```
WHERE location = 'Snappy Bagel' OR rating = 10;
```

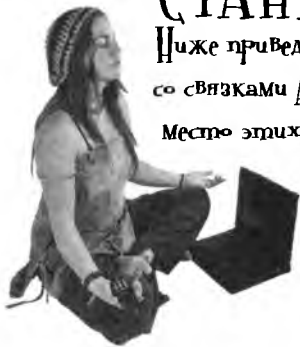
с глазурью

```
WHERE location = 'Snappy Bagel' AND rating = 3;
```

нет

```
WHERE location = 'Snappy Bagel' OR rating = 3;
```

нет



СТАТЬ УСЛОВИЕМ

Ниже приведена серия условий **WHERE** со связками **AND** и **OR**. Поставьте себя на место этих условий и определите, какой результат вернут такие запросы.

```
SELECT type FROM doughnut_ratings
```

```
WHERE location = 'Krispy King' AND rating <> 6;
```

```
WHERE location = 'Krispy King' AND rating = 3;
```

```
WHERE location = 'Snappy Bagel' AND rating >= 6;
```

```
WHERE location = 'Krispy King' OR rating > 5;
```

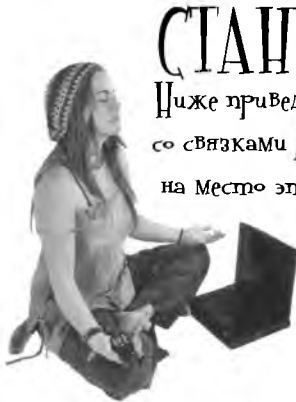
```
WHERE location = 'Krispy King' OR rating = 3;
```

```
WHERE location = 'Snappy Bagel' OR rating = 6;
```

Результат:

.....

Для улучшения своей кармы напишите, чем два результата отличаются от остальных.



СТАТЬ условием. Ответ

Ниже приведена серия условий **WHERE** со связками **AND** и **OR**. Поставьте себя на место этих условий и определите, какой результат вернут такие запросы.

SELECT type FROM doughnut_ratings

Результат:

WHERE location = 'Krispy King' AND rating <> 6;

с глазурью

WHERE location = 'Krispy King' AND rating = 3;

нет

WHERE location = 'Snappy Bagel' AND rating >= 6;

нет

WHERE location = 'Krispy King' OR rating > 5;

с глазурью, NULL,
с вареньем

WHERE location = 'Krispy King' OR rating = 3;

с глазурью

WHERE location = 'Snappy Bagel' OR rating = 6;

NULL

Для улучшения своей кармы напишите, чем два результата отличаются от остальных.

Два запроса возвращают NULL.

Эти значения NULL могут создать проблемы в будущих запросах. В столбце таблицы лучше ввести какое-либо определенное значение, чем оставлять в нем NULL, потому что прямая выборка NULL из таблицы невозможна.

Использование IS NULL для поиска NULL



Я пытался выполнить поиск по столбцам, содержащим NULL, но у меня ничего не получилось. Как найти NULL в таблице?

drink_info

drink_name	cost	carbs	color	ice	calories
Выходной	NULL	14	NULL	Д	50
Дыхание дракона	2.9	7.2	коричневый	Н	NULL

Прямая выборка NULL невозможна.

```
SELECT drink_name FROM drink_info
```

```
WHERE
```

```
calories = NULL;
```

Не работает, потому что никакое значение не равно NULL — неопределенному значению.

```
SELECT drink_name FROM drink_info
```

```
WHERE
```

```
calories = 0;
```

Не работает; NULL и ноль — это разные значения.

```
SELECT drink_name FROM drink_info
```

```
WHERE
```

```
calories = 'NULL';
```

Тоже не работает, NULL не является текстовой строкой.

Однако NULL можно найти при помощи ключевых слов.

```
SELECT drink_name
```

```
FROM drink_info
```

```
WHERE
```

```
calories IS NULL;
```

Ключевые слова — не текстовые данные, они не заключаются в кавычки.

Прямая выборка значений NULL возможна только с использованием ключевых слов IS NULL.

Часть задаваемые вопросы

В: Вы говорите, что «прямая выборка» NULL невозможна без использования IS NULL. Значит, возможна, непрямая?

О: Верно. Если вы хотите получить значение из этого столбца, можно воспользоваться условием WHERE для других столбцов. Например, следующий запрос вернет NULL:

```
SELECT calories FROM drink_info
WHERE drink_name = 'Дыхание дракона';
```

В: И как будет выглядеть результат этого запроса?

О: Он будет выглядеть так:

```
+-----+
| calories |
+-----+
| NULL     |
+-----+
```

Тем временем в доме Грега...

Грег пытается найти в таблице `my_contacts` всех людей, живущих в Калифорнии. Часть запроса, над которым он работает, выглядит так:

Набирать все эти
OR так утомительно!



```
SELECT * FROM my_contacts
WHERE
location = 'Сан-Франциско, CA'
OR
location = 'Сан-Хосе, CA'
OR
location = 'Сан-Матео, CA'
OR
location = 'Саннивейл, CA'
OR
location = 'Марин, CA'
OR
location = 'Окленд, CA'
OR
location = 'Пало-Альто, CA'
OR
location = 'Сакраменто, CA'
OR
location = 'Лос-Анджелес, CA'
OR
И так далее...
```

LIKE: слово для экономии времени

В Калифорнии слишком много городов. Если Грег попытается перечислить их все в запросе, объединяя связкой OR, это займет у него слишком много времени. К счастью, существует полезное ключевое слово LIKE, которое в сочетании со специальными символами ищет часть текстовой строки и возвращает совпадения.

Грег может использовать LIKE следующим образом:

```
SELECT * FROM my_contacts
WHERE location LIKE '%CA';
```

Знак % в апострофах указывает вашей программе, что вас интересуют все значения столбца location, которые заканчиваются сокращением «CA».

Специальные символы

LIKE обычно используется в сочетании с двумя специальными символами — «заместителями», которые представляют фактическое содержимое строки. Специальные символы, словно джокер в карточных играх, равны любому символу (или последовательности символов) строки.



Какие еще специальные символы встречались вам в этой главе?

Я LIKE это

LIKE используется со специальными символами. Первый — знак % — обозначает любое количество произвольных символов.



```
SELECT first_name FROM my_contacts  
WHERE first_name LIKE '%им';
```

Представляет любое количество неизвестных символов.

Запрос возвращает имена, которые состоят из любого количества символов и заканчиваются на «им» — Эфраим, Джим, Тим и т. д.

Второй специальный символ, так часто встречающийся в компании LIKE, — знак подчеркивания (_) — представляет ровно один произвольный символ.



```
SELECT first_name FROM my_contacts  
WHERE first_name LIKE '_им';
```

Заменяет ровно один произвольный символ.

Запрос возвращает имена, которые состоят из одной буквы и «им» — Ким, Тим и т. д.



Магниты с кодами

Условия WHERE с ключевым словом LIKE и их результаты полностью перепутались. Сможете ли вы восстановить соответствие? Некоторые условия могут возвращать несколько результатов. Напишите собственные условия LIKE со специальными символами для лишних результатов.

Ананас

Джон

Мичиган

Стремительный

Алабама

Блендер

Элси

Нью-Джерси

Монтана

Джошуа

Индия

Джошуа

SQL для начинающих

Мэйн

Джошуа

Нью-Йорк

Стрелка

WHERE state LIKE 'Нью-?';

WHERE cow_name LIKE '__си';

WHERE title LIKE 'SQL%';

WHERE rhyme_word LIKE '%ендер';

WHERE first_name LIKE 'Джо?';



Магниты с кодами

Условия WHERE с ключевым словом LIKE и их результаты полностью перепутались. Можете ли вы восстановить соответствие? Некоторые условия могут возвращать несколько результатов. Напишите собственные условия LIKE со специальными символами для лишних результатов.

WHERE state LIKE 'Нью-?';
Нью-Йорк Нью-Джерси

WHERE cow_name LIKE '_sie';
Элси

WHERE title LIKE 'SQL?';
SQL для начинающих

WHERE word LIKE 'Стр?';
Стремительный Стрелка

WHERE rhyme_word LIKE '?ендер';
Блендер

WHERE state LIKE 'M%' OR state LIKE 'A%';
Мичиган Монтана Алабама
Мэйн

WHERE first_name LIKE 'Дж?';
Джон Джошуа

WHERE word LIKE '_н?';
Ананас Индия

Проверка диапазонов с использованием AND и операторов сравнения

Владелец бара хочет отобразить напитки, калорийность которых находится в заданном диапазоне. Как составить запрос для получения названий напитков, у которых калорийность находится в диапазоне от 30 до 60 включительно?

drink_info

drink_name	cost	carbs	color	ice	calories
Терновник	3	8.4	желтый	Д	33
Голубая луна	2.5	3.2	синий	Д	12
Вот тебе на	3.5	8.6	оранжевый	Д	35
Лаймовый физз	2.5	5.4	зеленый	Д	24
Поцелуй	5.5	42.5	фиолетовый	Д	171
Горячее золото	3.2	32.1	оранжевый	Н	135
Одинокое дерево	3.6	4.2	красный	Д	17
Борзая	4	14	желтый	Д	50
Бабье лето	2.8	7.2	коричневый	Н	30
Лягушка	2.6	21.5	бронзовый	Д	80
Сода плюс	3.8	4.7	красный	Н	19

```
SELECT drink_name FROM drink_info
```

```
WHERE
```

```
calories >= 30
```

```
AND
```

```
calories <= 60;
```

← Результат запроса включает напитки, у которых калорийность больше или равна 30, но меньше или равна 60

Только МЕЖДУ нами... Есть и другой способ

Также для проверки вхождения значений в диапазон можно воспользоваться ключевым словом BETWEEN. Такая форма записи короче предыдущего запроса, но возвращает те же результаты. Обратите внимание: BETWEEN включает границы диапазона (30 и 60). Конструкция BETWEEN эквивалентна использованию операторов \leq и \geq , но не $<$ и $>$.

```
SELECT drink_name FROM drink_info  
WHERE
```

```
calories BETWEEN 30 AND 60;
```

Включает напитки
с 30 и 60 калориями.

Дает точно такой же резуль-
тат, как и запрос на предыду-
щей странице, но вводится
намного быстрее!

```
File Edit Window Help MediumCalories  
> SELECT drink_name FROM drink_info  
WHERE  
calories BETWEEN 30 AND 60;  
+-----+  
| drink_name |  
+-----+  
| Терновник |  
| Вот тебе на |  
| Борзая |  
| Бабье лето |  
| Сода плюс |  
+-----+
```

Возьми в руку карандаш



Измените запрос на предыдущей странице так, чтобы он возвращал названия всех напитков, содержащих более 60 или менее 30 калорий.

Попробуйте использовать `BETWEEN` с текстовыми столбцами. Напишите запрос, который возвращает названия всех напитков, начинающиеся с букв от «Д» до «О».

Как вы думаете, какой результат вернет следующий запрос?

```
SELECT drink_name FROM drink_info WHERE  
calories BETWEEN 60 AND 30;
```

Возьми в руку карандаш



Решение

Измените запрос на предыдущей странице так, чтобы он возвращал названия всех напитков, содержащих более 60 или менее 30 калорий.

```
SELECT drink_name FROM drink_info
```

```
WHERE
```

```
calories < 30 OR calories > 60;
```

Названия напитков, содержащих более 60 калорий.

Напитки, содержащие менее 30 калорий.

Попробуйте использовать BETWEEN с текстовыми столбцами. Напишите запрос, который возвращает названия всех напитков, начинающиеся с букв от «Д» до «О».

```
SELECT drink_name FROM drink_info
```

```
WHERE
```

```
drink_name BETWEEN 'D' AND 'O';
```

Запрос возвращает названия всех напитков, начинающихся с D, O и всех букв между ними.

Как вы думаете, какой результат вернет следующий запрос?

```
SELECT drink_name FROM drink_info WHERE  
calories BETWEEN 60 AND 30;
```

Порядок указания границ важен, поэтому этот запрос не вернет ни одной записи.

Запрос ищет значения, находящиеся в диапазоне от 60 до 30. Однако в этом диапазоне значений нет, потому что 60 больше 30. Чтобы ключевое слово BETWEEN работало так, как положено, меньшее число всегда должно указываться первым.

Условие IN

Аманда, подруга Грега, использует список контактов Грега для поиска парней. Она уже побывала на нескольких свиданиях, завела собственную таблицу со своими впечатлениями.

Аманда назвала свою таблицу `black_book`. Она хочет получить список удачных свиданий, поэтому отбирает значения к положительным оценкам.

```
SELECT date_name
FROM black_book
```

WHERE

```
rating = 'оригинально'
```

OR

```
rating = 'потрясающе'
```

OR

```
... ;
```

Положительные
оценки

Отдельное условие
для каждой положи-
тельной оценки.

black_book

date_name	rating
Алекс	оригинально
Джеймс	скучно
Иэн	потрясающе
Борис	так себе
Мелвин	пресно
Эрик	убого
Энтони	восхитительно
Сэмми	неплохо
Айвен	ужасно
Вик	смехотворно

Вместо того чтобы строить длинные цепочки OR, мы можем упростить запрос при помощи ключевого слова IN. После IN следует набор значений в круглых скобках. Если значение столбца совпадает с одним из значений набора, то запись или заданное подмножество столбцов включаются в результат запроса.

```
SELECT date_name
```

```
FROM black_book
```

WHERE

```
rating IN ('оригинально',
'потрясающе',
'восхитительно',
'неплохо');
```

Набор положительных
оценок.

За ключевым словом IN
следует набор допусти-
мых значений.

```
File Edit Window Help GoodDates
> SELECT date_name FROM black_book
WHERE
rating IN ('оригинально', 'потрясающе',
'восхитительно', 'неплохо');

+-----+
| date_name |
+-----+
| Алекс    |
| Иэн      |
| Энтони   |
| Сэмми    |
+-----+
```

Ключевые слова NOT IN

И конечно, Аманда хочет знать, кто из ее знакомых получил плохие оценки. Если они позвонят, у нее обнаружатся какие-нибудь неотложные дела.

Чтобы получить имена знакомых, получивших низкие оценки, поставьте перед IN ключевое слово NOT. С конструкцией NOT IN в выборку включаются записи, у которых значение столбца **не входит** в заданный набор.

```
SELECT date_name  
FROM black_book  
WHERE
```

```
rating NOT IN  
( 'оригинально',  
  'потрясающе',  
  'восхитительно',  
  'неплохо' );
```

Ключевые слова NOT IN означают, что результат не входит в заданный набор.

Запрос NOT IN возвращает список знакомых, не получивших положительных оценок, а следовательно, не имеющих шанса на второе свидание.

Неходишь
в набор?
Свободен!



```
File Edit Window Help BadDates  
> SELECT date_name FROM black_book  
WHERE  
rating NOT IN ('оригинально',  
'потрясающе', 'восхитительно', 'неплохо');  
+-----+  
| date_name |  
+-----+  
| Джеймс   |  
| Борис    |  
| Мелвин   |  
| Эрик     |  
| Айвен    |  
| Вик      |  
+-----+  
6 rows in set (2.43 sec)
```

 **МОЗГОВОЙ ШТУРМ**

Когда NOT IN удобнее IN?

Другие применения NOT

Ключевое слово NOT может использоваться не только с IN, но и с BETWEEN и LIKE. Однако необходимо помнить, что **NOT** следует сразу же после **WHERE**. Рассмотрим несколько примеров.

```
SELECT drink_name FROM drink_info
WHERE NOT carbs BETWEEN 3 AND 5;
```

Если ключевое слово NOT используется с AND или OR, то оно записывается после AND или OR.

```
SELECT date_name from black_book
WHERE NOT date_name LIKE 'A%'
AND NOT date_name LIKE 'B%';
```

Часто задаваемые вопросы

В: Вы же только что сказали, что NOT записывается после WHERE. А как насчет NOT IN?

О: Это исключение. И даже если поставить NOT после WHERE, команда все равно будет работать. Следующие две команды возвращают одинаковые результаты:

```
SELECT * FROM easy_drinks
WHERE NOT main IN ('содовая', 'холодный чай');
```

```
SELECT * FROM easy_drinks
WHERE main NOT IN ('содовая', 'холодный чай');
```

В: Будет ли NOT работать с <> (оператор «не равно»)?

О: Будет, но это будет двойное отрицание. Намного логичнее заменить эту конструкцию знаком =. Следующие два запроса возвращают одинаковые результаты:

```
SELECT * FROM easy_drinks
WHERE NOT drink_name <> 'Терновник';
```

```
SELECT * FROM easy_drinks
WHERE drink_name = 'Терновник';
```

В: Как NOT работает с NULL?

О: Так, как и следовало ожидать. Например, чтобы выбрать все записи, у которых столбец не содержит NULL, можно воспользоваться следующим запросом:

```
SELECT * FROM easy_drinks
WHERE NOT main IS NULL;
```

Однако следующий запрос тоже подойдет:

```
SELECT * FROM easy_drinks
WHERE main IS NOT NULL;
```

В: А как насчет AND и OR?

О: При использовании с AND и OR ключевое слово NOT ставится после них:

```
SELECT * FROM easy_drinks
WHERE NOT main = 'содовая'
AND NOT main = 'холодный чай';
```



Упражнение

Перепишите каждое из условий WHERE так, чтобы они были как можно проще. Используйте AND, OR, NOT, BETWEEN, LIKE, IN, IS NULL и операторы сравнения. Структура и содержимое таблиц приводились в этой главе.

```
SELECT drink_name from easy_drinks
WHERE NOT amount1 < 1.50;
```

```
SELECT drink_name FROM drink_info
WHERE NOT ice = 'Д';
```

```
SELECT drink_name FROM drink_info
WHERE NOT calories < 20;
```



Упражнение
Решение

Перепишите каждое из условий WHERE так, чтобы они были как можно проще. Используйте AND, OR, NOT, BETWEEN, LIKE, IN, IS NULL и операторы сравнения. Структура и содержимое таблиц приводились в этой главе.

```
SELECT drink_name from easy_drinks  
WHERE NOT amount1 < 1.50;
```

```
SELECT drink_name FROM easy_drinks
```

```
WHERE amount1 >= 1.50;
```

```
SELECT drink_name FROM drink_info  
WHERE NOT ice = 'Д';
```

```
SELECT drink_name FROM drink_info
```

```
WHERE ice = 'H';
```

```
SELECT drink_name FROM drink_info  
WHERE NOT calories < 20;
```

```
SELECT drink_name FROM drink_info
```

```
WHERE calories >= 20;
```

```

SELECT drink_name FROM easy_drinks
WHERE main = 'персиковый нектар'
OR main = 'содовая';

```

```

.....
SELECT drink_name FROM easy_drinks
WHERE main BETWEEN 'П' AND 'Т';

```

Работает, потому что в столбце main нет других ингредиентов, удовлетворяющих условию. Для реальной таблицы с множеством записей такое решение не подойдет.

```

SELECT drink_name FROM drink_info
WHERE NOT calories = 0;

```

```

.....
SELECT drink_name FROM drink_info
WHERE calories > 0;

```

Калорийность не бывает отрицательной, поэтому использование оператора > безопасно.

```

SELECT drink_name FROM drink_info
WHERE NOT carbs BETWEEN 3 AND 5;

```

```

.....
SELECT drink_name FROM drink_info
WHERE carbs < 3
OR
carbs > 5;

```

```

SELECT date_name from black_book
WHERE NOT date_name LIKE 'A%'
AND NOT date_name LIKE 'B%';

```

```

.....
SELECT date_name FROM black_book
WHERE date_name NOT BETWEEN 'A' AND 'B';

```



Новые инструменты

Мы подошли к концу главы 2, а ваш инструментарий пополнился несколькими новыми операторами и ключевыми словами. Полный список инструментов приведен в приложении III.

SELECT *
Используется для выборки всех столбцов таблицы.

Экранирование
Апострофы в текстовых данных экранируются символом \ или удвоением апострофа.

AND и OR
Связки AND и OR объединяют критерии в условии WHERE для повышения точности отбора.

NOT
NOT вычисляет условие, логически противоположное заданному.

= <> < > <= >=
В вашем распоряжении полный набор операторов сравнения.

IS NULL
Условие для проверки значения NULL.

BETWEEN
Используется для проверки вхождения значений в диапазон.

LIKE с % и _
LIKE со специальными символами используется для поиска по частям строк.

↑ ↗
Новые инструменты операторы!



Упражнение

Со с. 93

Решение

Бар Head First Lounge включает в свое меню фруктовые коктейли. Используя то, что вы узнали в главе 1, создайте таблицу и вставьте в нее приведенные ниже данные.

Таблица входит в базу данных с именем `drinks`. База данных содержит таблицу `easy_drinks` с рецептами напитков, состоящих всего из двух ингредиентов.

```
CREATE DATABASE drinks;
```

```
USE drinks;
```

```
CREATE TABLE easy_drinks
```

```
(drink_name VARCHAR(16), main VARCHAR(20), amount1 DEC(3,1),
second VARCHAR(20), amount2 DEC(4,2), directions VARCHAR(250));
```

Желательно выделить несколько лишних символов на случай, если там когда-нибудь появится более длинное название.

```
INSERT INTO easy_drinks
```

```
VALUES
```

```
('Терновник', 'тоник', 1.5, 'ананасовый сок', 1, 'взболтать со льдом, разлить по бокалам, украсить лимонной цедрой'), ('Голубая луна', 'содовая', 1.5, 'черничный сок', .75, 'взболтать со льдом, разлить по бокалам, украсить лимонной цедрой'), ('Вот тебе на', 'персиковый нектар', 1, 'ананасовый сок', 1, 'взболтать со льдом, разлить по стаканам'), ('Лаймовый физз', 'Спрайт', 1.5, 'сок лайма', .75, 'взболтать со льдом, разлить по бокалам'), ('Поцелуй', 'вишневый сок', 2, 'абрикосовый нектар', 7, 'подавать со льдом и соломинкой'), ('Горячее золото', 'персиковый нектар', 3, 'апельсиновый сок', 6, 'вливать в кружку горячий апельсиновый сок, добавить персиковый нектар'), ('Одинокое дерево', 'содовая', 1.5, 'вишневый сок', .75, 'взболтать со льдом, разлить по бокалам'), ('Борзая', 'содовая', 1.5, 'грейпфрутовый сок', 5, 'подавать со льдом, тщательно взболтать'), ('Бабье лето', 'яблочный сок', 2, 'горячий чай', 6, 'налить сок в кружку, добавить горячий чай'), ('Лягушка', 'холодный чай', 1.5, 'лимоннад', 5, 'подавать на льду с ломтиком лайма'), ('Сода плюс', 'содовая', 2, 'виноградный сок', 1, 'взболтать в бокале, подавать без льда');
```

Не забудьте: числовые данные в кавычки не заключаются!

Набор данных каждого напитка заключен в круглые скобки.

Напитки разделяются запятыми.

3 DELETE и UPDATE

О пользе изменений

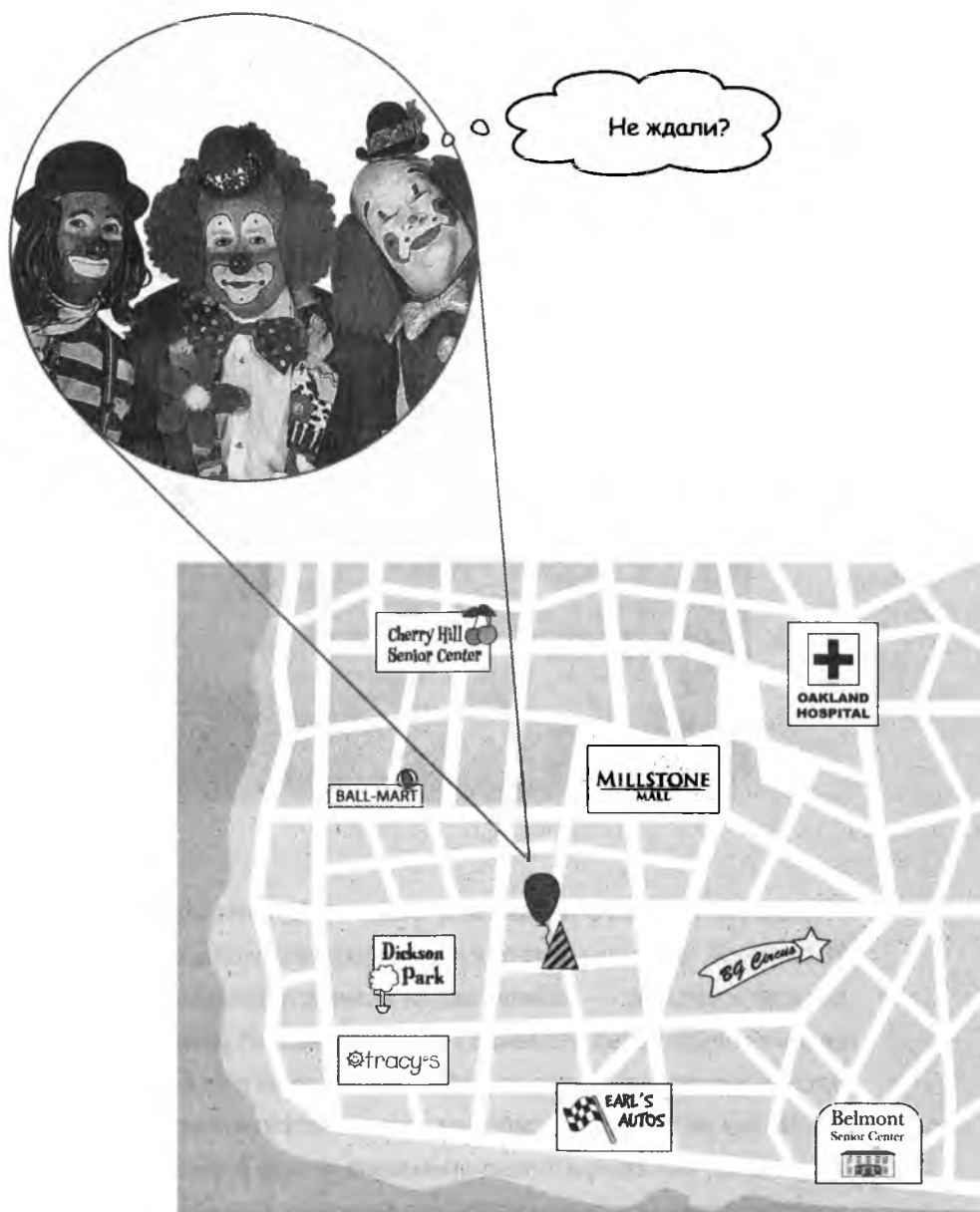
И в следующий раз не увлекайся с этой командой DELETE, ладно? У меня на гостинцы никаких денег не хватит.



Никак не можете прийти к окончательному решению? И ладно! Команды, с которыми вы познакомитесь в этой главе — **DELETE** и **UPDATE**, — избавят вас от возни с данными, которые вы ввели полгода назад. Команда **UPDATE** **изменяет данные**, а команда **DELETE** **удаляет данные из таблицы**, которые вам больше не нужны. Но мы не только рассмотрим новые инструменты; в этой главе вы узнаете, как избирательно применять новые возможности и как предотвратить случайное удаление полезных данных.

Клоуны вокруг нас

Предположим, мы хотим хранить информацию о клоунах, работающих в городке Дейтавилль. Данные хранятся в таблице `clown_info`, а текущее местонахождение клоунов содержится в столбце `last_seen`.



Информация о клоунах

Вот как выглядит наша таблица. Информацию, которой мы пока не располагаем, можно пропустить — она будет введена позднее. Каждый раз, когда в городе появляется новый клоун, в таблицу добавляется новая запись. Чтобы содержимое таблицы оставалось актуальным, его придется часто изменять.

Место, где каждого клоуна видели в последний раз

clown_info

name	last_seen	appearance	activities
Элси	Дом престарелых Черри Хилл	Ж, рыжие волосы, зеленый костюм, огромные ботинки	шарики, машинки
Пиклз	Вечеринка Джека Грина	М, оранжевые волосы, синий костюм, огромные ботинки	мим
Снаглз	Болмарт	Ж, желтая рубашка, красные штаны	рожок, зонтик
Мистер Хобо	Цирк BG	М, сигара, черные волосы, маленькая шляпа	скрипка
Кларабелл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цветок, синее платье	кричалки, танцы
Скутер	Больница Окленд	М, синие волосы, красный костюм, большой нос	шарики
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы
Бэйб	Автошкола Эрла	Ж, розовый костюм с блестками	эквилибристика, машинки
Бонзо		М, женское платье в горошек	ление, танцы
Снифлз	Заведение Трейси	М, зелено-фиолетовый костюм, длинный нос	



Пустые ячейки будут заполнены позднее.

Возьми в руку карандаш



Перемещения клоунов

Напишите команды SQL для занесения текущей информации о клоунах в таблицу `clown_info`. Учтите, что часть информации остается неизменной; за полными сведениями обращайтесь к таблице на с. 155.

Зиппо теперь поет.

```
INSERT INTO clown_info
```

```
VALUES
```

```
('Зиппо', 'Торговый центр Милстоун', 'Ж,  
оранжевый костюм, штаны', 'танцы, пение');
```

Снаглз носит синие штаны.

```
INSERT INTO clown_info
```

```
VALUES
```

```
('Снаглз', 'Болмарт', 'Ж, желтая рубашка, синие  
штаны', 'рожок, зонтик');
```

Бонзо видели в парке Диксон.

Снифлз разбегает на машинке.

Мистера Хобо видели на вечеринке Эрика Грея

Как будет выглядеть таблица clown_info после выполнения команд INSERT? Допишите новые записи.

name	last_seen	appearance	activities
Элси	Дом престарелых Черри Хилл	Ж, рыжие волосы, зеленый костюм, огромные ботинки	шарики, машинки
Пиклз	Вечеринка Джека Грина	М, оранжевые волосы, синий костюм, огромные ботинки	мим
Снаглз	Болмарт	Ж, желтая рубашка, красные штаны	рожок, зонтик
Мистер Хобо	Цирк BG	М, сигара, черные волосы, маленькая шляпа	скрипка
Кларабелл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цветок, синее платье	кричалки, танцы
Скутер	Больница Окленд	М, синие волосы, красный костюм, большой нос	шарики
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы
Бэйб	Автошкола Эрла	Ж, розовый костюм с блестками	эквилибристика, машинки
Бонзо		М, женское платье в горошек	пение, танцы
Снифлз	Заведение Трэиси	М, зелено-фиолетовый костюм, длинный нос	

Возьми в руку карандаш
Решение



Перемещения клоунов

Напишите команды SQL для занесения текущей информации о клоунах в таблицу `clown_info`. Допишите в таблицу записи, которые появятся в ней после выполнения команд `INSERT`.

Зинно теперь поет.

```
INSERT INTO clown_info
```

```
VALUES
```

```
('Зинно', 'Торговый центр Милстоун', 'Ж,  
оранжевый костюм, штаны', 'танцы, пение');
```

Снаглз носит синие штаны.

```
INSERT INTO clown_info
```

```
VALUES
```

```
('Снаглз', 'Болмарт', 'Ж, желтая рубашка,  
синие штаны', 'рожок, зонтик');
```

Бонзо видели в парке Диксон.

```
INSERT INTO clown_info
```

```
VALUES
```

```
('Бонзо', 'Парк Диксон', 'М, женское платье  
в горошек', 'пение, танцы');
```

Снифлз разъезжает на машинке.

```
INSERT INTO clown_info
```

```
VALUES
```

```
('Снифлз', 'Заведение Трэйси', 'М, зелено-  
фиолетовый костюм, длинный нос',  
'разъезжает на машинке');
```

Мистера Хобо видели на вечеринке Эрика Грея.

```
INSERT INTO clown_info
```

```
VALUES
```

```
('Мистер Хобо', 'Вечеринка Эрика Грея', 'М, сига-  
ра, черные волосы, маленькая шляпа', 'скрипка');
```

name	last_seen	appearance	activities
Элси	Дом престарелых Черри Хилл	Ж, рыжие волосы, зеленый костюм, огромные ботинки	шарики, машинки
Пиклз	Вечеринка Джека Грина	М, оранжевые волосы, синий костюм, огромные ботинки	мим
Снаглз	Болмарт	Ж, желтая рубашка, красные штаны	рожок, зонтик
Мистер Хобо	Цирк ВГ	М, сигара, черные волосы, маленькая шляпа	скрипка
Кларабелл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цветок, синее платье	кричалки, танцы
Скутер	Больница Окленд	М, синие волосы, красный костюм, большой нос	шарики
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы
Бэйб	Автошкола Эрла	Ж, розовый костюм с блестками	эквилибристика, машинки
Бонзо		М, женское платье в горошек	пение, танцы
Снифлз	Заведение Трэиси	М, зелено-фиолетовый костюм, длинный нос	
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы, пение
Снаглз	Болмарт	Ж, желтая рубашка, красные штаны	рожок, зонтик
Бонзо	Парк Диксон	М, женское платье в горошек	пение, танцы
Снифлз	Заведение Трэиси	М, зелено-фиолетовый костюм, длинный нос	развозжает на машинке
Мистер Хобо	Цирк ВГ	М, сигара, черные волосы, маленькая шляпа	скрипка

МОЗГОВОЙ ШТУРМ

Как узнать текущее местонахождение конкретного клоуна?

можно ли получать данные в хронологическом порядке?

Как вводятся сведения о клоунах

В службе сбора информации о клоунах работают волонтеры. Иногда отчеты лежат неделю-другую в ожидании ввода данных. А иногда двое сотрудников делят стопку отчетов между собой и вводят данные одновременно.

Учитывая этот факт, рассмотрим содержимое таблицы для клоуна Зиппо. Для выборки данных можно воспользоваться командой SELECT:

```
File Edit Window Help CatchTheClown
SELECT * FROM clown_info WHERE name = 'Зиппо';
```

name	last_seen	appearance	activities
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы, пение
Зиппо	Больница Окленд	Ж, оранжевый костюм, штаны	танцы, пение
Зиппо	Заведение Трэйси	Ж, оранжевый костюм, штаны	танцы, пение
Зиппо	Болмарт	Ж, оранжевый костюм, штаны	танцы, жонглирование
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы, пение
Зиппо	Больница Окленд	Ж, оранжевый костюм, штаны	танцы, пение

Абсолютно одинаковые записи.

Эти записи тоже полностью совпадают.

Информация повторяется снова и снова.

Можно ли обратиться к данным с запросом и получить только последние сведения о Зиппо? Можно ли определить, где этот клоун выступал в последний раз?



Конечно, это
очень просто. Достаточно
обратиться к последней записи.

К сожалению, ничто не гарантирует, что последняя запись окажется самой новой.

Вспомните, что вводом данных одновременно занимается сразу несколько человек, а отчеты могут быть переложены в другом порядке. Но даже если этого не произошло, не рассчитывайте на то, что записи таблицы следуют в хронологическом порядке.

Существует целый ряд внутренних факторов, которые могут повлиять на порядок хранения записей в таблице, в том числе используемая РСУБД и индексы столбцов (об этом позднее).

Ничто не гарантирует, что последняя запись таблицы была добавлена позднее других записей.

Бонзо, у нас проблема

Так как мы не можем быть уверены в том, что последняя запись была добавлена позже остальных, возникает серьезная проблема. Из таблицы можно получить информацию о том, где находились клоуны в тот или иной момент времени. **Но сама таблица создавалась для получения информации о том, где каждого клоуна видели в последний раз!**

И это не все. Обратили внимание на одинаковые записи? Две записи содержат информацию о том, что Зиппо видели в одном и том же месте, за одним и тем же занятием. Дубликаты занимают место и замедляют работу РСУБД с увеличением объема таблиц. В таблице **не должно быть дубликатов**. В одной из следующих глав мы поговорим о том, почему дубликаты нежелательны и как предотвратить их появление. Вы узнаете, как создать таблицы, в которых дубликаты в принципе невозможны. А пока исправим существующую таблицу так, чтобы она содержала полезные данные.

Часть Задаваемые Вопросы

В: Почему нельзя считать, что последняя запись была добавлена позже остальных записей?

О: Порядок следования записей в таблице не гарантирован; к тому же скоро вы узнаете, как изменить порядок записей в полученных результатах. Нет полной уверенности в том, что последняя запись действительно была вставлена последней. Кроме того, порядок следования записей может быть нарушен из-за «человеческого фактора». Предположим, мы вводим две команды INSERT для одного клоуна. Если не хранить в таблице информацию о том, в каком порядке делались эти наблюдения, мы не будем знать, какое из них произошло первым.

В: Допустим, мы помним порядок наблюдений. Так почему не использовать последнюю запись?

О: Немного расширим пример. Информация о клоунах собиралась годами. В штате есть несколько помощников, которые тоже ведут наблюдения и добавляют свои записи. Для некоторых клоунов созданы сотни записей. При выборке мы получим эти сотни записей и нам придется перебирать их до последней — которая, как мы надеемся, была введена позже других.

В: А стоит ли хранить такие данные в таблице? Есть ли смысл во вставке новых записей с сохранением старых?

О: Безусловно. Возьмем текущий пример: таблица в своем текущем виде хранит не только последнее местонахождение каждого клоуна, но и историю его перемещений. Вполне возможно, что эта информация окажется полезной. Проблема в том, что запись не содержит информации о том, когда произошло данное событие. Если добавить в таблицу столбец с датой и временем, мы получим возможность отслеживать перемещения клоунов с гораздо большей точностью.

Но сначала необходимо как-то избавиться от дубликатов, чтобы упростить содержимое таблицы.

В: К концу книги я буду знать, как спроектировать таблицу без дубликатов. А если плохо спроектированная таблица досталась мне от человека, который раньше работал на моем месте?

О: Плохо спроектированные таблицы встречаются сплошь и рядом. Большинству людей, изучающих SQL, приходится исправлять чужие ошибки.

Существует несколько методов борьбы с дубликатами. На данный момент мы еще не располагаем инструментами, необходимыми для исправления плохих данных, но непременно вернемся к ним позднее.

Уничтожение записей командой DELETE

Похоже, нам придется почистить таблицу и избавиться от некоторых записей. Чтобы с таблицей было удобнее работать, для каждого клоуна в ней останется только одна запись. При появлении очередной информации о Зиппо (которая заведомо новее предыдущих) из таблицы удаляются старые данные Зиппо, ставшие неактуальными.

Удаление записей из таблиц осуществляется командой DELETE. В этой команде используются уже знакомые нам условия WHERE. Попробуйте понять, как выглядит синтаксис команды, прежде чем мы покажем его.

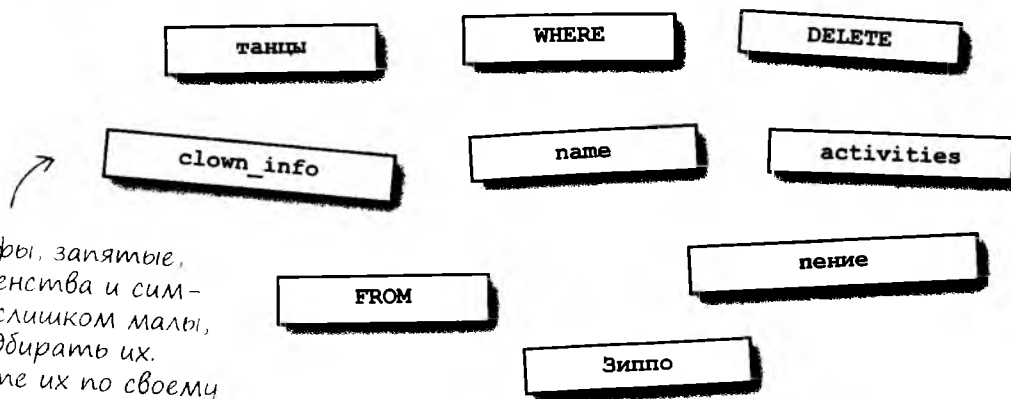
Снова приведем записи Зиппо.

name	last_seen	appearance	activities
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы, пение
Зиппо	Больница Окленд	Ж, оранжевый костюм, штаны	танцы, пение
Зиппо	Заведение Трэйси	Ж, оранжевый костюм, штаны	танцы, пение
Зиппо	Болмарт	Ж, оранжевый костюм, штаны	танцы, жонглирование
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы, пение
Зиппо	Больница Окленд	Ж, оранжевый костюм, штаны	танцы, пение



Магниты с кодами

Мы написали простую команду для уничтожения одной из записей о клоуне Зиппо, но магниты с фрагментами команды полностью перепутались. Соберите фрагменты и укажите, что, по вашему мнению, делает каждая часть новой команды.



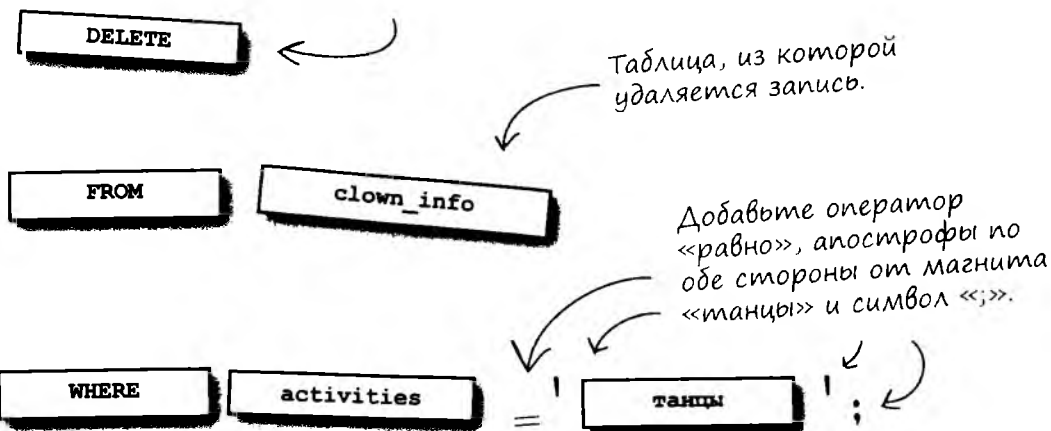
Апострофы, запятые, знаки равенства и символы «<>» слишком малы, чтобы подбирать их. Расставьте их по своему усмотрению.



Магниты с кодами

Мы написали простую команду для уничтожения одной из записей о клоуне Зиппо, но магниты с фрагментами команды полностью перепутались. Соберите фрагменты и укажите, что, по вашему мнению, делает каждая часть новой команды.

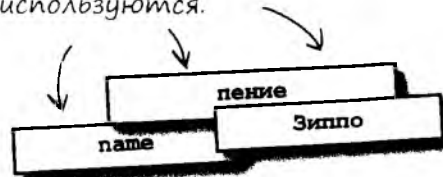
В отличие от команды SELECT, указывать, что именно удаляется, не нужно — команда удаляет всю запись.



И не забудьте добавить условие WHERE, а то команда удалит все записи!

Условие WHERE (см. предыдущую главу) определяет, какая именно запись удаляется командой DELETE.

Эти магниты в команде не используются.



Условие WHERE в командах DELETE работает точно так же, как в командах SELECT.

Использование команды DELETE

Взгляните на построенную нами команду DELETE. Она работает именно так, как и следовало ожидать: все записи, соответствующие условию WHERE, удаляются из таблицы.

```
DELETE FROM clown_info
WHERE
activities = 'танцы';
```

name	last_seen	appearance	activities
Элси	Дом престарелых Черри Хилл	Ж, рыжие волосы, зеленый костюм, огромные ботинки	шарики, машинки
Пиклз	Вечеринка Джека Грина	М, оранжевые волосы, синий костюм, огромные ботинки	мим
Снаглз	Болмарт	Ж, желтая рубашка, красные штаны	рожок, зонтик
Мистер Хобо	Цирк BG	М, сигара, черные волосы, маленькая шляпа	скрипка
Кларабелл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цветок, синее платье	кричалки, танцы
Скутер	Больница Окленд	М, синие волосы, красный костюм, большой нос	шарики
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы
Бэйб	Автошкола Эрла	Ж, розовый костюм с блестками	эквилибристика, машинки
Бонзо		М, женское платье в горошек	пение, танцы
Снифлз	Заведение Трэйси	М, зелено-фиолетовый костюм, длинный нос	
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	пение
Снаглз	Болмарт	Ж, желтая рубашка, красные штаны	рожок, зонтик
Бонзо	Парк Диксон	М, женское платье в горошек	пение, танцы
Снифлз	Заведение Трэйси	М, зелено-фиолетовый костюм, длинный нос	разъезжает на машинке
Мистер Хобо	Вечеринка Эрика Грея	М, сигара, черные волосы, маленькая шляпа	скрипка

← Запись, которая будет удалена.



Как вы думаете, может ли команда DELETE удалить только один столбец из записи?

Правила DELETE

- Команда DELETE не позволяет удалить значение одного столбца или группы столбцов.
- Команда DELETE удаляет из таблицы одну или несколько записей (в зависимости от условия WHERE).
- Мы рассмотрели пример удаления одной записи из таблицы. Также возможно удаление сразу нескольких записей. Для этого критерий выбора удаляемых записей определяется при помощи условия WHERE. Синтаксис условия WHERE полностью совпадает с синтаксисом WHERE в команде SELECT (см. главу 2); в нем могут использоваться все конструкции из главы 2, в том числе LIKE, IN, BETWEEN и операторы сравнения.
- Будьте осторожны — следующая команда удаляет из таблицы все записи:

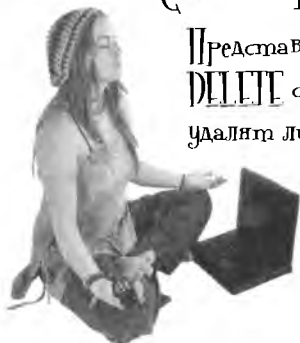
```
DELETE FROM your_table
```

Часто Задаваемые Вопросы

В: Условие WHERE в команде DELETE чем-нибудь отличается от WHERE в команде SELECT?

О: Ничем. Условия WHERE одинаковые, но сами команды SELECT и DELETE существенно отличаются. Команда SELECT возвращает копию столбцов из записей, удовлетворяющих условию WHERE, не изменяя таблицу. Команда DELETE удаляет все записи, удовлетворяющие условию WHERE.

Стань DELETE с условием WHERE



Представьте себя на месте Группы Команд DELETE с условиями WHERE. Определите, удалят ли эти команды какие-либо записи из таблицы.

```
DELETE FROM doughnut_ratings
```

```
WHERE location = 'Krispy King' AND rating <> 6;
```

```
WHERE location = 'Krispy King' AND rating = 3;
```

```
WHERE location = 'Snappy Bagel' AND rating >= 6;
```

```
WHERE location = 'Krispy King' OR rating > 5;
```

```
WHERE location = 'Krispy King' OR rating = 3;
```

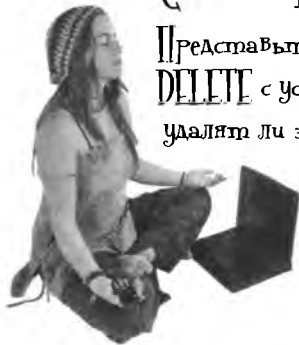
```
WHERE location = 'Snappy Bagel' OR rating = 6;
```

Проведите линию от каждого условия к удаляемой записи (или записям).

doughnut_ratings

location	time	date	type	rating	comments
Krispy King	8:50	27/9	с глазурью	10	почти идеально
Duncan's Donuts	8:59	25/8	NULL	6	жирноваты
Starbuzz Coffee	19:35	24/5	кекс с корицей	5	вчерашние, но вкусные
Duncan's Donuts	19:03	26/4	с вареньем	7	мало варенья

Стань DELETE с условием WHERE. Ответ



Представьте себя на месте Группы команд DELETE с условиями WHERE. Определите, удалят ли эти команды какие-либо записи из таблицы.

DELETE FROM doughnut_ratings

WHERE location = 'Krispy King' AND rating <> 6;

WHERE location = 'Krispy King' AND rating = 3;

Проведите линию от каждого условия к удаляемой записи (или записям):

Нет совпадений, записи не удаляются.

WHERE location = 'Snappy Bagel' AND rating >= 6;

Нет совпадений, записи не удаляются.

WHERE location = 'Krispy King' OR rating > 5;

WHERE location = 'Krispy King' OR rating = 3;

WHERE location = 'Snappy Bagel' OR rating = 6;

Нет совпадений, записи не удаляются.

doughnut_ratings

location	time	date	type	rating	comments
Krispy King	8:50	27/9	с глазурью	10	почти идеально
Duncan's Donuts	8:59	25/8	NULL	6	жирноваты
Starbuzz Coffee	19:35	24/5	кекс с корицей	5	вчерашние, но вкусные
Duncan's Donuts	19:03	26/4	с вареньем	7	мало варенья

Значения NULL могут создать проблемы при будущих запросах. Лучше ввести какое-нибудь значение, чем оставлять NULL в столбце, потому что NULL не находится по условию «равно».

Танцы INSERT-DELETE

В таблице содержится всего одна запись с данными клоуна Кларабелл. Так как мы хотим, чтобы в таблице оставалась всего одна запись с самой новой информацией, нужно создать новую запись и удалить старую.

От другой записи эта отличается только занятием.

Кларабелл танцует в доме престарелых Бельмонт. Ж, розовые волосы, большой цветок, синее платье

Наша задача — сохранить эту информацию в таблице. Для экономии места в таблице на с. 165 приведена всего одна запись.

name	last_seen	appearance	activities
Кларабелл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цветок, синее платье	кричалки, танцы

- 1 Сначала команда INSERT добавляет новую информацию (и старую тоже).

```
INSERT INTO clown_info
VALUES
```

```
('Кларабелл', 'Дом престарелых Бельмонт', 'Ж, розовые
волосы, большой цветок, синее платье', 'танцы');
```

При вставке используются исходные данные, а изменяется только обновленный столбец.

INSERT →

name	last_seen	appearance	activities
Кларабелл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цве-	кричалки, танцы
Кларабелл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цве- ток, синее платье	танцы

- 2 Затем старая запись удаляется командой DELETE с условием WHERE.

```
DELETE FROM clown_info
WHERE
activities = 'кричалки, танцы'
AND name = 'Кларабелл';
```

Для поиска и удаления старой записи используется условие WHERE.

И в таблице остается только одна — новая — запись.

name	last_seen	appearance	activities
Кларабелл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цве- ток, синее платье	танцы

Возьми в руку карандаш



Используйте команды INSERT и DELETE и внесите в таблицу drink_info необходимые изменения. Затем запишите измененные данные в пустой таблице справа.

drink_info

drink_name	cost	carbs	color	ice	calories
Терновник	3	8.4	желтый	Д	33
Голубая луна	2.5	3.2	синий	Д	12
Вот тебе на	3.5	8.6	оранжевый	Д	35
Лаймовый физз	2.5	5.4	зеленый	Д	24
Поцелуй	5.5	42.5	фиолетовый	Д	171
Горячее золото	3.2	32.1	оранжевый	Н	135
Одинокое дерево	3.6	4.2	красный	Д	17
Борзая	4	14	желтый	Д	50
Бабье лето	2.8	7.2	коричневый	Н	30
Лягушка	2.6	21.5	бронзовый	Д	80
Сода плюс	3.8	4.7	красный	Н	19

Уменьшите калорийность «Поцелуя» до 170.

Замените желтый цвет «золотистым».

drink_info

drink_name	cost	carbs	color	ice	calories
Терновник					
Голубая луна					
Вот тебе на					
Лаймовый физз					
Поцелуй					
Горячее золото					
Одинокое дерево					
Борзая					
Бабье лето					
Лягушка					
Сода плюс					

Очередное каверзное упражнение?

Для всех напитков, стоящих \$2.50, поднять цену до \$3.50, а для напитков с ценой \$3.50 — до \$4.50.





Возьми в руку карандаш Решение

Используйте команды INSERT и DELETE и внесите в таблицу drink_info необходимые изменения. Затем запишите измененные данные в пустой таблице справа.

drink_info

drink_name	cost	carbs	color	ice	calories
Терновник	3	8.4	желтый	Д	33
Голубая луна	2.5	3.2	синий	Д	12
Вот тебе на	3.5	8.6	оранжевый	Д	35
Лаймовый физз	2.5	5.4	зеленый	Д	24
Поцелуй	5.5	42.5	фиолетовый	Д	171
Горячее золото	3.2	32.1	оранжевый	Н	135
Одинокое дерево	3.6	4.2	красный	Д	17
Борзая	4	14	желтый	Д	50
Бабье лето	2.8	7.2	коричневый	Н	30
Лягушка	2.6	21.5	бронзовый	Д	80
Сода плюс	3.8	4.7	красный	Н	19

Уменьшите калорийность «Поцелуя» до 170.

```
INSERT INTO drink_info VALUES ('Поцелуй', 5.5, 42.5,
'фиолетовый', 'Д', 170);
```

```
DELETE FROM drink_info WHERE calories = 171;
```

Замените желтый цвет «золотистым».

```
INSERT INTO drink_info
VALUES ('Терновник', 3, 8.4, 'золотистый', 'Д', 33),
('Борзая', 4, 14, 'золотистый', 'Д', 50);
```

```
DELETE FROM drink_info WHERE color = 'желтый';
```

drink_info

drink_name	cost	carbs	color	ice	calories
Терновник	3	8.4	золотистый	Д	33
Голубая луна	3.5	3.2	синий	Д	12
Вот тебе на	4.5	8.6	оранжевый	Д	35
Лаймовый физз	3.5	5.4	зеленый	Д	24
Поцелуй	5.5	42.5	фиолетовый	Д	170
Горячее золото	3.2	32.1	оранжевый	Н	135
Одинокое дерево	3.6	4.2	красный	Д	17
Борзая	4	14	золотистый	Д	50
Бабье лето	2.8	7.2	коричневый	Н	30
Лягушка	2.6	21.5	бронзовый	Д	80
Сода плюс	3.8	4.7	красный	Н	19

↑
Так должна выглядеть таблица после внесения изменений. В вашей базе данных порядок записей может быть другим, но это совершенно неважно.

Не такое уж каверзное, но подумать придется. Если сначала поднять цену напитков с \$2.50 до \$3.50, а потом с \$3.50 до \$4.50, то «Голубая луна» подорожает дважды. Вместо этого нужно сначала изменить большую цену (с \$3.50 до \$4.50), а потом меньшую («Голубая луна» — с \$2.50 до \$3.50).

Очередное каверзное упражнение?

Для всех напитков, стоящих \$2.50, поднять цену до \$3.50, а для напитков с ценой \$3.50 — до \$4.50.

```
INSERT INTO drink_info VALUES ('Вот тебе на', 4.5, 8.6,
'оранжевый', 'Д', 35);
```

```
DELETE FROM drink_info WHERE cost = 3.5;
```

```
INSERT INTO drink_info VALUES ('Голубая луна', 3.5, 3.2, 'синий', 'Д', 12),
('Лаймовый физз', 3.5, 5.4, 'зеленый', 'Д', 24);
```

```
DELETE FROM drink_info WHERE cost = 2.5;
```

Дополнительные баллы, если вы объединили две команды INSERT в одну!



Будьте внимательны при выполнении DELETE

При выполнении команды DELETE всегда существует опасность случайного удаления записей, которые вы удалять не собирались. Допустим, в таблицу была добавлена новая запись о Мистере Хобо:

Добавляемая информация и команда INSERT для ее добавления.

Мистера Хобо видели в заведении Трэйси.

**INSERT INTO clown_info
VALUES**

('Мистер Хобо', 'Заведение Трэйси', 'М, сигара, черные волосы, маленькая шляпа', 'скрипка');

Будьте внимательны при использовании DELETE.

Убедитесь в том, что условие WHERE точно описывает удаляемые записи и не включает ничего лишнего.

name	last_seen	appearance	activities
Элси	Дом престарелых Черри Хилл	Ж, рыжие волосы, зеленый костюм, огромные ботинки	шарики, машинки
Пиклз	Вечеринка Джека Грина	М, оранжевые волосы, синий костюм, огромные ботинки	мим
Снаглз	Болмарт	Ж, желтая рубашка, красные штаны	рожок, зонтик
Мистер Хобо	Цирк BG	М, сигара, черные волосы, маленькая шляпа	скрипка
Кларабелл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цветок, синее платье	кричалки, танцы
Скутер	Больница Окленд	М, синие волосы, красный костюм, большой нос	шарики
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы, пение
Бэйб	Автошкола Эрла	Ж, розовый костюм с блестками	эквилибристика, машинки
Бонзо		М, женское платье в горошек	пение, танцы
Снифлз	Заведение Трэйси	М, зелено-фиолетовый костюм, длинный нос	
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	пение
Снаглз	Болмарт	Ж, желтая рубашка, красные штаны	рожок, зонтик
Бонзо	Парк Диксон	М, женское платье в горошек	пение, танцы
Снифлз	Заведение Трэйси	М, зелено-фиолетовый костюм, длинный нос	разъезжает на машинке
Мистер Хобо	Парк Диксон	М, сигара, черные волосы, маленькая шляпа	скрипка
мистер Хобо	Заведение Трэйси	М, сигара, черные волосы, маленькая шляпа	скрипка

DELETE

А теперь станьте командой DELETE.



Станьте командой DELETE

Ниже приведены условия WHERE для серии команд DELETE,

которые должны удалить лишние записи из таблицы clown_info table на предыдущей странице. Определите, какие условия действительно полезны, а какие лишь создают новые проблемы.

```
DELETE FROM clown_info
```

```
WHERE last_seen = 'Больница Окленд';
```

```
WHERE activities = 'скрипка';
```

```
WHERE last_seen = 'Парк Диксон'
AND name = 'Мистер Хобо';
```

```
WHERE last_seen = 'Больница Окленд' AND
last_seen = 'Парк Диксон';
```

```
WHERE last_seen = 'Больница Окленд' OR
last_seen = 'Парк Диксон';
```

```
WHERE name = 'Мистер Хобо'
OR last_seen = 'Больница Окленд';
```

А теперь напишите одну команду DELETE, которая удалит лишние записи Мистера Хобо, не затрагивая других записей.

Команда полезна? Если нет — укажите, почему.



Станьте командой DELETE. Ответ

Ниже приведены условия WHERE для серии команд DELETE, которые должны удалить лишние записи из таблицы clown_info table на предыдущей странице. Определите, какие условия действительно полезны, а какие лишь создают новые проблемы.

```
DELETE FROM clown_info
```

↙ Запись Скутера тоже удовлетворяет этому условию.

```
WHERE last_seen = 'Больница Окленд';
```

↙ Новая запись не должна удаляться.

```
WHERE activities = 'скрипка';
```

```
WHERE last_seen = 'Парк Диксон'
```

```
AND name = 'Mr. Хобо';
```

↙ Связка AND означает, что оба условия должны быть истинными.

```
WHERE last_seen = 'Больница Окленд'
```

```
AND last_seen = 'Парк Диксон';
```

```
WHERE last_seen = 'Больница Окленд'
```

```
OR last_seen = 'Парк Диксон';
```

```
WHERE name = 'Мистер Хобо'
```

```
OR last_seen = 'Больница Окленд';
```

А теперь напишите одну команду DELETE, которая удалит лишние записи Мистера Хобо, не затрагивая других записей.

Команда полезна? Если нет — укажите, почему.

Удаляет только одну запись Мистера

Хобо. Также удаляет запись Скутера.

Удаляет все записи Мистера Хобо,
в том числе и новую.

Удаляет только одну из старых записей
Мистера Хобо.

Ничего не удаляет.


Вместе со старыми записями Мистера
Хобо удаляет записи Бонзо и Скутера.

Удаляет все записи Мистера Хобо,
в том числе и новую, а также запись
Скутера.

```
DELETE FROM clown_info
```

```
WHERE name = 'Мистер Хобо'
```

```
AND last_seen <> 'Заведение Трэиси';
```



Похоже, вы удалили что-то лишнее? Возможно, стоило сначала выполнить команду `SELECT` и посмотреть, какие записи будут удалены с конкретным условием `WHERE`.

Точно! Если у вас нет полной уверенности относительно того, что условие `WHERE` удалит только нужные записи, сначала выполните команду `SELECT`.

Так как обе команды используют одинаковые условия `WHERE`, то команда `SELECT` вернет записи, которые будут удалены командой `DELETE` с этим же условием `WHERE`.

Этот нехитрый прием предотвратит случайное удаление посторонних записей, а также поможет убедиться в том, что из таблицы будут удалены все нежелательные записи.

Проблемы с неточными условиями DELETE

Правильно написать команду DELETE сложно. Стоит допустить малейшую неточность, и команда удалит посторонние данные. Для предотвращения удаления лишних данных в схему INSERT-DELETE включается дополнительный шаг.

Вот как выглядит новый план ИЗ ТРЕХ ШАГОВ:

Чтобы не удалить посторонние записи, сначала выполните команду SELECT.

- 1 Сначала произведите выборку удаляемой записи командой SELECT. Убедитесь, что удаляются только те записи, которые вы собирались удалить — и никаких посторонних записей.

```
SELECT FROM clown_info  
WHERE  
activities = 'танцы';
```



name	last_seen	appearance	activities
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы

- 2 Теперь добавьте новую запись командой INSERT.

```
INSERT INTO clown_info  
VALUES  
( 'Зиппо', 'Торговый центр Милстоун',  
'Ж, оранжевый костюм, штаны', 'танцы, пение' );
```

Измените только тот столбец, который нужно изменить.



name	last_seen	appearance	activities
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы, пение

- 3 Наконец, удалите старые записи командой DELETE с тем же условием WHERE, которое использовалось в команде SELECT на первом шаге.

```
DELETE FROM clown_info
WHERE
activities = 'танцы;
```

Используйте условие WHERE из команды SELECT (шаг 1) для поиска и удаления старой записи.

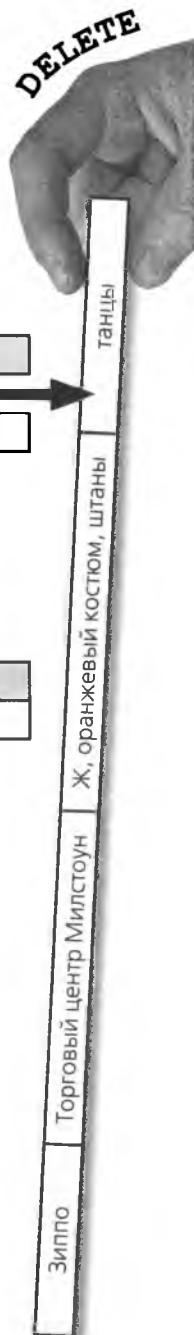
name	last_seen	appearance	activities
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы, пение

В таблице остается только новая запись.

name	last_seen	appearance	activities
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы, пение



А как было бы здорово, если бы все можно было сделать за один шаг, не беспокоясь о том, что новая запись будет удалена вместе со старой... Но я знаю, что это всего лишь мечты...



Изменение данных командой UPDATE

Итак, вы достаточно хорошо разобрались в командах INSERT и DELETE, чтобы обеспечить актуальность данных в своих таблицах. Также мы рассмотрели возможность совместного использования этих команд для косвенной модификации отдельных записей.

Но вместо того чтобы вставлять в таблицу новую запись и удалять старую, можно обновить существующую запись, изменив в ней значения только тех столбцов, которые требуется изменить.

Для этой цели используется команда SQL UPDATE. Она обновляет столбец (или столбцы) новыми значениями. Как и в случае с командами SELECT и DELETE, для выбора обновляемой записи (или записей) можно задать условие WHERE.

Пример использования команды UPDATE:

```
UPDATE doughnut_ratings
SET
type = 'глазированные'
WHERE type = 'с глазурью';
```

Здесь задается новое значение.

Знакомое условие WHERE, как и в командах SELECT и DELETE.

Ключевое слово SET сообщает РСУБД о том, что в столбце, имя которого указано перед знаком =, должно быть сохранено значение, указанное после знака =. В приведенном примере в столбец type записывается текст 'глазированные'. Условие WHERE указывает, что изменения вносятся только в строках, у которых столбец type содержит значение 'с глазурью'.

doughnut_ratings

location	time	date	type	rating	comments
Krispy King	8:50	27/9	с глазурью	10	почти идеально
Duncan's Donuts	8:59	25/8	NULL	6	жирноваты
Starbuzz Coffee	19:35	24/5	кекс с корицей	5	вчерашние, но вкусные
Duncan's Donuts	19:03	26/4	с вареньем	7	мало варенья

doughnut_ratings

location	time	date	type	rating	comments
Krispy King	8:50	27/9	глазированные	10	почти идеально
Duncan's Donuts	8:59	25/8	NULL	6	жирноваты
Starbuzz Coffee	19:35	24/5	кекс с корицей	5	вчерашние, но вкусные
Duncan's Donuts	19:03	26/4	с вареньем	7	мало варенья

Правила UPDATE

- Команда UPDATE может использоваться для изменения значения одного столбца или группы столбцов. Включите дополнительные пары столбец = значение в условии SET и поставьте запятую после каждой пары:

```
UPDATE your_table
SET первый_столбец= 'новое_значение',
    второй_столбец = 'старое_значение';
```

- Команда UPDATE может изменять одну запись или несколько записей в зависимости от условия WHERE.

Часть Задаваемые Вопросы

В: Что произойдет, если условие WHERE не задано?

О: Каждый столбец, указанный в условии SET, будет обновлен новым значением.

В: В запросе SQL на предыдущей странице два оператора =, которые используются для разных целей. Это нормально?

О: Абсолютно. Оператор = в условии SET означает «присвоить столбцу указанное значение», а оператор = в условии WHERE проверяет, равно ли текущее значение столбца значению, указанному после знака.

В: Могу ли использовать следующую команду для выполнения той же операции?

```
UPDATE doughnut_ratings SET type = 'глазиро-
ванные' WHERE location = 'Krispy King';
```

О: Да, можете. Команда внесет те же изменения в ту же запись. Для таблицы из четырех записей это нормально, но при работе с таблицей, содержащей сотни и тысячи записей, вам пришлось бы изменять по отдельности каждую запись, относящуюся к Krispy King.

В: Можно ли убедиться в том, что обновление затронет только нужные мне записи?

О: Как и в случае с командой DELETE, если вы не уверены в том, что условие WHERE определяет только нужные записи — сначала выполните команду SELECT!

В: Может ли команда содержать более одной секции SET?

О: Нет, но это и не нужно. Все столбцы с новыми значениями могут перечисляться в одном условии SET, как показано выше.

UPDATE как замена INSERT-DELETE

При выполнении команды **UPDATE** из таблицы *ничего не удаляется*. Вместо удаления старая запись обновляется новыми данными.

Команда начинается с ключевого слова **UPDATE**...

...далее идет имя таблицы, содержащей обновляемую запись.

SET определяет изменения, вносимые в запись.

```
UPDATE table_name  
SET имя_столбца = новое_значение  
WHERE имя_столбца = старое_значение;
```

Условие **WHERE** определяет записи, в которые вносятся изменения.

Команда UPDATE
заменяет комбинацию
INSERT / DELETE.

Рассмотрим на примере команды, работающей с таблицей `clown_info`.

Обновляется запись из таблицы `clown_info`.

В столбец `last_seen` записывается строка 'Заведение Трэйси'.

```
UPDATE clown_info  
SET last_seen = 'Заведение Трэйси'  
WHERE name = 'Мистер Хобо'  
AND last_seen = 'Парк Диксон';
```

Условие **WHERE** определяет запись, в которую вносятся изменения — в данном случае запись Мистера Хобо, у которой столбец `last_seen` содержит значение 'Парк Диксон'.

UPDATE в действии

Команда UPDATE заменяет текущее значение столбца last_seen ('Парк Диксон') значением 'Заведение Трэйси'.

Мистера Хобо
видели в «Заведении Трэйси».

Информация, которая должна быть добавлена в таблицу, и команда UPDATE для ее добавления.

```
UPDATE clown_info
SET last_seen = 'Заведение Трэйси'
WHERE name = 'Мистер Хобо'
AND last_seen = 'Вечеринка Эрика Грея';
```

name	last_seen	appearance	activities
Элси	Дом престарелых Черри Хилл	Ж, рыжие волосы, зеленый костюм, огромные ботинки	шарики, машинки
Пиклз	Вечеринка Джека Грина	М, оранжевые волосы, синий костюм, огромные ботинки	мим
Снаглз	Болмарт	Ж, желтая рубашка, красные штаны	рожок, зонтик
Мистер Хобо	Цирк BG	М, сигара, черные волосы, маленькая шляпа	скрипка
Кэти Хилл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цветок, синее платье	кричалки, танцы
Стив	Больница Окленд	М, синие волосы, красный костюм, большой нос	шарики
Зип	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы
	Шоколада Эрла	Ж, розовый костюм с блестками	эквилибристика, машинки
		М, женское платье в горошек	пение, танцы
	Заведение Трэйси	М, зелено-фиолетовый костюм, длинный нос	
	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	пение
	Ball-Mart	Ж, красные штаны	рожок, зонтик
		Ж, polka dotted dress	пение, танцы
		Ж, костюм, длинный нос	разъезжает на машинке
Мистер Хобо	Заведение Трэйси	М, сигара, black hair, tiny hat	скрипка

Команда UPDATE выполняет замену «на месте», что исключает опасность удаления посторонних данных (хотя перезапись существующих данных по-прежнему возможна).

Возьми в руку карандаш



Обновление информации о клоунах

На этот раз давайте выполним обновление правильным способом. Напишите команду UPDATE для каждого наблюдения (мы уже написали одну, чтобы вам было проще). Потом запишите, как будет выглядеть таблица после выполнения всех команд UPDATE.

Зинно теперь поет.

```
UPDATE clown_info
```

```
SET activities = 'пение'
```

```
WHERE name = 'Зинно';
```

Снагз носит синие штаны.

Бонзо видели в парке Диксон.

Снифлз разъезжает на машинке.

Мистера Хобо видели на вечеринке Эрика Грея.

name	last_seen	appearance	activities
Элси	Дом престарелых Черри Хилл	Ж, рыжие волосы, зеленый костюм, огромные ботинки	шарики, машинки
Пиклз	Вечеринка Джека Грина	М, оранжевые волосы, синий костюм, огромные ботинки	мим
Снаглз	Болмарт	Ж, желтая рубашка, красные штаны	рожок, зонтик
Мистер Хобо	Цирк BG	М, сигара, черные волосы, маленькая шляпа	скрипка
Кларабелл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цветок, синее платье	кричалки, танцы
Скутер	Больница Окленд	М, синие волосы, красный костюм, большой нос	шарики
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы
Бэйб	Автошкола Эрла	Ж, розовый костюм с блестками	эквилибристика, машинки
Бонзо		М, женское платье в горошек	пение, танцы
Снифлз	Заведение Трэйси	М, зелено-фиолетовый костюм, длинный нос	

<i>name</i>	<i>last_seen</i>	<i>appearance</i>	<i>activities</i>
Элси	Дом престарелых Черри Хилл	Ж, рыжие волосы, зеленый костюм, огромные ботинки	шарики, машинки
Пиклз	Вечеринка Джека Грина	М, оранжевые волосы, синий костюм, огромные ботинки	мим
Снаглз			
Мистер Хобо			
Кларабелл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цветок, синее платье	кричалки, танцы
Скутер	Больница Окленд	М, синие волосы, красный костюм, большой нос	шарики
Зиппо			
Бэйб	Автошкола Эрла	Ж, розовый костюм с блестками	эквилибристика, машинки
Бонзо			
Снифлз			

Возьми в руку карандаш
Решение



Обновление информации о клоунах

На этот раз давайте выполним обновление правильным способом. Напишите команду UPDATE для каждого наблюдения (мы уже написали одну, чтобы вам было проще). Потом запишите, как будет выглядеть таблица после выполнения всех команд UPDATE.

Зинно теперь поет.

```
UPDATE clown_info
SET activities = 'пение'
WHERE name = 'Зинно';
```

Не забудьте включить другую информацию из столбца appearance — она не должна потеряться при обновлении.

Снагз носит синие штаны.

```
UPDATE clown_info
SET appearance = 'Ж, желтая рубашка, синие штаны'
WHERE name = 'Snuggles';
```

Бонзо видели в парке Диксон.

```
UPDATE clown_info
SET last_seen = 'Парк Диксон'
WHERE name = 'Бонзо';
```

Снифлз развезжает на машинке.

```
UPDATE clown_info
SET activities = 'развезжает на машинке'
WHERE name = 'Снифлз';
```

Мистера Хобо видели на вечеринке Эрика Грея.

```
UPDATE clown_info
SET last_seen = 'Вечеринка Эрика Грея'
WHERE name = 'Мистер Хобо';
```


name	last_seen	appearance	activities
Элси	Дом престарелых Черри Хилл	Ж, рыжие волосы, зеленый костюм, огромные ботинки	шарики, машинки
Пиклз	Вечеринка Джека Грина	М, оранжевые волосы, синий костюм, огромные ботинки	мим
Снаглз	Болмарт	Ж, желтая рубашка, красные штаны	рожок, зонтик
Мистер Хобо	Цирк BG	М, сигара, черные волосы, маленькая шляпа	скрипка
Кларабелл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цветок, синее платье	кричалки, танцы
Скутер	Больница Окленд	М, синие волосы, красный костюм, большой нос	шарики
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы
Бэйб	Автошкола Эрла	Ж, розовый костюм с блестками	эквилибристика, машинки
Бонзо		М, женское платье в горошек	пение, танцы
Снифлз	Заведение Трэиси	М, зелено-фиолетовый костюм, длинный нос	

Серые записи остаются неизменными.

name	last_seen	appearance	activities
Элси	Дом престарелых Черри Хилл	Ж, рыжие волосы, зеленый костюм, огромные ботинки	шарики, машинки
Пиклз	Вечеринка Джека Грина	М, оранжевые волосы, синий костюм, огромные ботинки	мим
Снаглз	Болмарт	Ж, желтая рубашка, синие штаны	рожок, зонтик
Мистер Хобо	Вечеринка Эрика Грея	М, сигара, черные волосы, маленькая шляпа	скрипка
Кларабелл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цветок, синее платье	кричалки, танцы
Скутер	Больница Окленд	М, синие волосы, красный костюм, большой нос	шарики
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	пение
Бэйб	Автошкола Эрла	Ж, розовый костюм с блестками	эквилибристика, машинки
Бонзо	Парк Диксон	М, женское платье в горошек	пение, танцы
Снифлз	Заведение Трэиси	М, зелено-фиолетовый костюм, длинный нос	разбегает на машинке

Изменились только столбцы, указанные в условии SET команды UPDATE. Наконец-то мы заполнили пробелы в таблице на с. 155.

Обновление цен

Помните, как мы обновляли цены в таблице `drink_info`? Напитки с ценой \$2.50 подорожали до \$3.50, а напитки с ценой \$3.50 — до \$4.50.

`drink_info`

<code>drink_name</code>	<code>cost</code>	<code>carbs</code>	<code>color</code>	<code>ice</code>	<code>calories</code>
Терновник	3	8.4	желтый	Д	33
Голубая луна	2.5	3.2	синий	Д	12
Вот тебе на	3.5	8.6	оранжевый	Д	35
Лаймовый физз	2.5	5.4	зеленый	Д	24
Поцелуй	5.5	42.5	фиолетовый	Д	171
Горячее золото	3.2	32.1	оранжевый	Н	135
Одинокое дерево	3.6	4.2	красный	Д	17
Борзая	4	14	желтый	Д	50
Бабье лето	2.8	7.2	коричневый	Н	30
Лягушка	2.6	21.5	бронзовый	Д	80
Сода плюс	3.8	4.7	красный	Н	19

Как подойти к решению этой задачи? Один из возможных способов с использованием команды `UPDATE` — перебрать все записи по отдельности и написать серию команд `UPDATE` следующего вида:

```
UPDATE drink_info
SET cost = 3.5 ← Цена увеличивается на $1.
WHERE drink_name = 'Голубая луна';
```

↑
Условие `WHERE` однозначно идентифицирует обновляемые записи.

Возьми в руку карандаш



Запишите команды UPDATE для каждой записи из таблицы drinks_info, чтобы добавить доллар к стоимости каждого напитка.

drink_name	cost	carbs	color	ice	calories
Терновник	3	8.4	желтый	Д	33
Голубая луна	2.5	3.2	синий	Д	12
Вот тебе на	3.5	8.6	оранжевый	Д	35
Лаймовый	5.4	4	зеленый	Д	24
Пицца	171				
Горячая					

Минутку, а почему мы должны проделывать все это вручную? Нет ли какого-нибудь оператора, который можно использовать в UPDATE, вместо того чтобы изменять каждую запись по отдельности?

Вы правы.

Действительно, существует удобный оператор, который идеально подходит для таких ситуаций. Давайте посмотрим, как обновить цены напитков без ручного перебора всех записей... и без риска повторного изменения ранее измененных данных.

Достаточно одной команды UPDATE

В столбце `cost` хранится число. В SQL с **числовыми столбцами** можно выполнять *основные математические операции*. Так, значение столбца `cost` можно увеличить на 1 для каждой записи в таблице, которую требуется изменить. Вот как это делается:

```
UPDATE drink_info
SET cost = cost + 1;
WHERE
drink_name='Голубая луна'
OR
drink_name='Вот тебе на'
OR
drink_name= 'Лаймовый физз';
```

↑
увеличить на 1 цену в каждой из трех записей, которые требуется изменить (напитки за \$2.50 и \$3.50).

Часто задаваемые вопросы

В: Можно ли использовать вычитание? А какие еще операции?

О: Умножение, деление, вычитание — можете использовать любые из них. И конечно, в операциях могут использоваться другие числа, не только 1.

В: Можно привести пример с использованием умножения?

О: Конечно. Допустим, имеется таблица со списком товаров, каждый товар имеет свою цену. Вы можете воспользоваться командой `UPDATE` и умножить цену каждого товара на фиксированную величину, чтобы вычислить цену с налогом.


В: Какие еще операции можно выполнять с данными, кроме простых математических?

О: Их довольно много. Позднее вы узнаете, что можно сделать с текстовыми столбцами и какие еще операции могут выполняться с числовыми столбцами.

В: Ну например? Хотя бы наметьте.

О: Ладно — например, функция `UPPER()` преобразует все значение текстового столбца в таблице к верхнему регистру. А функция `LOWER()`, как нетрудно догадаться, преобразует текст к нижнему регистру.

Команда UPDATE может работать с группами записей таблицы. Используйте ее с основными математическими операторами для обновления числовых данных.



Хорошо, что данные можно обновлять. Но я хотел бы понять, как с самого начала правильно спроектировать структуру таблицы.

Данные изменяются, поэтому очень важно уметь обновлять их.

Но чем лучше будет спроектирована таблица при ее создании, тем меньше обновлений потребуется позднее. Правильно выбранная структура таблицы позволит вам полностью сосредоточиться на ее содержимом.

Заинтересовались? В следующей главе мы вплотную займемся вопросами проектирования таблиц.



Новые инструменты

Глава 3 скоро останется в прошлом.

Ниже приведена краткая сводка новых команд SQL, которые вы узнали в этой главе. Полный список инструментов приведен в приложении III.

UPDATE

Команда обновляет текущие значения столбца или группы столбцов. В ней тоже используется условие **WHERE**.

DELETE

Команда удаляет записи данных из таблицы. Используйте условие **WHERE** для точного определения удаляемых записей.

SET

Ключевое слово используется в команде **UPDATE** для изменения значения существующего столбца.

4 Проектирование таблиц

Как важно быть нормальным ✨



До настоящего момента мы не особо задумывались при создании таблиц. Работают — и ладно; в конце концов, с ними можно выполнять команды `SELECT`, `INSERT`, `DELETE` и `UPDATE`. Но при увеличении объема данных постепенно становится ясно, что следовало бы сделать при создании таблицы для упрощения условий `WHERE`: ее следовало бы сделать более нормальной.

Две таблицы

Джек и Марк создали таблицы для хранения информации о рекордах рыбной ловли. В таблице Марка имеются столбцы для бытового и научного названия рыбы, ее веса и места, где она была поймана. Столбца для имени человека, поймавшего рыбу, в этой таблице нет.

fish_info

common	species	locatlon	weight
большеротый окунь	M. salmoides	Монтгомери Лейк, GA	22 фт 4 унц
судак	S. vitreus	Олд Хикори Лейк, TN	25 фт 0 унц
лосось Кларка	O. Clarki	Пирамид Лейк, NV	41 фт 0 унц
желтый окунь	P. Flavescens	Бордентаун, NJ	4 фт 3 унц
синезаберник	L. Macrochirus	Кетона Лейк, AL	4 фт 12 унц
панцирник	L. Osseus	Тринити Ривер, TX	50 фт 5 унц
белый краппи	P. annularis	Дамба Энид, MS	5 фт 3 унц
красноперая щука	E. americanus	Дьюарт Лейк, IN	1 фт 0 унц
серебристый карась	C. auratus	Лейк Ходжес, CA	6 фт 10 унц
чавыча	O. Tshawytscha	Кенай Ривер, АК	97 фт 4 унц

Таблица состоит из четырех столбцов. Сравните с таблицей fish_records на следующей странице.



Я ихтиолог. В своей таблице я буду проводить поиск только по бытовому или научному названию, чтобы узнать вес и место вылова рыбы.

Марк

В таблице Джека тоже хранятся бытовые и научные названия рыб, но в ней также имеются столбцы для имени и фамилии рыбака, а место вылова разбито на два столбца: название водоема хранится отдельно от штата.

Эта таблица тоже содержит информацию о рыболовных рекордах, но в ней почти вдвое больше столбцов.

fish_records

first_name	last_name	common	location	state	weight	date
Джордж	Перри	большеротый окунь	Монтгомери Лейк	GA	22 фт 4 унц	2/6/1932
Мабри	Харпер	судак	Олд Хикори Лейк	TN	25 фт 0 унц	2/8/1960
Джон	Скиммерхорн	лосось Кларка	Пирамид Лейк	NV	41 фт 0 унц	1/12/1925
С.С.	Эббот	желтый окунь	Бордентаун	NJ	4 фт 3 унц	1/5/1865
Т.С.	Хадсон	синежаберник	Кетона Лейк	AL	4 фт 12 унц	9/4/1950
Таунсенд	Миллер	панцирник	Тринити Ривер	TX	50 фт 5 унц	30/7/1954
Фред	Брайт	белый краппи	Дамба Энид	MS	5 фт 3 унц	31/7/1957
Майк	Берг	красноперая щука	Дьюарт Лейк	IN	1 фт 0 унц	9/6/1990
Флорентино	Абена	серебристый карась	Лейк Ходжес	CA	6 фт 10 унц	17/4/1996
Лес	Андерсон	чавыча	Кенай Ривер	AK	97 фт 4 унц	17/5/1985

Возьми в руку карандаш



Напишите запрос для каждой таблицы, возвращающий все записи для штата Нью-Джерси.

А я пишу статьи для рыболовного журнала. И мне нужно знать имена рыбаков, даты и места рекордного вылова.



Джек

Возьми в руку карандаш



Решение

Напишите для каждой таблицы запрос, возвращающий все записи для штата Нью-Джерси.

Для получения результатов из строки «город, штат» приходится использовать ключевое слово LIKE.

Мне почти никогда не приходится искать записи по штату. Я храню название штата в одном столбце с названием города.

```
SELECT * FROM fish_info
.....
WHERE location LIKE '%NJ';
```




common	species	location	weight
желтый окунь	P. Flavescens	Бордентаун, NJ	4 фт 3 унц

Запрос напрямую обращается к столбцу state.

А мне часто приходится искать по штату, поэтому я выделил название штата в отдельный столбец при создании таблицы.

```
SELECT * FROM fish_records
.....
WHERE state = 'NJ';
```



first_name	last_name	common	location	state	weight	date
C.C.	Эббот	желтый окунь	Бордентаун	NJ	4 фт 3 унц	1/5/1865

Часть
**Задаваемые
 Вопросы**

В: Выходит, таблица Джека лучше, чем таблица Марка?

О: Нет. Это разные таблицы с разными целями. Марку редко приходится проводить поиск по штату, потому что его интересуют только названия (бытовое и научное) выловленных рыб и их вес.

С другой стороны, Джеку *потребуется* искать данные по штату в своих запросах. Именно поэтому он создал в своей таблице отдельный столбец, чтобы было удобнее указывать штат в запросах.

В: Следует ли избегать оператора LIKE в запросах? Что в нем плохого?

О: В операторе LIKE нет ничего плохого, но он усложняет структуру запроса и повышает риск получения посторонних результатов. Если столбцы содержат сложную информацию, LIKE не позволяет легко и однозначно определить критерий поиска.

В: Почему короткие запросы лучше длинных?

О: Чем проще запрос, тем лучше. С увеличением объема базы данных и добавлением новых таблиц запросы усложняются. Начинать с самых простых запросов, позднее вы их оцените.

В: Значит, в моих столбцах всегда должны храниться как можно меньшие фрагменты данных?

О: Не обязательно. Как показывает пример с таблицами Марка и Джека, все зависит от *использования* данных. Для примера представьте таблицы со списком машин, предназначенные для автомеханика и продавца. Механику необходима подробная информация о каждой машине, а продавцу может быть достаточно фирмы-производителя, модели и номера.

В: Допустим, в записи хранится почтовый адрес. Почему бы не создать один столбец для хранения полного адреса и несколько других столбцов для хранения его составных частей?

О: Дублирование данных поначалу может показаться вполне разумной мерой, но подумайте, сколько лишнего пространства будет расходоваться на жестком диске, если база данных вырастет до значительных размеров. А еще при дублировании данных в команду UPDATE должно включаться дополнительное лишнее условие, и вы должны помнить о нем при каждом изменении данных.

Давайте более подробно разберемся в том, как спроектировать оптимальную структуру таблицы для ваших целей..

Структура таблицы зависит от того, как вы собираетесь использовать свои данные.

 **МОЗГОВОЙ
 ШТУРМ**

SQL — язык, используемый реляционными базами данных. Как вы думаете, что означает термин «реляционный» в контексте баз данных SQL?

Логические связи как суть таблицы

SQL известен как язык Реляционных Систем Управления Базами Данных (РСУБД). Термин запоминать не обязательно, нас интересует только слово «РЕЛЯЦИОННЫХ*». Для нас оно означает, прежде всего, одно: чтобы правильно спроектировать таблицу, необходимо продумать, как столбцы связываются друг с другом для описания некоторого объекта.

Ваша задача — описать объект при помощи столбцов так, чтобы по возможности упростить получение необходимой информации. Конечно, выбор во многом зависит от ваших требований к таблице, но существуют некоторые общие меры, которые следует принять при выборе структуры таблицы.

1. Выберите один объект, который должна описывать таблица.

Какой основной объект описывает ваша таблица?

2. Составьте список того, что необходимо знать об этом объекте при работе с таблицей.

Как будет использоваться ваша таблица?

3. Используя список, разбейте необходимую информацию об объекте на фрагменты, которые могут использоваться для определения структуры таблицы.

Как проще всего запросить данные из таблицы?

* Встречается мнение, что термин «РЕЛЯЦИОННЫЙ» относится к логическим связям между *таблицами*. Это неверно.



Упражнение

Сможете ли вы определить столбцы таблицы по тем словам, которыми ихтиолог Марк описывает выборку данных из таблицы? Запишите имена столбцов в прямоугольничках.

Я провожу поиск по бытовому или научному названию рыбы и хочу узнать вес и место вылова.

Теперь ваша очередь. Напишите аналогичную фразу для Джека, автора статей по рыбной ловле, который использует таблицу для получения подробной информации для своих статей. Затем проведите стрелки от каждого столбца к его упоминанию в описании.

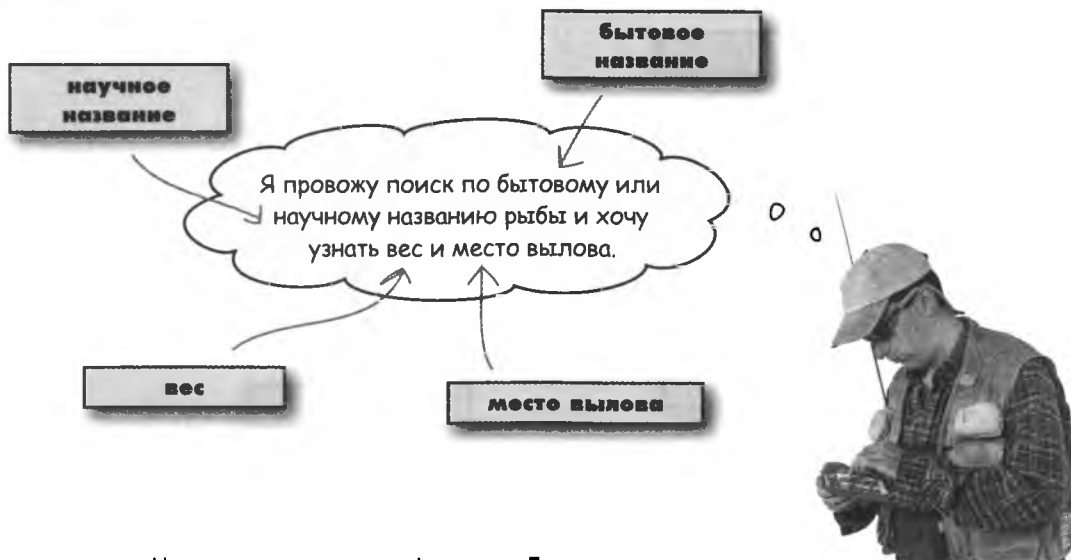
Фамилия Имя Дата

Штат

Вес Бытовое название Место вылова



Сможете ли вы определить столбцы таблицы по тем словам, которыми ихтиолог Марк описывает выборку данных из таблицы? Запишите имена столбцов в прямоугольниках.



Теперь ваша очередь. Напишите аналогичную фразу для Джека, автора статей по рыбной ловле, который использует таблицу для получения подробной информации для своих статей. Затем проведите стрелки от каждого столбца к его упоминанию в описании.



Но почему мы остановились на таблице Джека? Ведь дату можно разбить на день, месяц и год? Да и место вылова можно уточнить до названия улицы и номера дома рыбака.



Да, можно, но такой уровень детализации данных просто не нужен.

По крайней мере не в этом конкретном случае. Если бы Джек писал статьи о том, куда лучше отправиться на выходных, чтобы поймать большую рыбу, *тогда* он, возможно, указал бы название улицы и номер дома, чтобы читатели могли поискать жилье где-нибудь поблизости.

Но Джека интересует только место вылова и штат, и он добавил только эти столбцы, чтобы не увеличивать объем базы данных без необходимости. На этой стадии он решил, что его данные достаточно детализованы — то есть являются *атомарными*.



МОЗГОВОЙ ШТУРМ

Как вы думаете, что означает термин *атомарный* в контексте данных SQL?

Атомарные данные

Что такое «атом»? Маленький блок информации, который невозможно (или нежелательно) разделить на составные части меньшего размера. Это определение относится и к данным: АТОМАРНЫЕ данные были разделены на наименьшие компоненты, дальнейшее деление которых *невозможно или нежелательно*.

Доставка за 30 минут, или Пицца бесплатно

Для примера возьмем курьера, доставляющего пиццу клиентам. Чтобы выполнить свою работу, ему достаточно знать улицу и номер дома в одном столбце. Для него эти данные являются атомарными: курьеру никогда не приходится искать номер дома отдельно от названия улицы.

Более того, разбиение адреса доставки на название улицы и номер дома только усложнит его работу, а клиентам придется дольше дожидаться своих заказов.



Для курьера адрес доставки, объединяющий улицу и номер дома в одном столбце, достаточно атомарен.

```
File Edit Window Help SimplePizzaFactory
+-----+
| order_number | address
+-----+
| 246          | 59 N. Ajax Rapids
| 247          | 849 SQL Street
| 248          | 2348 E. PMP Plaza
| 249          | 1978 HTML Heights
| 250          | 24 S. Servlets Springs
| 251          | 807 Infinite Circle
| 252          | 32 Design Patterns Plaza
| 253          | 9208 S. Java Ranch
| 254          | 4653 W. EJB Estate
| 255          | 8678 OOA&D Orchard
+-----+
> SELECT address FROM pizza_deliveries WHERE order_num = 252;
+-----+
| address
+-----+
| 32 Design Patterns Plaza
+-----+
1 row in set (0.04 sec)
```


С другой стороны

А теперь возьмем агента по торговле недвижимостью. Вполне возможно, что ему понадобится отдельный столбец с номером дома — допустим, чтобы он мог получить список всех предложений по заданной улице. Для него название улицы и номер дома являются атомарными данными.

Для агента по торговле недвижимостью ситуация выглядит иначе. Отделение номера дома от названия улицы позволит легко получить список домов, продаваемых на заданной улице.



street_number	street_name	property_type	price
59	N. Ajax Rapids	condo	189000
849	SQL Street	apartment	109000
2348	E. PMP Plaza	house	355000
1978	HTML Heights	apartment	134000
24	S. Servlets Springs	house	355000
807	Infinite Circle	condo	143900
32	Design Patterns Plaza	house	465000
9208	S. Java Ranch	house	699000
4653	SQL Street	apartment	115000
8678	OOA&D Orchard	house	355000


```

> SELECT price, property_type FROM real_estate WHERE street_name = 'SQL Street';
+-----+-----+
| price | property_type |
+-----+-----+
| 109000.00 | apartment |
| 115000.00 | apartment |
+-----+-----+
2 rows in set (0.01 sec)

```

Атомарные данные и таблицы

Выбирая данные, которые будут храниться в ваших таблицах, задайте себе следующие вопросы.



1. Какой **один объект** описывает ваша таблица?

← Что описывает ваша таблица: клоунов, коров, пончики, людей?



2. Как вы предполагаете **ИСПОЛЬЗОВАТЬ** таблицу для получения информации о ее **объекте**?

← Спроектируйте таблицу так, чтобы запросы были простыми!



3. Содержат ли **СТОЛБЦЫ** таблицы **атомарные данные**, чтобы запросы были короткими и конкретными?

Часть Задаваемые Вопросы

В: Атомы совсем крошечные, верно? Значит ли это, что данные нужно разбить на *мельчайшие* фрагменты?

О: Нет. Атомарность данных подразумевает разбиение данных на наименьшие части, необходимые для создания эффективной таблицы, а не просто на самые мелкие части из всех возможных.

Не дробите данные сверх необходимости. Если лишние столбцы вам не нужны, не добавляйте их.

В: Как атомарность данных упростит мою работу?

О: Атомарность упрощает контроль за правильностью данных в таблице. Например, если в столбце хранятся номера домов, можно проследить за тем, чтобы в этом столбце хранились только числовые данные.

Кроме того, атомарность повышает эффективность запросов: запросы к атомарным данным быстрее пишутся и выполняются, что дает ощутимый эффект при хранении очень больших объемов данных.

Возьми в руку карандаш



Перед вами общепринятые правила определения атомарных данных. Для каждого правила приведите **два** гипотетических примера таблиц, нарушающих данное правило.

ПРАВИЛО 1. Столбец, содержащий атомарные данные, не может состоять из нескольких однотипных элементов.

Столбец interests таблицы Грега my_contacts нарушает это правило.

ПРАВИЛО 2. Таблица с атомарными данными не может содержать несколько однотипных столбцов.

Таблица easy_drinks нарушает это правило.

Возьми в руку карандаш
Решение



Перед вами общепринятые правила определения атомарных данных. Для каждого правила приведите **два** гипотетических примера таблиц, нарушающих данное правило.

ПРАВИЛО 1. Столбец, содержащий атомарные данные, не может состоять из нескольких однотипных элементов.

Конечно, ваши ответы будут другими, но вот вам пример:

food_name	ingredients
хлеб	мука, молоко, яйца, дрожжи, масло
салат	огурцы, помидоры, зелень

Помните таблицу Грега? В ней был столбец со списком увлечений, из-за которого поиск превращался в настоящий кошмар!

Здесь то же самое: представьте, что вам потребуется найти помидоры в списке ингредиентов.

ПРАВИЛО 2. Таблица с атомарными данными не может содержать несколько однотипных столбцов.

teacher	student1	student2	student3
Миссис Мартини	Джо	Рон	Келли
Мистер Говард	Санжяя	Тим	Джулия

Слишком много однотипных столбцов!



Упражнение

Теперь, когда вы знаете «официальные» правила атомарности и три этапа создания атомарных таблиц, взгляните на каждую таблицу, приведенную ранее в книге, и объясните, почему она является (или не является) атомарной.

Таблица Грега, с. 83

Таблица с оценками пончиков, с. 112

Таблица с клоунами, с. 155

Таблица с описаниями напитков, с. 93

Информация о рыбах, с. 194

О пользе нормализации

Ваша фирма по проектированию данных заработала, вы наняли новых проектировщиков баз данных SQL. И конечно, вам не хотелось бы целыми часами объяснять им, как работают ваши таблицы.

Если ваши таблицы будут НОРМАЛИЗОВАНЫ, то они будут соответствовать стандартным правилам, понятным для новых проектировщиков. К счастью, наши таблицы с атомарными данными уже наполовину нормализованы!

Атомарность данных — первый шаг на пути создания НОРМАЛИЗОВАННОЙ таблицы.



Упражнение

Теперь, когда вы знаете «официальные» правила атомарности и три этапа создания атомарных таблиц, взгляните на каждую таблицу, приведенную ранее в книге, и объясните, почему она является (или не является) атомарной.

Таблица Грега, с. 83 Не атомарна — столбцы «interest» и «seeking» нарушают правило 1.

Таблица с оценками пончиков, с. 112 Атомарна. В отличие от таблицы easy_drinks, в столбцах хранится разнотипная информация. И в отличие от столбца «activities» таблицы клоунов, в каждом столбце хранится только один информационный элемент.

Таблица с клоунами, с. 155 Не атомарна. В некоторых записях столбец «activities» содержит список из нескольких занятий, а это нарушает правило 1.

Таблица с описаниями напитков, с. 93 Не атомарна. Таблица содержит более одного столбца ингредиентов, а это нарушает правило 2.

Информация о рыбах, с. 194 Атомарна. В разных столбцах хранится разнотипная информация, и в каждом столбце содержится только один информационный элемент.

Преимущества нормализованных таблиц

1. Нормализованные таблицы не содержат дубликатов данных, а это сокращает размер базы данных.

Отсутствие дубликатов экономит дисковое пространство.

2. Уменьшение объема данных, по которым ведется поиск, ускоряет выполнение запросов.



Мои таблицы не так уж велики. Зачем мне тратить время на их нормализацию?



Потому что даже в небольших таблицах выигрыш суммируется.

К тому же объем данных увеличивается со временем. Если ваша таблица будет изначально нормализована, вам не придется изменять ее структуру позднее, когда окажется, что запросы выполняются слишком медленно.

Ненормализованные клоуны

Помните таблицу с информацией о клоунах? Сбор информации о клоунах неожиданно превратился в национальное увлечение, и старая таблица уже не справляется с потоком информации, потому что столбцы appearance и activities содержат *слишком* много данных. Для наших целей эта таблица не является атомарной.

Запросы с поиском к этим двум столбцам получаются очень сложными — столбцы содержат слишком много данных!

clown_info

name	last_seen	appearance	activities
Элси	Дом престарелых Черри Хилл	Ж, рыжие волосы, зеленый костюм, огромные ботинки	шарики, машинки
Пиклз	Вечеринка Джека Грина	М, оранжевые волосы, синий костюм, огромные ботинки	мим
Снаглз	Болмарт	Ж, желтая рубашка, синие штаны	рожок, зонтик
Мистер Хобо	Цирк VG	М, сигара, черные волосы, маленькая шляпа	скрипка
Кларабелл	Дом престарелых Бельмонт	Ж, розовые волосы, большой цветок, синее платье	кричалки, танцы
Скутер	Больница Окленд	М, синие волосы, красный костюм, большой нос	шарики
Зиппо	Торговый центр Милстоун	Ж, оранжевый костюм, штаны	танцы
Бэйб	Автошкола Эрла	Ж, розовый костюм с блестками	эквилибристика, машинки
Бонзо	Парк Диксон	М, женское платье в горошек	пение, танцы
Снифлз	Заведение Трэйси	М, зелено-фиолетовый костюм, длинный нос	разъезжает на машинке

Возьми в руку карандаш



Давайте сделаем таблицу более атомарной. Предположим, поиск должен вестись по столбцам appearance и activities, а также по столбцам last_seen. Запишите более правильную структуру столбцов.

Продолжить на с. 229.

На полпути к 1НФ

Даже когда таблица содержит атомарные данные, пройдена только половина пути. Полностью нормализованная таблица находится в ПЕРВОЙ НОРМАЛИЗОВАННОЙ ФОРМЕ, или сокращенно 1НФ.

Таблица, находящаяся в форме 1НФ, должна выполнять следующие два правила.

Мы уже знаем,
как это делается.

Каждая запись должна содержать атомарные значения.

Чтобы таблица
была полностью
нормализована,
каждой записи
необходимо при-
своить первич-
ный ключ.

Каждая запись должна обладать уникальным идентификатором, который называется первичным ключом.



Как вы думаете, какие столбцы хорошо подойдут на роль первичного ключа?



Правила первичных ключей

Столбец таблицы, который станет ее первичным ключом, назначается при создании таблицы. Через несколько страниц мы создадим таблицу и назначим первичный ключ, но сначала давайте повнимательнее разберемся с тем, какими свойствами должен обладать первичный ключ.



Первичный ключ используется для однозначной идентификации записей.

Это значит, что данные в столбце первичного ключа не могут повторяться. Для примера возьмем следующую таблицу; как вы думаете, какие из ее столбцов хорошо подойдут на роль первичного ключа?

SSN (номер социального страхования)	last_name (фамилия)	first_name (имя)	phone_number (телефон)
-------------------------------------	---------------------	------------------	------------------------

↑
Каждому человеку назначается уникальный номер социального страхования; этот столбец может стать первичным ключом.

↖ ↗ ↘
В этих трех столбцах с высокой вероятностью будут встречаться повторяющиеся значения — например, в базе данных могут быть записи нескольких людей по имени Джон, а несколько людей, живущих вместе, могут иметь одинаковые телефоны. Вероятно, это не лучшие кандидаты на роль первичного ключа.



Будьте осторожны!

Будьте осторожны при использовании номеров социального страхования в базах данных.

Количество краж личных данных только увеличивается, и люди неохотно сообщают свои коды социального страхования — и вполне обоснованно. Эти данные слишком важны, чтобы рисковать ими. Можете ли вы гарантировать, что ваша база данных защищена на 100%? Если нет — номера социального страхования могут быть похищены вместе с личными данными ваших клиентов.



Первичный ключ — столбец таблицы, имеющий уникальное значение для каждой записи.



Первичный ключ не может содержать NULL

Значение NULL не может быть уникальным, потому что в других записях этот столбец тоже может содержать NULL.



Значение первичного ключа должно задаваться при вставке записи

При вставке в таблицу записи без указания значения первичного ключа возникает риск создания записи с первичным ключом NULL и появления дубликатов, а это нарушает требования первой нормальной формы.



Первичный ключ должен быть компактным

Первичный ключ должен содержать только ту информацию, которая обеспечивает его уникальность, и ничего более.



Значения первичного ключа должны оставаться неизменными

Если бы первичный ключ можно было изменять, то ему можно было бы случайно присвоить уже используемое значение. Помните, что первичный ключ должен быть уникальным.

 **МОЗГОВОЙ ШТУРМ**

Сможете ли вы предложить хороший первичный ключ с учетом всех этих правил?

Еще раз просмотрите таблицы, встречавшиеся нам в книге. Есть ли в какой-либо из них столбец, содержащий уникальные значения?

Погодите, если я не могу использовать номер социального страхования, но при этом первичный ключ должен быть компактным, отличным от NULL и неизменным — то что же использовать?



Лучшим первичным ключом может быть новый первичный ключ.

В том, что касается первичных ключей, лучшим решением часто оказывается создание столбца, содержащего уникальный номер. Представьте таблицу, которая содержит все прежние данные, к которым добавляется новый числовой столбец. В следующем примере он будет называться ID (идентификатор).

Если бы не столбец ID, две записи Джона Брауна были бы одинаковыми, но в данном случае речь идет о двух разных людях. Столбец ID обеспечивает уникальность этих записей. Таблица находится в первой нормальной форме.

id	last_name	first_name	nick_name
1	Браун	Джон	Джон
2	Элсуорт	Ким	Ким
3	Браун	Джон	Джон
4	Петрильо	Мария	Мария
5	Франкен	Эсме	Эм

← Запись Джона Брауна.

← Тоже запись Джона Брауна, но столбец ID показывает, что эта уникальная запись относится к другому Джону Брауну.



Для любознательных

В мире SQL идут ожесточенные споры по поводу использования синтетических (то есть искусственно созданных, как столбец ID в этом примере) и естественных ключей — данных, уже хранящихся в таблице (номер машины, номер социального страхования и т. д.). Мы не будем становиться на ту или иную сторону; в главе 7 первичные ключи будут рассмотрены более подробно.

Часть
**Задаваемые
 Вопросы**

В: Вы упоминаете о «первой» нормальной форме. Значит, есть и вторая? И третья?

О: Да, вторая и третья нормальные формы действительно существуют; они определяются более жесткими правилами. Вторая и третья нормальные формы рассматриваются в главе 7.

В: Мы изменили свои таблицы, чтобы в них хранились атомарные значения. Какая-нибудь из этих таблиц находится в 1НФ?

О: Нет. До настоящего момента ни одна из созданных нами таблиц не имела первичного ключа с уникальными значениями.

В: Столбец `comments` в таблице с описаниями пончиков мне не кажется атомарным. Другими словами, я не вижу, как удобно провести поиск по этому столбцу.

О: Совершенно верно. Поле не особенно атомарно, но структура нашей таблицы этого и не требует. Если бы мы захотели ограничить комментарии заранее определенным набором слов, то поле могло бы стать атомарным. С другой стороны, тогда поле не содержало бы искренние комментарии в произвольной форме.

Как прийти в НОРМУ

Пришло время отступить на шаг и нормализовать наши таблицы. Для этого необходимо сделать данные атомарными и назначить первичные ключи. Создание первичного ключа — один из стандартных этапов написания кода команды `CREATE TABLE`.



А вы помните, как добавить столбец в существующую таблицу?

Исправление таблицы Грега

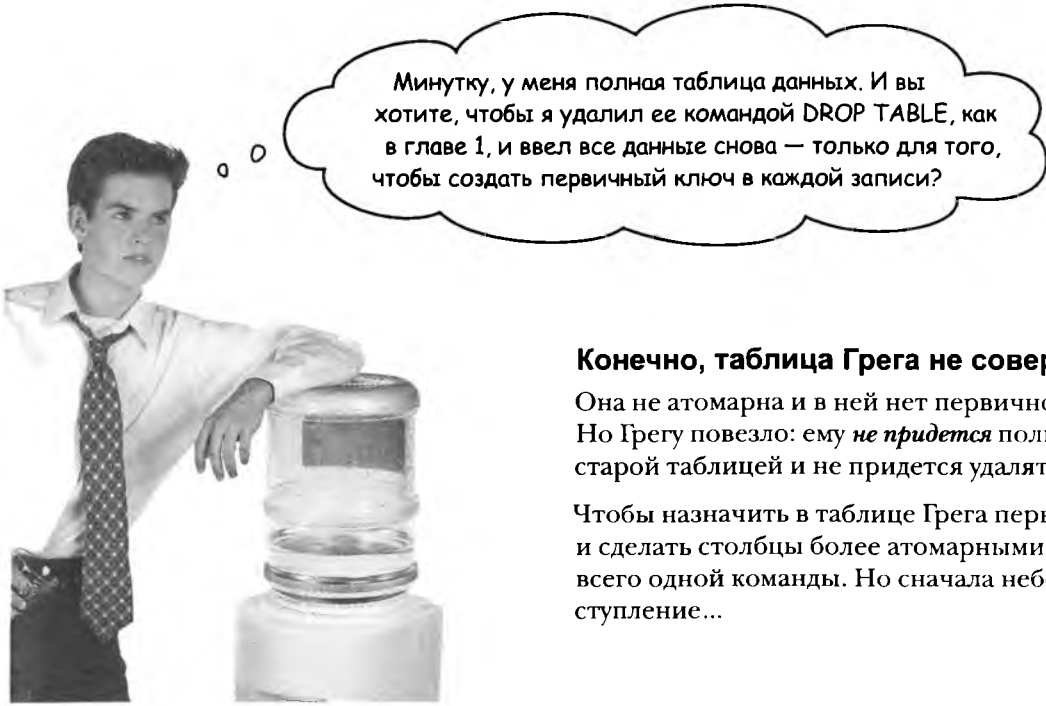
После всего сказанного становится ясно, что необходимо сделать для исправления таблицы Грега.

Исправление таблицы Грега, шаг 1. Выполнить выборку всех данных командой SELECT и как-то сохранить их.

Исправление таблицы Грега, шаг 2. Создать новую нормализованную таблицу.

Исправление таблицы Грега, шаг 3. Вставить все старые данные в новую таблицу, изменяя каждую запись в соответствии с новой структурой таблицы.

Теперь старую таблицу можно удалить.



Минутку, у меня полная таблица данных. И вы хотите, чтобы я удалил ее командой DROP TABLE, как в главе 1, и ввел все данные снова — только для того, чтобы создать первичный ключ в каждой записи?

Конечно, таблица Грега не совершенна.

Она не атомарна и в ней нет первичного ключа. Но Грегу повезло: ему *не придется* пользоваться старой таблицей и не придется удалять данные.

Чтобы назначить в таблице Грега первичный ключ и сделать столбцы более атомарными, достаточно всего одной команды. Но сначала небольшое отступление...

Старая команда CREATE TABLE

Таблице Грега нужен первичный ключ. После всех разговоров об атомарности данных Грег понимает, что он может принять меры для того, чтобы сделать столбцы своей таблицы более атомарными. Но прежде чем разбираться с тем, как исправить существующую таблицу, давайте вспомним, как она создавалась!

Вот как выглядела команда создания таблицы из главы 1.

```
CREATE TABLE my_contacts
```

```
(
  last_name VARCHAR(30),
  first_name VARCHAR(20),
  email VARCHAR(50),
  gender CHAR(1),
  birthday DATE,
  profession VARCHAR(50),
  location VARCHAR(50),
  status VARCHAR(20),
  interests VARCHAR(100),
  seeking VARCHAR(100)
);
```

Нет подходящего кандидата на роль первичного ключа.

Нельзя ли сделать эти столбцы более атомарными при создании таблицы?



А если старая команда CREATE TABLE нигде не была записана? Как получить доступ к коду создания таблицы?

Сначала покажи ~~деньги~~ ^{таблицу}

Может, для просмотра кода создания таблицы воспользоваться командой DESCRIBE my_contacts? Результат ее выполнения будет выглядеть примерно так:

```
File Edit Window Help GregsListAgain
+-----+-----+-----+-----+-----+-----+
| Column | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| last_name | varchar(30) | YES | | NULL | |
| first_name | varchar(20) | YES | | NULL | |
| email | varchar(50) | YES | | NULL | |
| gender | char(1) | YES | | NULL | |
| birthday | date | YES | | NULL | |
| profession | varchar(50) | YES | | NULL | |
| location | varchar(50) | YES | | NULL | |
| status | varchar(20) | YES | | NULL | |
| interests | varchar(100) | YES | | NULL | |
| seeking | varchar(100) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
```

Но нас интересует код **CREATE**, а не описания полей таблицы. И нам хотелось бы узнать, как должна выглядеть исходная команда, не вводя ее заново.

Команда `SHOW CREATE TABLE` возвращает команду `CREATE TABLE`, которая была использована для создания таблицы (до занесения в таблицу первых данных). Попробуйте ввести следующую команду:

```
SHOW CREATE TABLE my_contacts;
```


Команда для экономии Времени

Взгляните на код, который использовался для создания таблицы на с. 217, и приведенный ниже результат выполнения команды SHOW CREATE TABLE my_contacts. Эти фрагменты не идентичны, но если вставить этот код в команду CREATE TABLE, результат будет тем же. Удалять обратные апострофы или параметры данных не нужно, но если вы это сделаете, команда получится более компактной.

Имена столбцов и таблицы заключены в обратные апострофы. Эти символы присутствуют в результатах команды SHOW CREATE TABLE.

```
CREATE TABLE `my_contacts`
(
  `last_name` varchar(30) default NULL,
  `first_name` varchar(20) default NULL,
  `email` varchar(50) default NULL,
  `gender` char(1) default NULL,
  `birthday` date default NULL,
  `profession` varchar(50) default NULL,
  `location` varchar(50) default NULL,
  `status` varchar(20) default NULL,
  `interests` varchar(100) default NULL,
  `seeking` varchar(100) default NULL,
) ENGINE=MyISAM DEFAULT CHARSET=cp1251
```

SQL считает, что столбцы по умолчанию инициализируются значением NULL (если явно не задано другое значение).

При создании таблицы желательно указывать, может ли столбец содержать NULL.

Не обращайте внимания на текст после закрывающей круглой скобки. Он описывает механизм хранения данных и используемую кодировку символов. Пока нас устраивают значения по умолчанию.

Если исходная таблица не была удалена, то этой таблице придется присвоить новое имя.

Если скопировать и выполнить этот код, он создаст таблицу.

Команда CREATE TABLE с назначением первичного ключа

Перед вами код, полученный при выполнении команды SHOW CREATE TABLE. Мы удалили из него обратные апострофы и последнюю строку. В начало списка столбцов был добавлен столбец contact_id с условием NOT NULL, а в конце списка появилось условие PRIMARY KEY, в котором новый столбец contact_id назначается первичным ключом.

Помните, что столбец первичного ключа не может содержать NULL! Присутствие NULL в столбце первичного ключа не позволит однозначно идентифицировать каждую запись в таблице.

Мы создали новый столбец contact_id, который станет первичным ключом таблицы. Хранящиеся в нем целые числа уникальны для каждой записи, а таблица становится атомарной.

```
CREATE TABLE my_contacts  
(  
  contact_id INT NOT NULL,  
  last_name varchar(30) default NULL,  
  first_name varchar(20) default NULL,  
  email varchar(50) default NULL,  
  gender char(1) default NULL,  
  birthday date default NULL,  
  profession varchar(50) default NULL,  
  location varchar(50) default NULL,  
  status varchar(20) default NULL,  
  interests varchar(100) default NULL,  
  seeking varchar(100) default NULL,  
  PRIMARY KEY (contact_id)  
)
```

Здесь назначается первичный ключ таблицы. Синтаксис прост: за ключевыми словами PRIMARY KEY в круглых скобках указывается имя столбца, который будет первичным ключом — в нашем примере это новый столбец contact_id.

Часть Задаваемые Вопросы

В: Вы говорите, что первичный ключ не может содержать NULL. Что еще предотвращает появление в нем дубликатов?

О: Прежде всего вы сами. При вставке значений в таблицу столбцу `contact_id` присваиваются уникальные значения. Например, в первой команде `INSERT` столбцу `contact_id` присваивается значение 1, во второй — значение 2 и т. д.

В: Присваивать новое значение столбцу **PRIMARY KEY** при каждой вставке новой записи весьма хлопотно. Нет ли более простого способа?

О: Есть два таких способа. Первый — использование в качестве первичного ключа заведомо уникального столбца таблицы. Мы уже упоминали о том, что этот способ может создать проблемы (как, например, при использовании номеров социального страхования).

Второй, более простой способ заключается в создании нового столбца с уникальными идентификаторами — как, например, `contact_id` на предыдущей странице. Вы можете приказать своей РСУБД автоматически генерировать его значения при помощи специальных ключевых слов (подробности на следующей странице).

В: Для чего еще можно использовать `SHOW`, кроме вывода команды `CREATE`?

О: Команда `SHOW` может использоваться для вывода информации о столбцах таблицы:

```
SHOW COLUMNS FROM tablename;
```

Команда выводит описания всех столбцов таблицы с типами данных, а также другими сведениями, относящимся к конкретным столбцам.

```
SHOW CREATE DATABASE databasename;
```

По аналогии с командой `SHOW CREATE <таблица>`, эта команда выводит код команды создания базы данных.

```
SHOW INDEX FROM tablename;
```

Команда выводит информацию об индексируемых

столбцах и типах индексов. До настоящего момента из индексов нам встречались только первичные ключи, но скоро вы лучше поймете смысл этой команды.

И еще одна **ОЧЕНЬ** полезная команда:

```
SHOW WARNINGS;
```

Если на консоли выводится сообщение о том, что выполнение команды SQL привело к выдаче предупреждений, то для просмотра предупреждений используется команда `SHOW WARNINGS`.

Существуют и другие разновидности команды `SHOW`. Мы рассмотрели лишь те, которые имеют прямое отношение к интересующим нас темам.

В: Для чего нужны обратные апострофы в результатах `SHOW CREATE TABLE`? Вы уверены, что без них можно обойтись?

О: РСУБД в некоторых ситуациях не может определить, что имя столбца действительно является именем столбца. Например, если имена столбцов будут заключаться в обратные апострофы, вы сможете использовать в качестве имен зарезервированные ключевые слова SQL (хотя это крайне неудачная мысль).

Допустим, по каким-то непостижимым причинам вы хотите включить в таблицу столбец с именем `select`. Такое объявление столбца недопустимо:

```
select varchar(50)
```

А такое объявление `сработает`:

```
`select` varchar(50)
```

В: А почему ключевые слова нельзя использовать в именах столбцов?

О: Можно, но нежелательно. Только представьте, какими запутанными станут ваши запросы и сколько хлопот будет с вводом обратных апострофов, когда можно обойтись без них. Кроме того, `select` — неудачное имя столбца: оно ничего не сообщает о данных, которые в нем хранятся.

1, 2, 3 и так далее

Если снабдить столбец `contact_id` ключевым словом `AUTO_INCREMENT`, то РСУБД будет автоматически заполнять его значениями: 1 для записи 1, 2 для записи 2 и т. д.

```
CREATE TABLE my_contacts
(
  contact_id INT NOT NULL AUTO_INCREMENT,
  last_name varchar(30) default NULL,
  first_name varchar(20) default NULL,
  email varchar(50) default NULL,
  gender char(1) default NULL,
  birthday date default NULL,
  profession varchar(50) default NULL,
  location varchar(50) default NULL,
  status varchar(20) default NULL,
  interests varchar(100) default NULL,
  seeking varchar(100) default NULL,
  PRIMARY KEY (contact_id)
)
```

Вот оно: в большинстве реализаций SQL просто добавьте ключевое слово `AUTO_INCREMENT` (Пользователи MS SQL указывают ключевое слово `INDEX` с начальным значением и приращением. За конкретной информацией обращайтесь к справочному руководству по MS SQL).

У первой записи в этом столбце сохраняется значение 1. Затем значение столбца автоматически увеличивается на 1 при каждой вставке новой записи.



Пока все достаточно просто. Но как должна выглядеть команда `INSERT`, если этот столбец заполняется автоматически? Могу ли я случайно присвоить ему другое значение?

Как вы думаете, что произойдет?

А еще лучше – попробуйте и посмотрите сами.



Упражнение

1. Напишите команду CREATE TABLE для создания приведенной ниже таблицы, в которой хранятся имена и фамилии. Таблица должна содержать столбец первичного ключа с ключевым AUTO_INCREMENT и два атомарных столбца.

2. Откройте терминал SQL или графический интерфейс, выполните команду CREATE TABLE.

3. Попробуйте выполнить каждую из приведенных ниже команд INSERT. Обведите кружком команды, которые были успешно выполнены.

```
INSERT INTO your_table (id, first_name, last_name)
VALUES (NULL, 'Марсия', 'Брэди');
```

```
INSERT INTO your_table (id, first_name, last_name)
VALUES (1, 'Джен', 'Брэди');
```

```
INSERT INTO your_table
VALUES ('', 'Бобби', 'Брэди');
```

```
INSERT INTO your_table (first_name, last_name)
VALUES ('Синди', 'Брэди');
```

```
INSERT INTO your_table (id, first_name, last_name)
VALUES (99, 'Питер', 'Брэди');
```

4. Все ли команды были выполнены успешно? Напишите, как будет выглядеть содержимое таблицы после выполнения команд INSERT.

your_table

id	first_name	last_name



Упражнение
Решение

1. Напишите команду CREATE TABLE для создания приведенной ниже таблицы, в которой хранятся имена и фамилии. Таблица должна содержать столбец первичного ключа с ключевым AUTO_INCREMENT и два атомарных столбца.

```
CREATE TABLE your_table
(
  id INT NOT NULL AUTO_INCREMENT,
  first_name VARCHAR(20),
  last_name VARCHAR(30),
  PRIMARY KEY (id)
);
```

2. Откройте терминал SQL или графический интерфейс, выполните команду CREATE TABLE.

3. Попробуйте выполнить каждую из приведенных ниже команд INSERT. Обведите кружком команды, которые были успешно выполнены.

INSERT INTO your_table (id, first_name, last_name)
VALUES (NULL, 'Марсия', 'Брэди');

INSERT INTO your_table (id, first_name, last_name)
VALUES (1, 'Джен', 'Брэди');

INSERT INTO your_table
VALUES ('', 'Бобби', 'Брэди');

INSERT INTO your_table (first_name, last_name)
VALUES ('Синди', 'Брэди');

INSERT INTO your_table (id, first_name, last_name)
VALUES (99, 'Питер', 'Брэди');

Последняя команда «работает», но заменяет значение столбца AUTO_INCREMENT.

4. Все ли команды были выполнены успешно? Напишите, как будет выглядеть содержимое таблицы после выполнения команд INSERT.

your_table

id	first_name	last_name
1	Марсия	Брэди
2	Бобби	Брэди
3	Синди	Брэди
99	Питер	Брэди

Похоже, мы потеряли запись Джен, когда попытались назначить ей индекс, уже назначенный Марсии!

Часто
**Задаваемые
 Вопросы**

В: Почему первый запрос (с NULL в столбце id) вставляет запись, хотя для id установлено ограничение NOT NULL?

О: Хотя на первый взгляд команда выполняться не должна, с AUTO_INCREMENT значение NULL просто игнорируется. С другой стороны, без AUTO_INCREMENT вы получите сообщение об ошибке, а запись вставлена не будет. Убедитесь в этом сами.

Знаете, это не обнадеживает. Конечно, я могу скопировать код из результатов SHOW CREATE TABLE, но похоже, мне придется удалять таблицу и вводить все данные заново только для того, чтобы добавить первичный ключ.



Вводить данные заново не придется; вместо этого можно воспользоваться командой ALTER.

Таблицу с данными не обязательно удалять, а затем создавать заново. Структуру существующих таблиц можно изменить. Но для этого нам потребуется команда ALTER и некоторые ключевые слова, описанные в главе 5.

Добавление первичного ключа в существующую таблицу

Перед вами код добавления первичного ключа AUTO_INCREMENT в таблицу my_contacts. (Команда получается довольно длинной, так что книгу придется развернуть.)

FIRST приказывает ПСУБД поставить новый столбец на первое место в списке. Строго говоря, это не обязательно, но нахождение первичного ключа в начале списка считается «хорошим стилем».

Код добавления нового столбца в таблицу. Выглядит знакомо, не правда ли?

Новая команда SQL: ALTER.

ALTER TABLE my_contacts

ADD COLUMN contact_id INT NOT NULL AUTO_INCREMENT FIRST,

ADD PRIMARY KEY (contact_id);

Вероятно, вы узнали строку, в которой назначается первичный ключ.

Ключевые слова ADD COLUMN сообщают, что в таблицу добавляется новый столбец с именем contact_id.



Как вы думаете, создаст ли эта команда значения нового столбца contact_id для записей, уже находящихся в таблице, или же они будут создаваться только для вновь вставляемых записей? Как это проверить?

ALTER TABLE и добавление первичного ключа

Проверьте, как работает этот код. Откройте терминал SQL, выполните команду USE для базы данных gregs_list и введите следующую команду:

Сообщает, что столбец был добавлен в 50 записях, уже хранящихся в нашей таблице. У вас их будет меньше.

```
File Edit Window Help Alterations
> ALTER TABLE my_contacts
  -> ADD COLUMN contact_id INT NOT NULL AUTO_INCREMENT FIRST,
  -> ADD PRIMARY KEY (contact_id);
Query OK, 50 rows affected (0.04 sec)
Records: 50 Duplicates: 0 Warnings: 0
```

Здорово! У меня появился первичный ключ, заполненный данными. Может ли команда ALTER TABLE добавить столбец с номером телефона?

Чтобы увидеть, что произошло с таблицей, выполните команду **SELECT * from my_contacts;**

Столбец contact_id включен в таблицу первым, до всех остальных столбцов.

```
File Edit Window Help Alterations
```

contact_id	last_name	first_name	email
1	Андерсон	Джиллиан	jill_anderson@yahoo.com
2	Иоффе	Кевин	kj@simuduck.com
3	Ньюсам	Аманда	aman2luv@yahoo.com
4	Гарсиа	Эд	ed99@mysoftware.com
5	Раундтри	Джо-Энн	jojo@yahoo.com
6	Бриггс	Хрис	cbriggs@mail.com

Так как мы использовали AUTO_INCREMENT, столбец автоматически заполняется значениями при обновлении записей таблицы.

При следующей вставке новой записи столбцу contact_id будет присвоено значение, на 1 большее максимального значения contact_id в таблице. Если у последней записи столбец contact_id содержит значение 23, то у следующей записи он будет равен 24.

Напомним, что это еще не конец таблицы; у Грега много знакомых.

Получит ли Грег свой столбец с номером телефона? Об этом вы узнаете в главе 5.



Новые инструменты

Вы взяли на вооружение материал главы 4. Только посмотрите, сколько у вас появилось новых инструментов! Полный список инструментов приведен в приложении III.

Правило атомарности данных 1:
Столбец, содержащий атомарные данные, не может состоять из нескольких одно-типных элементов.

Правило атомарности данных 2:
Таблица с атомарными данными не может содержать несколько однотипных столбцов.

Первичный ключ

Столбец или набор столбцов, значение которого однозначно идентифицирует запись в таблице.

AUTO_INCREMENT

Для столбца, объявленного с этим ключевым словом, при каждом выполнении команды **INSERT** автоматически генерируется уникальное целое значение.

Атомарные данные

Данные столбца называются атомарными, если они разбиты на наименьшие фрагменты, подходящие для ваших целей.

SHOW CREATE TABLE

Команда выводит правильный синтаксис создания существующей таблицы.

Первая нормальная форма (1НФ)

Каждая запись должна содержать атомарные значения, и каждая запись должна обладать уникальным идентификатором.

Возьми в руку карандаш



Решение

Давайте сделаем таблицу клоунов более атомарной. Предположим, поиск должен вестись по столбцам appearance и activities, а также по столбцам last_seen. Запишите более правильную структуру столбцов.

Здесь нет единственного правильного ответа.

Фактически лучшее, что можно сделать — это выделить в отдельные столбцы такие атрибуты, как пол, цвет костюма, цвет штанов, тип шляпы, музыкальный инструмент, шарики (да/нет), пение (да/нет)? танцы (да/нет) и т. д.

Чтобы таблица была атомарной, разные виды действий и элементы внешнего вида нужно разделить по разным столбцам.

Прибавьте дополнительные баллы, если вы решили разделить столбец местонахождения на штат, город и улицу!

5 ALTER

* Как изменить прошлое *

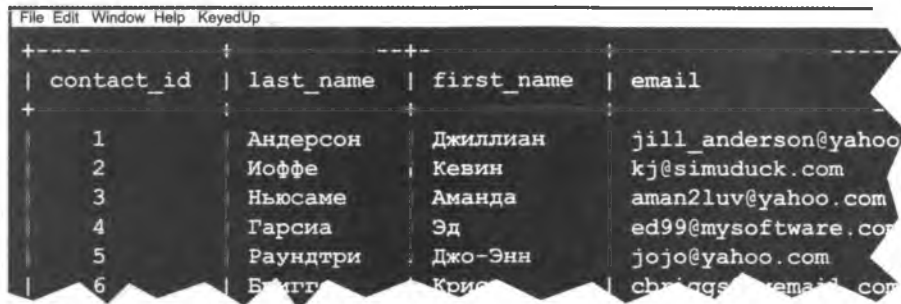
Если бы я мог вернуться в прошлое, я бы выбрал другой шампунь.



Вам никогда не хотелось исправить прошлые ошибки? Что же, теперь у вас есть такая возможность. Благодаря команде **ALTER** вы сможете применить свои новые знания к таблицам, созданным много дней, месяцев и даже лет назад. А самое замечательное, что это никак не повредит существующим данным! К настоящему моменту вы уже знаете, что означает понятие **нормализованный**, и можете применять его ко всем таблицам — как прошлым, так и будущим.

Нужно Внести пару изменений

Грег хочет внести в таблицу некоторые изменения, но так, чтобы не потерять существующие данные.



```
File Edit Window Help KeyedUp
+-----+-----+-----+-----+
| contact_id | last_name | first_name | email |
+-----+-----+-----+-----+
| 1 | Андерсон | Джиллиан | jill_anderson@yahoo.com |
| 2 | Иоффе | Кевин | kj@simuduck.com |
| 3 | Ньюсаме | Аманда | aman2luv@yahoo.com |
| 4 | Гарсиа | Эд | ed99@mysoftware.com |
| 5 | Раундтри | Джо-Энн | jojo@yahoo.com |
| 6 | Блэтти | Кристин | chriss@remail.com |
```



Да, он легко добавляется командой ALTER TABLE.

Более того, нам кажется, что вам стоит сделать это самостоятельно, потому что вы уже знакомы с командой ALTER. Чтобы узнать, как выглядит код команды, выполните следующее упражнение!

Возьми в руку карандаш



Взгляните повнимательнее на команду ALTER TABLE, использованную для добавления первичного ключа в главе 4. Удастся ли вам составить свою команду ALTER TABLE для добавления столбца с номером телефона из 10 цифр? Учтите, что в новой команде не обязательно использовать все ключевые слова из приведенного примера.

```
ALTER TABLE my_contacts  
ADD COLUMN contact_id INT NOT NULL AUTO_INCREMENT FIRST,  
ADD PRIMARY KEY (contact_id);
```

Запишите свою команду ALTER TABLE:

Вы также можете задать местонахождение нового столбца в таблице при помощи ключевого слова AFTER. Попробуйте определить, где должно находиться это ключевое слово, чтобы новый столбец был добавлен после столбца `first_name`.

Запишите новую версию своей команды ALTER TABLE:



Возьми в руку карандаш

Решение

Взгляните повнимательнее на команду ALTER TABLE, использованную для добавления первичного ключа в главе 4. Удастся ли вам составить свою команду ALTER TABLE для добавления столбца с номером телефона из 10 цифр? Учтите, что в новой команде не обязательно использовать все ключевые слова из приведенного примера.

```
ALTER TABLE my_contacts
ADD COLUMN contact_id INT NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (contact_id);
```

Ключевые слова NOT NULL, AUTO_INCREMENT и FIRST в новой команде не используются.

Запишите свою команду ALTER TABLE:

Изменения, как и прежде, вносятся в таблицу my_contacts.

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10);
```

Этот фрагмент сообщает команде ALTER, какие изменения вносятся в таблицу.

Имя нового столбца.

Будем считать, что длина телефонного номера не превышает 10 цифр. Грег не подумал о телефонах из других стран.

Вы также можете задать местонахождение нового столбца в таблице при помощи ключевого слова AFTER. Попробуйте определить, где должно находиться это ключевое слово, чтобы новый столбец был добавлен после столбца first_name.

Запишите новую версию своей команды ALTER TABLE:

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
AFTER first_name;
```

Ключевое слово AFTER с именем столбца, за которым должен следовать новый столбец. Таким образом, столбец phone в таблице следует за столбцом first_name.

Условие AFTER не обязательно. Если оно отсутствует, то столбец добавляется в конец таблицы.

Наряду с ключевыми словами **FIRST** и **AFTER** при вставке столбцов могут использоваться ключевые слова **BEFORE** и **LAST**.

А также **SECOND**, **THIRD** и так далее.

За Сценой 



Магниты с кодами

Измените позицию добавляемого столбца `phone` при помощи магнитов с ключевыми словами. Создайте как можно больше разных команд, запишите состояние столбцов после их выполнения. Конструкция **BEFORE** в MySQL не работает; довольствуйтесь **FIRST** и **AFTER**.

phone	contact_id	last_name	first_name	email
-------	------------	-----------	------------	-------

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
```

contact_id	last_name	first_name	email	phone
------------	-----------	------------	-------	-------

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
```

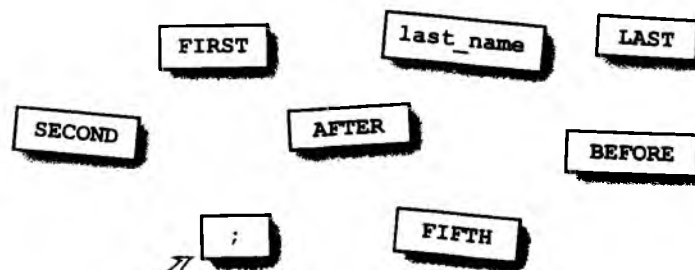
contact_id	phone	last_name	first_name	email
------------	-------	-----------	------------	-------

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
```

contact_id	last_name	phone	first_name	email
------------	-----------	-------	------------	-------

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
```

↑
Добавьте свои магниты в конец команды.



Используйте символ «;» везде, где потребуется.



Магниты с кодами

Измените позицию добавляемого столбца phone при помощи магнитов с ключевыми словами. Создайте как можно больше разных команд, запишите состояние столбцов после их выполнения. Конструкция BEFORE в MySQL не работает; довольствуйтесь FIRST и AFTER.

ALTER TABLE my_contacts

ADD COLUMN phone VARCHAR(10)



С ключевым словом FIRST столбец phone размещается перед всеми остальными столбцами.

phone	contact_id	last_name	first_name	email
-------	------------	-----------	------------	-------

ALTER TABLE my_contacts

ADD COLUMN phone VARCHAR(10)



ALTER TABLE my_contacts

ADD COLUMN phone VARCHAR(10)



ALTER TABLE my_contacts

ADD COLUMN phone VARCHAR(10)



С ключевым словом LAST столбец phone размещается после всех остальных столбцов. То же происходит при использовании ключевого слова FIFTH и без указания позиции.

contact_id	last_name	first_name	email	phone
------------	-----------	------------	-------	-------

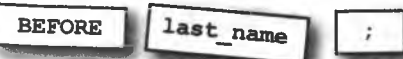
ALTER TABLE my_contacts

ADD COLUMN phone VARCHAR(10)



ALTER TABLE my_contacts

ADD COLUMN phone VARCHAR(10)

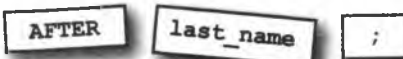


С ключевым словом SECOND столбец phone размещается во второй позиции; то же происходит при использовании ключевого слова BEFORE (с именем столбца last_name).

contact_id	phone	last_name	first_name	email
------------	-------	-----------	------------	-------

ALTER TABLE my_contacts

ADD COLUMN phone VARCHAR(10)



Конструкция AFTER last_name размещает столбец phone в третьей позиции. Если бы в наборе был магнит THIRD, то он сделал бы то же самое.

contact_id	last_name	phone	first_name	email
------------	-----------	-------	------------	-------

Изменение таблиц

Команда ALTER позволяет изменить практически любые атрибуты таблицы без необходимости повторной вставки данных. Будьте осторожны: изменение типа данных столбца может привести к потере данных.

Dataville Alterations

Обслуживание существующих таблиц:

CHANGE – изменение имени и типа данных столбцов *

MODIFY – изменение типа данных или позиции столбцов *

ADD – добавление столбцов в таблицу (тип данных по выбору заказчика)

DROP – удаление столбцов из таблицы *

* Возможна потеря данных, гарантия не предоставляется.

Одно маленькое изменение, это совсем не больно.



ДОПОЛНИТЕЛЬНЫЕ УСЛУГИ

Изменение порядка столбцов
(только при использовании ADD)



Какие изменения могут потребоваться в этой таблице?

projekts		
number	descriptionofproj	contractorejob
1	покраска дома	Мэрфи
2	перестройка кухни	Вальдес
3	укладка паркета	Келлер
4	кровельные работы	Джексон

Капитальный ремонт таблицы

Начнем с таблицы, которая явно нуждается в серьезной переработке.

Служба ремонта таблиц к вашим услугам! В наших руках даже негодная таблица преобразится и станет настоящей гордостью вашей базы данных!

Не содержит информации о том, какие данные должны храниться в этой таблице.

Название столбца ничего не говорит о его содержимом.

С парой символов подчеркивания имя столбца станет более понятным.

projekts

number	descriptionofproj	contractoronjob
1	покраска дома	Мэрфи
2	перестройка кухни	Вальдес
3	укладка паркета	Келлер
4	кровельные работы	Джексон



Имена таблицы и столбцов оставляют желать лучшего, но с данными все в порядке. Нам хотелось бы их сохранить.

Давайте воспользуемся командой DESCRIBE и посмотрим, какую структуру имеет таблица. Из ее результатов мы узнаем, есть ли у таблицы первичный ключ и данные какого типа хранятся в каждом из столбцов.

```
File Edit Window Help BadTableDesign
--> DESCRIBE projekts;
+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| number         | int(11)       | YES  |     | NULL    |      |
| descriptionofproj | varchar(50)   | YES  |     | NULL    |      |
| contractoronjob  | varchar(10)   | YES  |     | NULL    |      |
+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

Переименование таблицы

У таблицы в ее текущем состоянии имеются свои недостатки, но с помощью команды ALTER мы сделаем ее пригодной для хранения списка работ по ремонту дома. Начнем с присваивания таблице осмысленного имени командой ALTER TABLE.

«projekts» — старое имя таблицы.

```
ALTER TABLE projekts
RENAME TO project_list;
```

Таблицу нужно ПЕРЕИМЕНОВАТЬ!

«project_list» — новое имя, которое мы присваиваем таблице.



Упражнение

Что еще следует сделать для улучшения таблицы? Найдите в описании ключевые пункты предполагаемого использования таблицы, после чего заполните названия столбцов.

proj_id

Чтобы таблица стала НОРМАЛИЗОВАННОЙ, мы добавим в нее первичный ключ с уникальным идентификатором проекта. Также нам понадобятся столбцы для описания каждого проекта, даты начала работ, примерной стоимости, названия компании-подрядчика и номера телефона.



Упражнение
Решение

Что еще следует сделать для улучшения таблицы? Найдите в описании ключевые пункты предполагаемого использования таблицы, после чего заполните названия столбцов.

Не огорчайтесь, если ваши имена столбцов не совпадают с приведенными. Сокращения допустимы, если они не искажают смысла хранимых данных.

Убедитесь в том, что короткие имена (такие, как `proj_id`) будут понятны вам и другим пользователям базы данных.



Грандиозные планы

project_list

number	descriptionofproj	contractoronjob
1	покраска дома	Мэрфи
2	перестройка кухни	Вальдес
3	укладка паркета	Келлер
4	кровельные работы	Джексон

Данные трех новых столбцов уже хранятся в таблице. Вместо того чтобы создавать новые столбцы, мы переименуем уже существующие столбцы командой RENAME. Это позволит нам избежать повторной вставки данных в новые столбцы.



Какой из существующих столбцов станет хорошим кандидатом на роль первичного ключа?

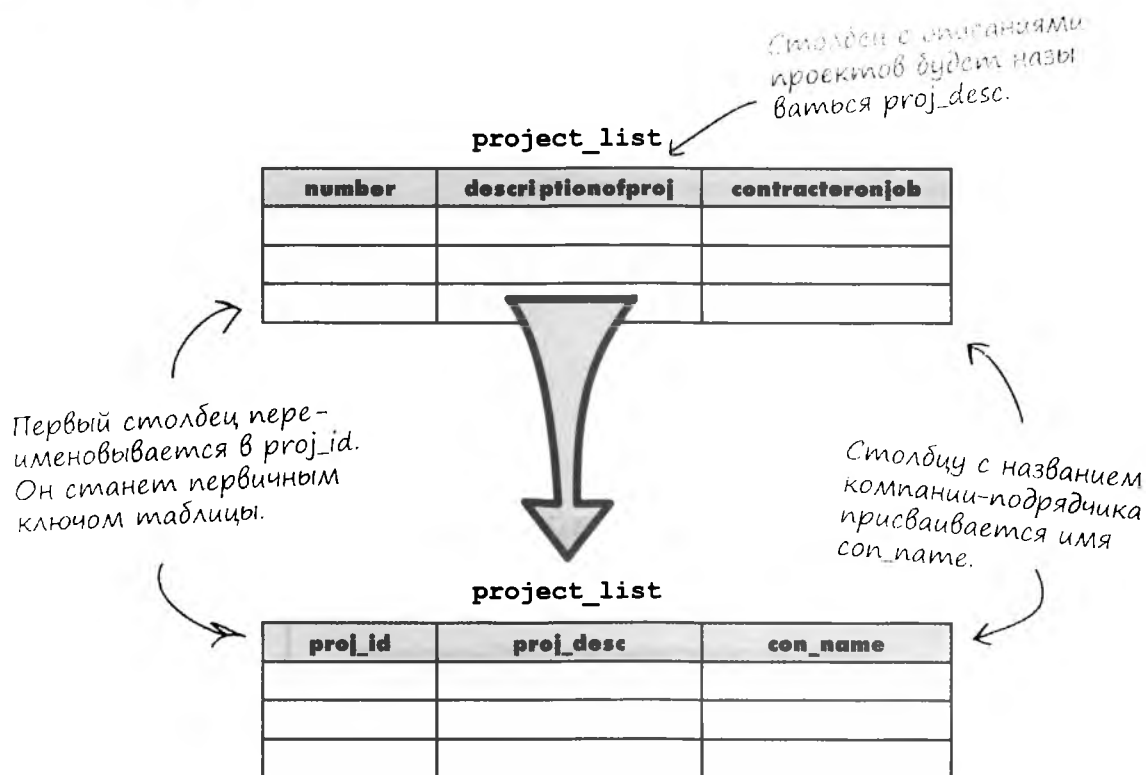
Перепланировка столбцов

Итак, план действий составлен. Теперь мы можем изменить существующие столбцы командой ALTER и привести их в соответствие с новой схемой назначения имен:

- number преобразуется в первичный ключ **proj_id**
- descriptionofproj преобразуется в столбец с описанием проекта **proj_desc**
- contractoronjob преобразуется в столбец с названием компании-подрядчика **con_name**



Остается добавить в таблицу три столбца с именами `est_cost`, `con_phone` и `start_date`.



Структурные изменения

Мы решили переоборудовать существующие столбцы под три из шести столбцов, предусмотренных запланированной структурой таблицы. Кроме переименования, также следует обратить внимание на тип данных каждого из этих столбцов.

Взгляните еще раз на структуру таблицы.

```
File Edit Window Help BadTableDesign
--> DESCRIBE projekts;
+-----+-----+-----+-----+-----+-----+
| Field          | Type       | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| number         | int(11)    | YES  |     | NULL    |      |
| descriptionofproj | varchar(50) | YES  |     | NULL    |      |
| contractoronjob  | varchar(10) | YES  |     | NULL    |      |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```



Проверьте типы столбцов и решите, соответствуют ли они тем данным, которые мы собираемся хранить в таблице.

ALTER и CHANGE

На следующем шаге мы присвоим столбцу number новое имя proj_id и установим для него атрибут AUTO_INCREMENT. Затем столбец будет назначен первичным ключом. Описание звучит устрашающе, но на самом деле все очень просто. Более того, задача решается всего одной командой.

На этот раз используется конструкция CHANGE COLUMN, потому что мы изменяем как имя, так и тип данных столбца, который ранее назывался «number».

Не забудьте, что таблице было присвоено новое имя.

«proj_id» — новое имя, которое присваивается столбцу.

...и столбец заполняется автоматически генерируемыми целыми числами, без значений NULL.

```
ALTER TABLE project_list  
CHANGE COLUMN number proj_id INT NOT NULL AUTO_INCREMENT,  
ADD PRIMARY KEY ('proj_id');
```

Эта часть команды приказывает РСУБД использовать столбец с новым именем proj_id в качестве первичного ключа.

Возьми в руку карандаш



Изобразите структуру таблицы после выполнения приведенной команды.

Ответ на с. 265.

Изменение двух столбцов одной командой SQL

А сейчас мы изменим не один, а целых два столбца *всего одной командой*. Столбцы `descriptionofproj` и `contractoronjob` будут переименованы одновременно с изменением их типов данных. Все, что для этого потребуется — включить в команду `ALTER TABLE` две строки `CHANGE COLUMN`, разделив их запятой.

«`descriptionofproj`» — старое имя столбца, которое изменяется этой командой.

«`proj_desc`» — новое имя столбца...

Увеличиваем количество символов, чтобы в таблице можно было хранить более длинные описания.

```
ALTER TABLE project_list
CHANGE COLUMN descriptionofproj proj_desc VARCHAR(100),
CHANGE COLUMN contractoronjob con_name VARCHAR(30);
```

Другое старое имя столбца, «`contractoronjob`», тоже будет заменено...

...новым именем `con_name`, а это новый тип данных.



**Будьте
осторожны!**

Преобразование столбца к новому типу данных может привести к потере информации.

Если новый тип данных несовместим со старым, то команда не выполняется, а РСУБД сообщает об ошибке в команде.

Но что еще хуже, если старый и новый типы **совместимы**, может произойти **усечение** данных.

Например, при преобразовании столбца `varchar(10)` к типу `char(1)` значение столбца 'Бонзо' превратится в 'Б'.

То же относится и к числовым типам. Тип столбца можно сменить, но данные будут преобразованы к новому типу, а это может привести к частичной потере информации!



А если я хочу изменить **тип данных** столбца (например, чтобы в нем помещалось больше символов), но при этом оставить ему прежнее имя — я могу просто повторить имя столбца, верно? Вот так:

```
ALTER TABLE myTable  
CHANGE COLUMN myColumn myColumn NEWTYPE;
```

Конечно, такой способ работает, но есть и более простой путь.

Воспользуйтесь командой `MODIFY`. Она меняет только тип данных столбца, оставляя неизменным его имя.

Допустим, вы хотите увеличить длину столбца для хранения данных `proj_desc` до `VARCHAR(120)`. Это делается так:

```
ALTER TABLE project_list  
MODIFY COLUMN proj_desc VARCHAR(120);
```

Имя изменяемого столбца.

Новый тип данных.

И не забудьте проследить за тем, чтобы новый тип не привел к усечению существующих данных!

Часть
Задаваемые Вопросы

В: А если я захочу изменить порядок столбцов? Могу ли я выполнить команду: `ALTER TABLE MODIFY COLUMN proj_desc AFTER con_name;`

О: После того как таблица будет создана, порядок столбцов изменить невозможно. Фактически лучше, что можно сделать — добавить новый столбец в нужной позиции и удалить старый, но тогда все данные в старом столбце будут потеряны.

В: Но разве хранение столбцов в неправильном порядке не создаст проблем?

О: Нет, потому что в команде `SELECT` можно указать порядок вывода столбцов в результатах запроса. Неважно, в каком порядке хранятся данные на жестком диске; вы всегда можете использовать запись вида:

```
SELECT column3, column1 FROM your_table;
```

или:

```
SELECT column1, column3 FROM your_table;
```

или получить столбцы в любом другом порядке.



Упражнение

Я говорю по телефону со своим агентом. Добавьте остальные столбцы самостоятельно, хорошо?



project_list

proj_id	proj_desc	con_name
1		
2		
3		

В таблицу необходимо добавить еще три столбца: для хранения **телефона**, **начальной даты** и **примерной стоимости работ**.

Напишите одну команду ALTER TABLE, которая добавит эти столбцы; обратите внимание на типы данных. Затем заполните приведенную ниже таблицу.

project_list



Упражнение
Решение



Я говорю по телефону со своим агентом. Добавьте остальные столбцы самостоятельно, хорошо?

project_list

proj_id	proj_desc	con_name
1		
2		
3		

В таблицу необходимо добавить еще три столбца: для хранения **телефона**, **начальной даты** и **примерной стоимости работ**.

Напишите одну команду ALTER TABLE, которая добавит эти столбцы; обратите внимание на типы данных. Затем заполните приведенную ниже таблицу.

В столбце VARCHAR из 10 символов помещается код города.

ALTER TABLE project_table

Мы добавляем новые столбцы, поэтому используется ADD.

- ADD COLUMN con_phone VARCHAR(10);
- ADD COLUMN start_date DATE;
- ADD COLUMN est_cost DECIMAL(7,2);

Напомним, что это поле DEC состоит из 7 цифр, с 2 цифрами в дробной части.

project_list

proj_id	proj_desc	con_name	con_phone	start_date	est_cost
1					
2					
3					

Стоп! Никаких лишних столбцов!

Только что выяснилось, что наш проект временно приостановлен. В результате столбец `start_date` можно удалить из таблицы. Нет смысла хранить в базе данных лишнюю информацию, которая только попусту занимает место.

В таблицах рекомендуется держать только те столбцы, с которыми вы действительно работаете. Если столбец не используется, удалите его. Если он вдруг снова понадобится вам в будущем, его можно будет легко добавить командой ALTER.

Чем больше столбцов в таблице, тем большую работу придется проделать РСУБД при обработке запросов и тем больше места занимает база данных. Возможно, при малом объеме данных это незаметно, но с увеличением объема таблицы задержка при получении результатов становится более ощутимой.



Возьми в руку карандаш



Напишите команду SQL для удаления столбца `start_date`. Мы еще не описывали ее синтаксис, но вы все равно попробуйте.

Возьми в руку карандаш



Решение

Напишите команду SQL для удаления столбца `start_date`. Мы еще не описывали ее синтаксис, но вы все равно попробуйте.

`ALTER TABLE project_table`

Имя таблицы.

`DROP COLUMN start_date;`

Столбцы удаляются командой `DROP`. Проще простого!

Имя удаляемого столбца.



Будьте осторожны!

Все данные, хранившиеся в удаленном столбце, теряются!

Будьте очень внимательны при использовании `DROP COLUMN`. Возможно, сначала стоит выполнить выборку данных из столбца, который вы собираетесь удалить, и убедиться в том, что в нем действительно нет ничего нужного! Лучше хранить в таблице лишние данные, чем лишиться жизненно необходимой информации.



Упражнение

ТАБЛИЦА НА ПРОКАЧКУ

Пришло время превратить вашу старую, скучную таблицу в настоящую «бомбу». Вы и не подозревали, что такие превращения возможны!

Итак, мы возьмем жалкую второсортную таблицу с данными подержанных автомобилей и командой ALTER преобразуем ее в новенькую и сверкающую. Задача усложняется тем, что хранящиеся в таблице данные должны остаться неповрежденными. Готовы к испытанию?

Дополнительные баллы, если вам удастся все сделать в одной команде ALTER TABLE.



AO

hooptie

color	year	make	mo	howmuch
серебристый	1998	Porsche	Boxter	17992.540
NULL	2000	Jaguar	XJ	15995
красный	2002	Cadillac	Escalade	40215.9

ЛОСАЕ

car_table

car_id	VIN	make	model	color	year	price
1	RNKLK66N33G213481	Porsche	Boxter	серебристый	1998	17992.54
2	SAEDA44B175B04113	Jaguar	XJ	NULL	2000	15995.00
3	3GYEK63NT2G280668	Cadillac	Escalade	красный	2002	40215.90



Упражнение
Решение

ТАБЛИЦА НА ПРОКАЧКУ

Пришло время превратить вашу старую, скучную таблицу в настоящую «бомбу». Вы и не подозревали, что такие превращения возможны!

Итак, мы возьмем жалкую второсортную таблицу с данными подержанных автомобилей и командой ALTER преобразуем ее в новенькую и сверкающую. Задача усложняется тем, что хранящиеся в таблице данные должны остаться неповрежденными. Готовы к испытанию?

Дополнительные баллы, если вам удастся все сделать в одной команде ALTER TABLE.



AO

hooptie

color	year	make	mo	howmuch
серебристый	1998	Porsche	Boxter	17992.540
NULL	2000	Jaguar	XJ	15995
красный	2002	Cadillac	Escalade	40215.9

POCABE

car_table

car_id	VIN	make	model	color	year	price
1	RNKLK66N33G213481	Porsche	Boxter	silver	1998	17992.54
2	SAEDA44B175B04113	Jaguar	XJ	NULL	2000	15995.00
3	3GYEK63NT2G280668	Cadillac	Escalade	red	2002	40215.90

Прежде всего выполните команду DESCRIBE и проверьте типы данных всех столбцов. Это поможет избежать возможной потери информации.

```
ALTER TABLE hooptie
RENAME TO car_table,
ALTER TABLE car_table
ADD COLUMN car_id INT NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (car_id),
ALTER TABLE car_table
ADD COLUMN VIN VARCHAR(16) SECOND,
CHANGE COLUMN mo model VARCHAR(20),
MODIFY COLUMN color AFTER model,
MODIFY COLUMN year SIXTH,
CHANGE COLUMN howmuch price DECIMAL(7,2);
```

Столбец «mo» переименовывается в «model», после чего столбцы «color» и «year» размещаются за ним.

Переименованному столбцу «model» назначается новый тип данных.

Также можно было использовать конструкцию «year AFTER model» или «year BEFORE price».

Часто задаваемые вопросы

В: Ранее вы говорили, что я не могу изменить порядок следования столбцов командой `MODIFY`, а моя РСУБД позволяет мне переставить столбцы. Как она это делает?

О: Ваша РСУБД незаметно для вас выполняет сразу несколько операций. Она копирует значения из перемещаемого столбца, сохраняет их во временной таблице, удаляет перемещаемый столбец, изменяет таблицу и создает новый столбец с таким же именем, как у старого, копирует в него данные из временной таблицы и удаляет ее.

Если столбцы уже содержат данные, а ваши программные инструменты SQL не выполняют все эти действия за вас, лучше оставить столбцы на старом месте. Вы всегда можете получить столбцы командой `SELECT` в любом нужном порядке.

В: Получается, легко изменить порядок следования столбцов можно только при добавлении нового столбца?

О: Правильно. Лучше всего заранее продумать порядок столбцов в ходе проектирования таблицы.

В: А если я случайно создам первичный ключ, а затем передумаю и захочу использовать другой столбец? Можно ли удалить атрибут первичного ключа без изменения данных, хранящихся в столбце?

О: Можно, притом очень просто:

```
ALTER TABLE your_table DROP PRIMARY KEY;
```

В: Как насчет атрибута `AUTO_INCREMENT`?

О: Его можно назначить столбцу, у которого этого атрибута нет, следующим образом:

```
ALTER TABLE your_table CHANGE your_id your_id
INT(11) NOT NULL AUTO_INCREMENT;
```

А удаление выполняется следующим образом:

```
ALTER TABLE your_table CHANGE your_id your_id
INT(11) NOT NULL;
```

Помните, что в таблице может быть только одно поле `AUTO_INCREMENT`, оно должно относиться к типу данных `INTEGER` и не может содержать `NULL`.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Команда `CHANGE` позволяет изменить как имя, так и тип данных столбца.
- Команда `MODIFY` используется для изменения только типа данных.
- Команда `DROP COLUMN` удаляет столбец с заданным именем из таблицы.
- Команда `RENAME` изменяет имя таблицы.
- Для определения порядка столбцов используются ключевые слова `FIRST`, `LAST`, `BEFORE имя_столбца`, `AFTER имя_столбца`, `SECOND`, `THIRD`, `FOURTH` и т. д.
- В некоторых РСУБД порядок столбцов может изменяться только при добавлении их в таблицу.



Теперь в моей таблице есть первичный ключ и столбец с номером телефона, но с атомарностью по-прежнему проблемы. Некоторые запросы остаются слишком сложными — как, например, запрос по названию штата в поле location.

Команда ALTER TABLE помогает улучшить структуру таблицы.

Используя ALTER TABLE вместе с командами SELECT и UPDATE, мы можем преобразовать громоздкие, неатомарные столбцы в точные и удобные атомарные столбцы. Для этого нужно лишь правильно скомбинировать уже известные вам команды SQL.

Рассмотрим команду CREATE TABLE для таблицы Ipepa my_contacts.

```
CREATE TABLE my_contacts
```

```
(
```

```

contact_id INT NOT NULL AUTO_INCREMENT
last_name VARCHAR(30) default NULL,
first_name VARCHAR(20) default NULL,
email VARCHAR(50) default NULL,
gender CHAR(1) default NULL,
birthday DATE default NULL,
profession VARCHAR(50) default NULL,
location VARCHAR(50) default NULL,
status VARCHAR(20) default NULL,
interests VARCHAR(100) default NULL,
seeking VARCHAR(100) default NULL,
PRIMARY KEY (contact_id)

```

Эти две строки создают и назначают первичный ключ.

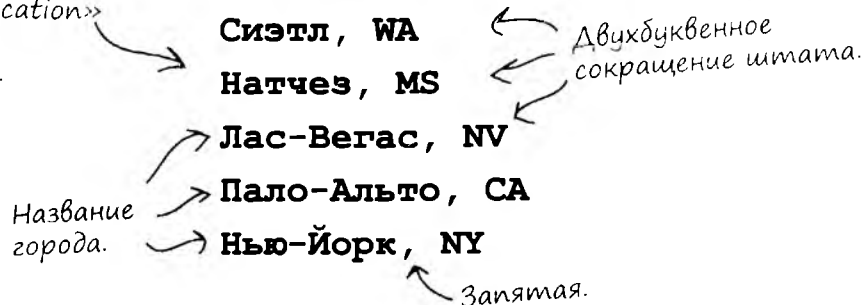
Эти четыре столбца не атомарны; мы подправим их командой ALTER TABLE.

Неатомарный столбец location

Иногда Грег хочет узнать лишь то, в каком штате или городе живет его знакомый, поэтому информацию location логично разбить на два столбца. Давайте посмотрим, как выглядят данные в этом столбце:

```
File Edit Window Help LocationLocationLocation
--> SELECT location FROM my_contacts;
+-----+
| location |
+-----+
| Сиэтл, WA |
| Натчез, MS |
| Лас Вегас, NV |
| Пало Альто, CA |
| Нью-Йорк, NY |
| Финикс, AZ |
+-----+
```

Часть данных столбца «location» таблицы my_contacts.



Данные построены по четко определенной схеме: сначала идет название города, потом запятая, а за ней двухбуквенное сокращение штата. Это поможет нам отделить город от штата.



Какую пользу принесет отделение города от штата?

Как вы думаете, что мы будем делать дальше?

В поисках закономерности

Все значения столбца `location` в таблице `my_contacts` построены по одной схеме: название города, запятая и двухбуквенное сокращение штата. Наличие четко определенного формата упростит разбиение и переход к атомарным данным.

город, **XX**

← Последние два символа всегда содержат сокращенное обозначение штата. Если бы в таблице присутствовал столбец `state`, эти данные должны были бы находиться в нем.

Запятая, которая всегда стоит перед сокращенным обозначением штата, тоже может пригодиться...

Сначала выделяем все данные до запятой и переносим их в столбец с названием города.

City Name

↑
Нам понадобится функция, которая возвращает все символы до запятой...

Затем берем два последних символа столбца `location` и переносим их в столбец `state`.

XX

↑
...и еще одна функция, которая возвращает два последних символа.

Возьми в руку карандаш



Напишите команду `ALTER TABLE`, которая добавляет в таблицу `my_contacts` столбцы `city` и `state`.

```
ALTER TABLE my_contacts
ADD COLUMN city
VARCHAR(50),
ADD COLUMN state
CHAR(2);
```

Удобные строковые функции

Мы обнаружили две закономерности в формате данных. На следующем шаге мы выделим сокращенное обозначение штата и вставим его в новый столбец **state**. Все символы, стоящие до запятой, необходимо перенести в столбец **city**. Вот как будет происходить выделение данных после того, как в таблице будут созданы новые столбцы.

Выборка двух последних символов

Для выделения заданного количества символов в столбце используются функции `RIGHT ()` и `LEFT ()` :

```
SELECT RIGHT(location, 2) FROM my_contacts;
```

Выделение символов от ПРАВОГО края значения (функция `LEFT` выделяет символы от левого края).

Используемый столбец.

Количество символов, выделяемых от правого края значения.

Выборка всех символов до запятой

Функция `SUBSTRING_INDEX ()` находит все символы текстового значения, **предшествующие заданному символу или подстроке**. Запятая заключается в апострофы, а функция `SUBSTRING_INDEX ()` возвращает все символы, стоящие перед запятой.

```
SELECT SUBSTRING_INDEX(location, ',', 1) FROM my_contacts;
```

Функция выделяет часть содержимого столбца (подстроку). Она ищет текст, заключенный в апострофы (запятая в данном случае), и возвращает все символы, предшествующие ему.

И снова имя столбца.

Запятая, которая ищется в тексте.

«1», потому что ищется первая запятая. С параметром «2» функция найдет вторую запятую в строке и вернет все предшествующие ей символы.

Текстовые значения, хранимые в столбцах `CHAR` и `VARCHAR`, тоже называются строками.

Строковые функции выделяют часть значения текстового столбца.



Упражнение

Попробуйте сами

В SQL существует ряд функций для работы со строковыми значениями в таблицах. Строковые значения хранятся в текстовых столбцах, обычно с типом данных VARCHAR или CHAR.

Ниже перечислены наиболее распространенные и полезные строковые функции. Попробуйте каждую функцию, введя соответствующую команду SELECT.

Функция **SUBSTRING** (**текст**, **начало**, **длина**) возвращает часть строкового значения **текст**, начиная с буквы в позиции **начало**. Параметр **длина** определяет длину возвращаемой строки.

```
SELECT SUBSTRING('Сан-Антонио, TX', 5, 3);
```

Функции **UPPER** (**текст**) и **LOWER** (**текст**) преобразуют все символы строки к верхнему или нижнему регистру соответственно.

```
SELECT UPPER('сШа');
```

```
SELECT LOWER('спаГеТти');
```

Функция **REVERSE** (**текст**) переставляет символы строки в обратном порядке.

```
SELECT REVERSE('спаГеТти');
```

Функции **LTRIM** (**текст**) и **RTRIM** (**текст**) возвращают строку, полученную удалением лишних пробелов в начале (у левого края) или в конце (у правого края) строки.

```
SELECT LTRIM(' собака ');
```

```
SELECT RTRIM(' кошка ');
```

Функция **LENGTH** (**текст**) возвращает количество символов в строке.

```
SELECT LENGTH('Сан-Антонио, TX');
```

ВАЖНО: Строковые функции НЕ изменяют данные, хранящиеся в таблице; они просто возвращают текст, сгенерированный в результате запроса.

* КТО И ЧТО ДЕЛАЕТ? *

Мы хотим извлечь данные из столбца location и переместить ее в два новых столбца, city и state.

Ниже перечислены операции, которые необходимо для этого выполнить. Свяжите операции с ключевыми словами SQL, которые в них используются.

SUBSTRING_INDEX()

SELECT

1. Просмотреть данные столбца, чтобы найти в них закономерность.

LEFT

ADD COLUMN

2. Добавить в таблицу пустые столбцы.

ADJUST

RIGHT

3. Извлечь часть данных из текстового столбца.

ALTER TABLE


DELETE

4. Переместить данные, полученные на шаге 3, в один из пустых столбцов.

UPDATE

INSERT

—————→ Ответы на с. 262.



Я знаю, как использовать каждую из частей, но еще не умею эффективно объединять их друг с другом. Может, попробовать включить эти строковые функции в команду UPDATE...

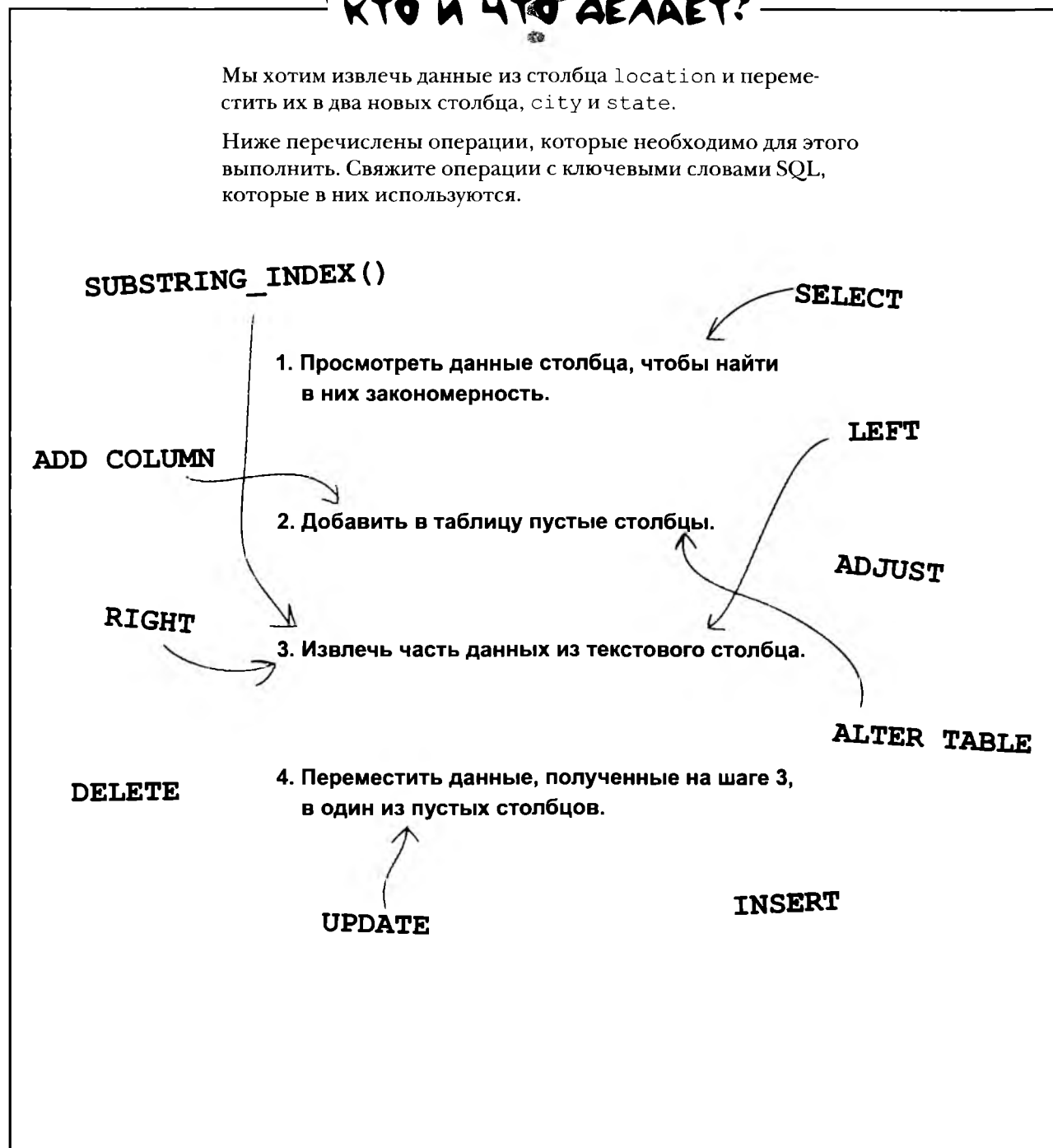
Если пользоваться только тем, что мы узнали до настоящего времени, нам придется написать команду UPDATE для последовательной обработки отдельных записей, с получением нужных данных командой SELECT.

С SQL вы сможете комбинировать команды. Переверните страницу, чтобы посмотреть как поместить значения в новые столбцы.

* КТО И ЧТО ДЕЛАЕТ? *

Мы хотим извлечь данные из столбца location и переместить их в два новых столбца, city и state.

Ниже перечислены операции, которые необходимо для этого выполнить. Свяжите операции с ключевыми словами SQL, которые в них используются.



Заполнение нового столбца существующими данными

Помните синтаксис UPDATE? Ранее мы использовали его для заполнения всех записей таблицы **одним и тем же значением**. Параметр `новое_значение` заменяется значением или именем другого столбца.

```
UPDATE table_name
```

```
SET столбец = новое_значение;
```

Команда заполняет столбец заданным значением во всех записях таблицы.

Чтобы заполнить данными новые столбцы `city` и `state`, мы включим вызов функции `RIGHT()` в команду `UPDATE`. Функция выделяет два последних символа из старого столбца `location` и помещает их в новый столбец `state`.

```
UPDATE my_contacts
```

```
SET state = RIGHT(location, 2);
```

Новый столбец для сокращенного обозначения штата.

Этот вызов функции выделяет два последних символа в столбце `location`.

Но как работает эта команда? В ней нет условия `WHERE`, определяющего обновляемые записи.

Команда работает без условия `WHERE`. Переверните страницу и посмотрите.



Как работает комбинация UPDATE с SET

Ваша РСУБД последовательно применяет команду к каждой записи таблицы до тех пор, пока все сокращенные обозначения штатов не будут перенесены в новый столбец state.

my_contacts

contact_id	location	city	state
1	Честер, NJ		
2	Кейти, TX		
3	Сан-Матео, CA		

Упрощенная версия нашей таблицы.

```
UPDATE my_contacts  
SET state = RIGHT(location, 2);
```

Команда SQL.

Давайте посмотрим, как работает этот процесс, на примере упрощенной таблицы. Сначала команда берет значение location из первой записи и обрабатывает его.

Затем команда начинает перебор сначала, находит значение location во второй строке, обрабатывает его — и так далее, пока столбец не будет разбит во всех записях, а в таблице не останется записей, которые еще не были обработаны.

**Строковые функции
могут использоваться
в командах SELECT,
UPDATE и DELETE.**

Первая итерация. → **UPDATE my_contacts
SET state = RIGHT('Честер, NJ',2)**

Команда обрабатывает столбец «location» первой записи.

Вторая итерация. → **UPDATE my_contacts
SET state = RIGHT('Кейти, TX',2)**

Теперь для второй записи.

Итерация третья и последняя (в таблице всего три записи). → **UPDATE my_contacts
SET state = RIGHT('Сан-Матео, CA',2)**

И наконец, для третьей.

Возьми в руку карандаш



Решение
Со с. 210.

Изобразите структуру таблицы после выполнения команды на с. 244.

project_list

→
Столбец
«number» превра-
тился в «proj_id»;
в нем хранятся
автоматически
увеличиваемые
значения первич-
ного ключа.

proj_id	descriptionofproj	contractoronjob
1	покраска дома	Мэрфи
2	перестройка кухни	Вальгес
3	укладка паркета	Келлер
4	кровельные работы	Джексон



Новые инструменты

Поздравляем — глава 5 осталась позади, а в вашем арсенале появилась команда ALTER. Полный список инструментов приведен в приложении III.

ALTER TABLE

Команда позволяет изменить имя таблицы и всю ее структуру без потери существующих данных.

ALTER с ADD

Добавление столбцов в таблицу в заданном порядке.

ALTER с DROP

Удаление столбцов из таблицы.

ALTER с CHANGE

Изменение как имени, так и типа данных существующего столбца.

ALTER с MODIFY

Изменение только типа данных существующего столбца.

Строковые функции

Функции, изменяющие копии содержимого текстовых столбцов, возвращаемые запросом. Исходные данные остаются неизменными.

6 Расширенные возможности SELECT

* Взглянуть на данные под другим углом *

Тогда я использую
CASE — и вижу вражеские
самолеты как на ладони!
Бабах!



Пора обзавестись более точными инструментами.

Вы уже знаете, как выполнять выборку данных, и умеете работать с условиями WHERE. Но в некоторых ситуациях нужна **точность**, на которую SELECT и WHERE не способны. В этой главе вы научитесь **упорядочивать и группировать** свои данные, а также выполнять **математические операции** с полученными результатами.

Перестройка в видеотеке

В видеотеке городка Дейтавилль дело организовано из рук вон плохо. Фильмы могут оказаться на разных полках в зависимости от того, кто из работников занимается их расстановкой. Владелец заказал новые полки, и он думает, что пришло время распределить фильмы по категориям.

DATAVILLE VIDEO



В текущей версии системы типы фильмов обозначаются флагами «да/нет», из-за чего классификация становится весьма затруднительной. Например, на какую полку ставить фильм, у которого установлены флаги категорий «Комедия» и «Фантастика»?

«Д» и «Н» — сокращения для «Да» и «Нет».

movie_table

Дата приобретения фильма.

movie_id	title	rating	drama	comedy	action	gore	scifi	for_kids	cartoon	purchased
1	Корпорация монстров	G	Н	Д	Н	Н	Н	Д	Д	6-3-2002
2	Крестный отец	R	Н	Н	Д	Д	Н	Н	Н	5-2-2001
3	Унесенные ветром	G	Д	Н	Н	Н	Н	Н	Н	20-11-1999
4	Американский пирог	R	Н	Д	Н	Н	Н	Н	Н	19-4-2003
5	Кошмар на улице Вязов	R	Н	Н	Д	Д	Н	Н	Н	19-4-2003
6	Касабланка	PG	Д	Н	Н	Н	Н	Н	Н	5-2-2001

Все эти столбцы существуют для того, чтобы работники могли отвечать на вопросы о содержании конкретных фильмов.

To: Персоналу видеотеки
From: Директор
Subject: Новым полкам — новые категории!

Всем привет,

Новые полки уже привезли, и я хочу привести в порядок наши фильмы. Мы будем использовать следующие категории:

Боевики и приключения
Драма
Комедия
Семейное кино
Ужасы
Фантастика и Фэнтези
Разное

Разберитесь сами, как заставить нашу текущую таблицу работать с новыми категориями.

А мне пора на обед.

Директор

Недостатки существующей таблицы

Ниже перечислены основные недостатки существующей таблицы.

Когда посетители возвращают фильмы, мы не знаем, куда их ставить.

Если флаг «Д» стоит в нескольких столбцах таблицы, невозможно четко определить, на какой полке должен стоять фильм. Каждый фильм должен относиться к одной категории.

Посетителям непонятно, к какому жанру относится фильм.

Посетителей сбивают с толку кровавые обложки в разделе «Комедия». В текущей версии все флаги «Д/Н» равноправны при размещении фильмов на полках.

Правка данных занимает много времени и часто приводит к ошибкам.

Каждый раз, когда в видеотеке появляется новый фильм, его необходимо занести в базу, и расставить все флаги «Д/Н». И чем больше фильмов хранится в таблице, тем больше ошибок. Иногда в столбце, в котором должен стоять флаг «Д», случайно ставится «Н», и наоборот. Столбец с категорией фильма поможет проверить содержимое столбцов «Д/Н» — а со временем и вовсе избавиться от них.

Новый столбец с информацией о категории ускорит расстановку, поможет посетителям понять, к какому жанру относится интересующий их фильм, а также сократит количество ошибок в данных.



Как бы вы преобразовали текущий набор столбцов в новые **категории**? Возможна ли в новой классификации ситуация, когда один фильм принадлежит сразу к нескольким категориям?

Классификация существующих данных

Вы уже знаете, как добавить в таблицу новый столбец category, но с его заполнением дело обстоит сложнее. К счастью, категорию каждого фильма можно определить по данным, уже хранящимся в таблице, и нам не придется просматривать все фильмы подряд.

Давайте сформулируем отношение в виде набора простых условий:

- Если 'Д' в столбце: **drama** в столбец category заносится 'драма'
- Если 'Д' в столбце: **comedy** в столбец category заносится 'комедия'
- Если 'Д' в столбце: **action** в столбец category заносится 'боевик'
- Если 'Д' в столбце: **gore** в столбец category заносится 'ужасы'
- Если 'Д' в столбце: **scifi** в столбец category заносится 'фантастика'
- Если 'Д' в столбце: **for_kids** в столбец category заносится 'семейное'
- Если 'Д' в столбце: **cartoon** и 'G' в столбце: **rating** в столбец category заносится 'семейное'
- Если 'Д' в столбце: **cartoon** и НЕ 'G' в столбце: **rating** в столбец category заносится 'разное'

Мультфильмы бывают и для взрослых. Столбец «rating» поможет определить, относится фильм к категории семейного кино или нет. Если фильму присвоен рейтинг G, мы относим его к категории «семейное», а если нет — к категории «разное».

Заполнение нового столбца

Теперь эти условия преобразуются
в команды SQL UPDATE:

```

UPDATE movie_table SET category = 'драма' where drama = 'Д';
UPDATE movie_table SET category = 'комедия' where comedy = 'Д';
UPDATE movie_table SET category = 'боевик' where action = 'Д';
UPDATE movie_table SET category = 'ужасы' where gore = 'Д';
UPDATE movie_table SET category = 'фантастика' where scifi = 'Д';
UPDATE movie_table SET category = 'семейное' where for_kids = 'Д';
UPDATE movie_table SET category = 'семейное' where cartoon = 'Д' AND rating = 'G';
UPDATE movie_table SET category = 'разное' where cartoon = 'Д' AND rating <> 'G';

```

Рейтинг отличен от 'G'

Возьми в руку карандаш

Заполните столбец category для следующих фильмов:

movie_table

title	rating	drama	comedy	action	gore	scifi	for_kids	cartoon	category
Большое приключение	G	Н	Н	Н	Н	Н	Д	Н	
Грег: Неизвестные истории	PG	Н	Н	Д	Н	Н	Н	Н	
Безумные клоуны	R	Н	Н	Н	Д	Н	Н	Н	
Параскеведекатриа-фобия	R	Д	Д	Д	Н	Д	Н	Н	
Крыса по имени Дарси	G	Н	Н	Н	Н	Н	Д	Н	
Конец очереди	R	Д	Н	Н	Д	Д	Н	Д	
Блестящие вещи	PG	Д	Н	Н	Н	Н	Н	Н	
Заберите обратно	R	Н	Д	Н	Н	Н	Н	Н	
Наживка для акул	G	Н	Н	Н	Н	Н	Д	Н	
Разгневанный пират	PG	Н	Д	Н	Н	Н	Н	Д	
Планета пригодна для жизни	PG	Н	Д	Н	Н	Д	Н	Н	

Зависит ли результат от порядка проверки столбцов Д/Н?



Возьми в руку карандаш
Решение

Заполните столбец category для следующих фильмов:

movie_table

title	rating	drama	comedy	action	gere	scifi	for_kids	cartoon	category
Большое приключение	G	Н	Н	Н	Н	Н	Д	Н	семейное
Грег: Неизвестные истории	PG	Н	Н	Д	Н	Н	Н	Н	боевик
Безумные клоуны	R	Н	Н	Н	Д	Н	Н	Н	ужасы
Параскеведекатриа-фобия	R	Д	Д	Д	Н	Д	Н	Н	?
Крыса по имени Дарси	G	Н	Н	Н	Н	Н	Д	Н	семейное
Конец очереди	R	Д	Н	Н	Д	Д	Н	Д	разное
Блестящие вещи	PG	Д	Н	Н	Н	Н	Н	Н	драма
Заберите обратно	R	Н	Д	Н	Н	Н	Н	Н	комедия
Наживка для акул	G	Н	Н	Н	Н	Н	Д	Н	?
Разгневанный пират	PG	Н	Д	Н	Н	Н	Н	Д	разное
Планета пригодна для жизни	PG	Н	Д	Н	Н	Д	Н	Н	?

Вопросительным знаком помечены столбцы, изменяемые более чем одной командой UPDATE. Значение столбца зависит от порядка выполнения UPDATE.

Зависит ли результат от порядка проверки столбцов Д/Н?

Да, зависит...

Порядок важен

Например, если столбцы будут перебираться последовательно, фильм «Параскеведекатриафобия» попадет в категорию фантастики, хотя уместнее было бы отнести его к комедиям. Если мы не знаем, к какой категории относится тот или иной фильм, возможно, лучше зачислить его в категорию «Разное».

Результат зависит от порядка проверки.

Две команды UPDATE могут изменять содержимое одного столбца.



Для маленькой таблицы этот способ подойдет, а если таблица содержит сотни столбцов? Можно ли как-то объединить все эти команды UPDATE в одну большую команду?

Да, можно написать одну большую команду UPDATE, но есть и более удобный способ.

Выражение CASE объединяет множество команд UPDATE, проверяя значение существующего столбца по условию. Если условие выполняется, то новый столбец заполняется заданным значением.

Вы даже сможете указать РСУБД, что делать с записями, не удовлетворяющими ни одному условию.

UPDATE my_table

SET новый_столбец =

CASE

WHEN столбец1 = значение1

THEN новое_значение1

WHEN column2 = значение2

THEN новое_значение2

ELSE значение3

END;

Этому столбцу присваивается одно из перечисленных ниже значений.

Начало выражения CASE.

ЕСЛИ выполняется это условие...

ТО столбцу «новый_столбец» присваивается ЭТО значение.

ЕСЛИ выполняется другое условие...

ТО столбцу «новый_столбец» присваивается другое значение.

Завершает выражение CASE и всю команду UPDATE (завершающий символ «;»).

Если ни одно из условий не выполняется, то столбцу «новый_столбец» присваивается ЭТО значение.

Отступы не влияют на обработку выражения; они всего лишь упрощают чтение кода.

UPDATE с Выражением CASE

Давайте посмотрим, как выражение CASE работает с таблицей `movie_table`.

```
UPDATE movie_table
SET category =
CASE
  WHEN drama = 'Д' THEN 'драма'
  WHEN comedy = 'Д' THEN 'комедия'
  WHEN action = 'Д' THEN 'боевик'
  WHEN gore = 'Д' THEN 'ужасы'
  WHEN scifi = 'Д' THEN 'фантастика'
  WHEN for_kids = 'Д' THEN 'семейное'
  WHEN cartoon = 'Д' THEN 'семейное'
  ELSE 'разное'
END;
```

↑
 Всем записям, не подходящим ни под одно из перечисленных условий, назначается категория «разное».

То же самое, что UPDATE movie_table SET category = 'драма' WHERE drama = 'Д' — но намного компактнее!

Значения, которые оставались неопределенными при заполнении нового столбца отдельными командами UPDATE, теперь определены.

Но также обратите внимание на новые значения для фильмов «Разгневанная пират» и «Конец очереди».

movie_table

title	rating	drama	comedy	action	gore	scifi	for_kids	cartoon	category
Большое приключение	PG	Н	Н	Н	Н	Н	Н	Д	семейное
Грег: Неизвестные истории	PG	Н	Н	Д	Н	Н	Н	Н	боевик
Безумные клоуны	R	Н	Н	Н	Д	Н	Н	Н	ужасы
Параскеведекатриафобия	R	Д	Д	Д	Н	Д	Н	Н	драма
Крыса по имени Дарси	G	Н	Н	Н	Н	Н	Д	Н	семейное
Конец очереди	R	Д	Н	Н	Д	Д	Н	Д	драма
Блестящие вещи	PG	Д	Н	Н	Н	Н	Н	Н	драма
Заберите обратно	R	Н	Д	Н	Н	Н	Н	Н	комедия
Наживка для акул	G	Н	Н	Н	Н	Н	Д	Н	семейное
Разгневанный пират	PG	Н	Д	Н	Н	Н	Н	Д	комедия
Планета пригодна для жизни	PG	Н	Т	Н	Н	Т	Н	Н	комедия

В процессе обработки значений «Д/Н» каждого фильма выражением CASE РСУБД ищет первый столбец с «Д», чтобы установить по нему категорию.

Давайте посмотрим, как происходит обработка данных фильма «Большое приключение»:

```

UPDATE movie_table
SET category =
CASE
  WHEN drama = 'Д' THEN 'драма'
  WHEN comedy = 'Д' THEN 'комедия'
  WHEN action = 'Д' THEN 'боевик'
  WHEN gore = 'Д' THEN 'ужасы'
  WHEN scifi = 'Д' THEN 'фантастика'
  WHEN for_kids = 'Д' THEN 'семейное'
  WHEN cartoon = 'Д' THEN 'семейное'
  ELSE 'разное'
END;
```

НЕТ: категория пока неизвестна
 НЕТ: категория пока неизвестна
 НЕТ: категория пока неизвестна
 НЕТ: категория пока неизвестна
 НЕТ: категория пока неизвестна
 НЕТ: категория пока неизвестна
 НЕТ: категория пока неизвестна
 НЕТ: категория пока неизвестна
 ДА: в столбец «category» заносится значение 'семейное'; управление передается END, выполнение команды завершается.

Теперь рассмотрим запись с совпадениями в нескольких категориях. Как и в предыдущем случае, категория фильма определяется первым найденным столбцом, содержащим «Д».

Вот что происходит при обработке записи фильма «Параскеведекатрифобия»:

```

UPDATE movie_table
SET category =
CASE
  WHEN drama = 'Д' THEN 'драма'
  WHEN comedy = 'Д' THEN 'комедия'
  WHEN action = 'Д' THEN 'боевик'
  WHEN gore = 'Д' THEN 'ужасы'
  WHEN scifi = 'Д' THEN 'фантастика'
  WHEN for_kids = 'Д' THEN 'семейное'
  WHEN cartoon = 'Д' THEN 'семейное'
  ELSE 'разное'
END;
```

ДА: фильму назначается категория «драма»; управление передается END, выполнение кода завершается. Остальные значения Д игнорируются.

Похоже, у нас проблемы

Произошло нечто непредвиденное. «Большое приключение» – мультфильм для взрослых – каким-то образом попал в категорию «семейного кино».

Сообщение

Дата Сегодня Time 13.41

Кому Директору

За время вашего отсутствия
Очень сердитый клиент

Звонил	<input checked="" type="checkbox"/>	Просил перезвонить	<input checked="" type="checkbox"/>
Хотел говорить с вами	<input type="checkbox"/>	Будет звонить снова	<input type="checkbox"/>
Хочет встретиться	<input type="checkbox"/>	Отвечал на ваш звонок	<input type="checkbox"/>

Сообщение *Одна пожилая леди жалуется, что ее внук посмотрел мультик с обилием нецензурных выражений – а теперь бежит за своей сестрой и называет ее %#!@*

Принял Я Срочно

Возьми в руку карандаш



Измените выражение CASE так, чтобы мультфильмы (столбец cartoon) попадали в категорию 'разное', а не 'семейное'. Только если мультфильму присвоен рейтинг G, он помещается в категорию семейного кино.

МОЗГОВОЙ ШТУРМ

Как использовать рейтинг R, чтобы подобные инциденты не происходили в будущем?

Возьми в руку карандаш



Решение

```
UPDATE movie_table
SET category =
CASE
  WHEN drama = 'Д' THEN 'драма'
  WHEN comedy = 'Д' THEN 'комедия'
  WHEN action = 'Д' THEN 'боевик'
  WHEN gore = 'Д' THEN 'ужасы'
  WHEN scifi = 'Д' THEN 'фантастика'
  WHEN for_kids = 'Д' THEN 'семейное'
  WHEN cartoon = 'Д' AND rating = 'G' THEN 'семейное'
  ELSE 'misc'
END;
```

Условие может состоять из нескольких частей: ключевое слово AND проверяет, что это мультфильм, и он имеет рейтинг 'G'. Если составное условие выполняется, фильму назначается категория 'семейное'.

Часть Задаваемые Вопросы

В: Обязательно ли использовать ELSE?

О: Нет, не обязательно. Не включайте это условие, если оно не требуется, но обычно бывает удобно предусмотреть способ обновления столбца, если ни одно условие не выполняется. В такой ситуации лучше заполнять столбец каким-то значением, отличным от NULL.

В: Что произойдет, если ни одно из условий WHEN не подходит, а ELSE отсутствует?

О: Обновляемый столбец не изменяется.

В: А если я хочу использовать выражение CASE только для некоторых значений столбцов? Например, если выражение CASE должно применяться только в случае category = 'разное'. Могу ли я использовать WHERE?

О: Да, после ключевого слова END можно добавить условие WHERE. Выражение CASE будет применяться только при выполнении условия WHERE.

В: Можно ли использовать выражение CASE с другими командами, кроме UPDATE?

О: Да. Выражение CASE может использоваться с командами SELECT, INSERT, DELETE — и, как вы уже видели, UPDATE.

КОМАНДЫ С CASE

Беспокойному директору видеотеки пришла в голову очередная «гениальная идея». Прочтите его сообщение и напишите одну команду SQL, которая делает то, что ему нужно.

Оказывается, с новыми категориями посетителям стало трудно найти нужный фильм. Напишите команду, которая удаляет только что созданные категории с рейтингом R.

Наконец, удалите столбцы Д/Н, которые нам больше не нужны.

To: Персоналу видеотеки
From: Директор
Subject: Новые категории

Дорогие коллеги!

Я решил создать несколько новых разделов. Мне кажется, что фильмы с рейтингом R должны находиться на полках отдельно от фильмов с рейтингами G и PG. Давайте создадим 5 новых категорий:

ужасы-г
боевик-г
драма-г
комедия-г
фантастика-г

А если в категории 'разное' есть фильмы с рейтингом G, давайте перенесем их в категорию семейного кино.

Всем спасибо,
Директор

КОМАНДЫ С CASE

Беспокойному директору видеотеки пришла в голову очередная «гениальная идея». Прочтите его сообщение и напишите одну команду SQL, которая делает то, что ему нужно.

```
UPDATE movietable
SET category =
CASE
  WHEN drama = 'Д' AND rating = 'R' THEN 'драма-r'
  WHEN comedy = 'Д' AND rating = 'R' THEN 'комедия-r'
  WHEN action = 'Д' AND rating = 'R' THEN 'боевик-r'
  WHEN gore = 'Д' AND rating = 'R' THEN 'ужасы-r'
  WHEN scifi = 'Д' AND rating = 'R' THEN 'фантастика-r'
  WHEN category = 'разное' AND rating = 'G' THEN 'семейное'
END;
```

Оказывается, с новыми категориями посетителям стало трудно найти нужный фильм. Напишите команду, которая удаляет только что созданные категории с рейтингом R.

```
UPDATE movietable
SET category =
CASE
  WHEN category = 'драма-r' THEN 'драма'
  WHEN category = 'комедия-r' THEN 'комедия'
  WHEN category = 'боевик-r' THEN 'боевик'
  WHEN category = 'ужасы-r' THEN 'ужасы'
  WHEN category = 'фантастика-r' THEN 'фантастика'
END;
```

Наконец, удалите столбцы Д/Н, которые нам больше не нужны.

```
ALTER TABLE movietable
DROP COLUMN drama,
DROP COLUMN comedy,
DROP COLUMN action,
DROP COLUMN gore,
DROP COLUMN scifi,
DROP COLUMN for_kids,
DROP COLUMN cartoon;
```

To: Персоналу видеотеки
From: Директор
Subject: Новые категории

Дорогие коллеги!

Я решил создать несколько новых разделов. Мне кажется, что фильмы с рейтингом R должны находиться на полках отдельно от фильмов с рейтингами G и PG. Давайте создадим 5 новых категорий:

ужасы-г
боевик-г
драма-г
комедия-г
фантастика-г

А если в категории 'разное' есть фильмы с рейтингом G, давайте перенесем их в категорию семейного кино.

Всем спасибо,
Директор

Трудности с таблицами

Когда в видеотеке появляется новый фильм, информация о нем заносится в базу данных, а его описание становится последней записью в таблице. Информация о фильмах в таблице никак не упорядочивается. И теперь, когда пришло время заново расставлять фильмы по полкам, возникает проблема. На каждой новой полке помещается 20 фильмов, а на каждом из 3000 с лишним фильмов должна присутствовать наклейка с обозначением категории. *Требуется получить список фильмов каждой категории, отсортированных в алфавитном порядке.*

Вы уже знаете, как обратиться к базе данных с запросом на выборку всех фильмов определенной категории, но теперь требуется каким-то образом упорядочить названия фильмов внутри категорий.

movie_table

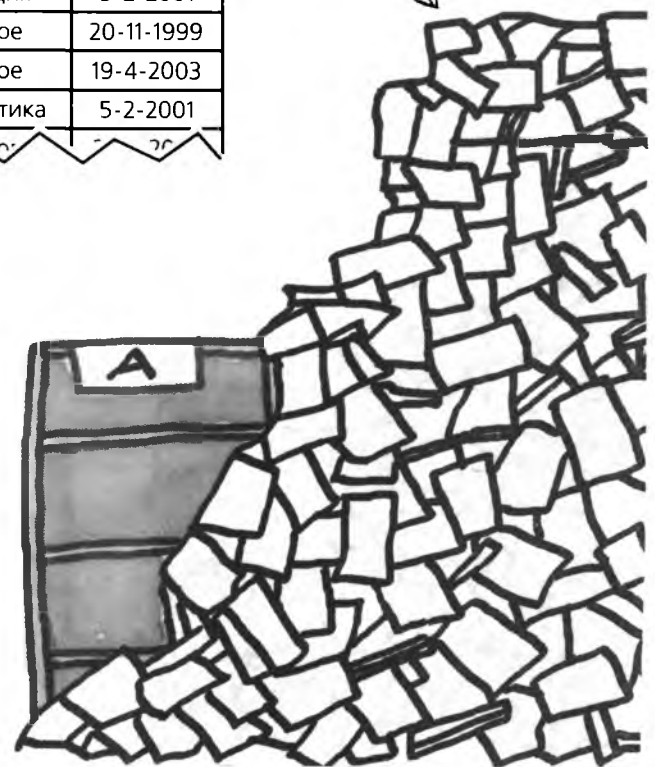
movie_id	title	rating	category	purchased
83	Большое приключение	G	семейное	6-3-2002
84	Грег: Неизвестные истории	PG	боевик	5-2-2001
85	Безумные клоуны	R	ужасы	20-11-1999
86	Параскеведекатриафобия	R	боевик	19-4-2003
87	Крыса по имени Дарси	G	семейное	19-4-2003
88	Конец очереди	R	разное	5-2-2001
89	Блестящие вещи	PG	драма	6-3-2002
90	Заберите обратно	R	комедия	5-2-2001
91	Наживка для акул	G	разное	20-11-1999
92	Разгневанный пират	PG	разное	19-4-2003
93	Планета пригодна для жизни	PG	фантастика	5-2-2001

Небольшая часть из 3000 с лишним фильмов, хранящихся в видеотеке.



**МОЗГОВОЙ
ШТУРМ**

Как упорядочить данные по алфавиту с использованием команды SQL?



Упорядочение результатов выборки

На каждом из 3000 с лишним фильмов необходимо разместить наклейку с обозначением категории, после чего фильмы расставляются на полке в алфавитном порядке.

Нам нужен список фильмов, в котором внутри каждой категории названия упорядочены по алфавиту. Вы уже умеете пользоваться командой SELECT, можете легко получить список фильмов заданной категории, и даже выполнить выборку по первой букве названия и по категории.

Но для упорядочения такого большого списка фильмов придется выполнить огромное количество команд SELECT. Вот лишь небольшая часть:

```
SELECT title, category FROM movie_table WHERE title LIKE 'A%' AND category = 'семейное';  
SELECT title, category FROM movie_table WHERE title LIKE 'Б%' AND category = 'семейное';  
SELECT title, category FROM movie_table WHERE title LIKE 'В%' AND category = 'семейное';  
SELECT title, category FROM movie_table WHERE title LIKE 'Г%' AND category = 'семейное';  
SELECT title, category FROM movie_table WHERE title LIKE 'Д%' AND category = 'семейное';  
SELECT title, category FROM movie_table WHERE title LIKE 'Е%' AND category = 'семейное';  
SELECT title, category FROM movie_table WHERE title LIKE 'Ж%' AND category = 'семейное';
```

Нужно знать название, чтобы найти сам фильм, и категорию, чтобы снабдить его наклейкой и поставить на полку.

Буква, с которой начинается название фильма.

Категория, в которой осуществляется поиск.

А еще не забудьте о фильмах, названия которых начинаются с цифры («101 далматин») или «300 спартанцев»).



Как вы думаете, где в этом списке будут находиться фильмы, названия которых начинаются с цифры или неалфавитного символа (например, с восклицательного знака)?



Возьми в руку карандаш



Чтобы определить правильный порядок записей, нам все равно придется вручную упорядочивать названия фильмов по буквам, следующими за начальной «А».

Перед вами результаты одного из 200 (или около того) запросов. Попробуйте расставить названия фильмов по алфавиту вручную.

`SELECT title, category FROM movie_table WHERE title LIKE 'A%' AND category = 'семейное';`

← Часть результатов запроса

title	category
Авиаторы	семейное
Алый рассвет	семейное
Арахисовое масло	семейное
Американская мечта	семейное
Аквалангисты	семейное
Абракадабра	семейное
Асфальтовые дороги	семейное
Алиса в стране чудес	семейное
Арбузная косточка	семейное
Апельсиновый джем	семейное
А когда я вырасту?	семейное
Ааргх!	семейное
Античный мир	семейное
Аляска: Страна лососей	семейное
Ангелы	семейное
Анна переживает	семейное
Авантюрный роман	семейное
Астронавты	семейное
Акулий зуб	семейное
Ааргх! 2	семейное

Возьми в руку карандаш



Решение

Чтобы определить правильный порядок записей, нам все равно придется вручную упорядочивать названия фильмов по буквам, следующими за начальной «А».

Перед вами результаты одного из 200 (или около того) запросов. Попробуйте расставить названия фильмов по алфавиту вручную.

```
SELECT title, category FROM movie_table WHERE title LIKE 'A%' AND category = 'семейное';
```

title	category
А когда я вырасту?	семейное
Аархг!	семейное
Аархг! 2	семейное
Абракадабра	семейное
Авантюрный роман	семейное
Авиаторы	семейное
Аквалангисты	семейное
Акулий зуб	семейное
Алиса в стране чудес	семейное
Алый рассвет	семейное
Аляска: Страна лососей	семейное
Американская мечта	семейное
Ангелы	семейное
Анна переживает	семейное
Античный мир	семейное
Апельсиновый джем	семейное
Арахисовое масло	семейное
Арбузная косточка	семейное
Астронавты	семейное
Асфальтовые дороги	семейное

Сколько времени вам потребовалось на то, чтобы упорядочить эти 20 фильмов?

А представляете, как долго придется возиться, если фильмов будет более 3000?

ORDER BY

Хотите упорядочить результаты своего запроса? Это совсем не-сложно — включите в команду **SELECT** ключевые слова **ORDER BY** и имя столбца таблицы.

Пока никаких сюрпризов — все так же, как в только что выполненной команде **SELECT**.

```
SELECT title, category
FROM movie_table
WHERE
title LIKE 'A%'
AND
category = 'семейное'
ORDER BY title;
```

А вот этого раньше не было... Эта часть означает, что запрос должен вернуть данные упорядоченными в алфавитном порядке по значению столбца «title».

Хотите сказать, что это единственный способ упорядочения результатов? Да я НИ ЗА ЧТО не стану этим заниматься для каждой буквы алфавита.



Возьми в руку карандаш



И верно. Что следует убрать из этого запроса, чтобы расширить его возможности?

СТОП! Выполните это упражнение до того, как перевернете страницу.

Упорядочение по одному столбцу

Если включить в запрос условие ORDER BY title, нам уже не придется отбирать названия, начинающиеся с определенной буквы — запрос сам вернет данные, выстроенные в алфавитном порядке по значению столбца title.

Для этого нужно лишь исключить из запроса условие title LIKE, а ORDER BY title сделает все остальное.

Возьми в руку карандаш

Решение

Что следует убрать из этого запроса, чтобы расширить его возможности?

```

SELECT title, category
FROM movie_table
WHERE
title LIKE 'A%'
AND
category = 'семейное'
ORDER BY title;
    
```

```

SELECT title, category
FROM movie_table
WHERE
category = 'семейное'
ORDER BY title;
    
```

На этот раз будет выведен полный список фильмов из категории «семейного кино».

И что еще лучше, в список будут включены фильмы, названия которых начинаются с цифры. Они будут находиться в самом начале списка.

Это не все результаты выборки; нам не хватит места, чтобы привести все названия до буквы «Я».

ORDER BY позволяет отсортировать данные любого столбца.

Обратите внимание: несколько первых названий начинаются с цифры.

title	category
1 безумный прищелец	семейное
10 жуков	семейное
101 овчарка	семейное
13-й день рождения	семейное
2 + 2 = 5	семейное
3001 способ потерпеть неудачу	семейное
8 рук лучше 2	семейное
А когда я вырасту?	семейное
Ааргх!	семейное
Ааргх! 2	семейное
Абракадабра	семейное
Авантюрный роман	семейное
Авиаторы	семейное
Аквалангисты	семейное
Акулий зуб	семейное
Алиса в стране чудес	семейное
Алый рассвет	семейное
Аляска: Страна лососей	семейное
Американская мечта	семейное
Ангелы	семейное
Анна переживает	семейное
Античный мир	семейное
Апельсиновый джем	семейное
Арахисовое масло	семейное
Арбузная косточка	семейное
Астронавты	семейное
Асфальтовые дороги	семейное



Упражнение

Создайте простую таблицу, состоящую из единственного столбца CHAR (1) с именем «test_chars».

Вставьте в нее числа, буквы (верхнего и нижнего регистра) и не-алфавитные символы, приведенные ниже (каждый символ вставляется в отдельную запись). Вставьте пробел и оставьте одну запись со значением NULL.

Примените к столбцу запрос на выборку с новой конструкцией ORDER BY. Заполните пустые места в книге.

**0123ABCDabcd!@#\$%^&*() -_+= []
{ } ; : ' » \ | ` ~ , . < > / ?**

Правила SQL

Выполните запрос ORDER BY и заполните пустые места, руководствуясь порядком символов в результатах выборки

Не-алфавитные символы следуют цифр.

Цифры следуют текстовых символов.

Значения NULL следуют цифр.

Значения NULL следуют алфавитных символов.

Символы верхнего регистра следуют символов нижнего регистра.

«A 1» будет следовать «A1».

Правила SQL

Выполните запрос ORDER BY и расставьте эти символы в порядке их следования в результатах.

+ = ! (& ~ " * @ ? ' <

Помните, как вставить апостроф? Это не так просто.



Создайте простую таблицу, состоящую из единственного столбца CHAR (1) с именем «test_chars».

Вставьте в нее числа, буквы (верхнего и нижнего регистра) и не-алфавитные символы, приведенные ниже (каждый символ вставляется в отдельную запись). Вставьте пробел и оставьте одну запись со значением NULL.

Примените к столбцу запрос на выборку с новой конструкцией ORDER BY. Заполните пустые места в книге.

!"#\$%&'()*+,-./0123:;<=>
?@ABCD[\]^_`abcd{|}~

Возможный порядок следования символов в результатах выборки. Обратите внимание на пробел в начале. Ваш порядок может быть немного другим в зависимости от РСУБД. Важно понимать, что порядок СУЩЕСТВУЕТ — и знать его для вашей РСУБД.

Правила SQL

Выполните запрос ORDER BY и заполните пустые места, руководствуясь порядком символов в результатах выборки

Не-алфавитные символы следуют
до и после цифр.

Цифры следуют до текстовых символов.

Значения NULL следуют до цифр.

Значения NULL следуют до алфавитных символов.

Символы верхнего регистра следуют до символов нижнего регистра.

«A 1» будет следовать до «A1».

Правила SQL

Выполните запрос ORDER BY и расставьте эти символы в порядке их следования в результатах.

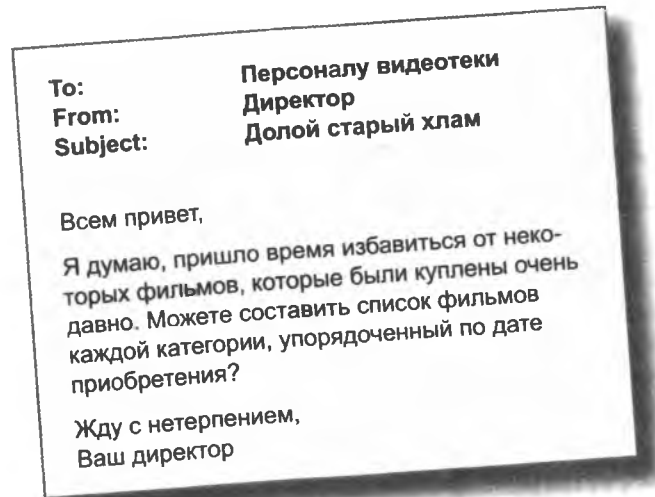
+ = ! (& ~ "
* @ ? ' "

!" & ' (* + = ?
@ ~

ORDER с двумя столбцами

Похоже, все идет прекрасно: мы можем расставить фильмы по алфавиту и построить алфавитный список для каждой категории.

К сожалению, директор придумал для вас еще кое-что...



К счастью, в одной команде можно упорядочить данные сразу по нескольким столбцам.

Дата приобретения включается в результат выборки.

```
SELECT title, category, purchased
FROM movie_table
ORDER BY category, purchased;
```

↑
Столбец первичной сортировки. Мы получим список всех фильмов в магазине, упорядоченный по столбцу «category».

↑
А по этому столбцу будет выполняться вторичная сортировка ПОСЛЕ сортировки по столбцу «category».

МОЗГОВОЙ ШТУРМ

Где будут находиться самые старые фильмы — в начале или конце каждой категории? И что произойдет, если два фильма в одной категории имеют одинаковую дату приобретения? Какой из них окажется на первом месте?

ORDER с несколькими столбцами

Возможности сортировки не ограничиваются всего двумя столбцами. Вы можете выполнить сортировку по любому количеству столбцов, чтобы получить нужную информацию.

Взгляните на следующую конструкцию ORDER BY с тремя столбцами. Ниже показано, как происходит сортировка.

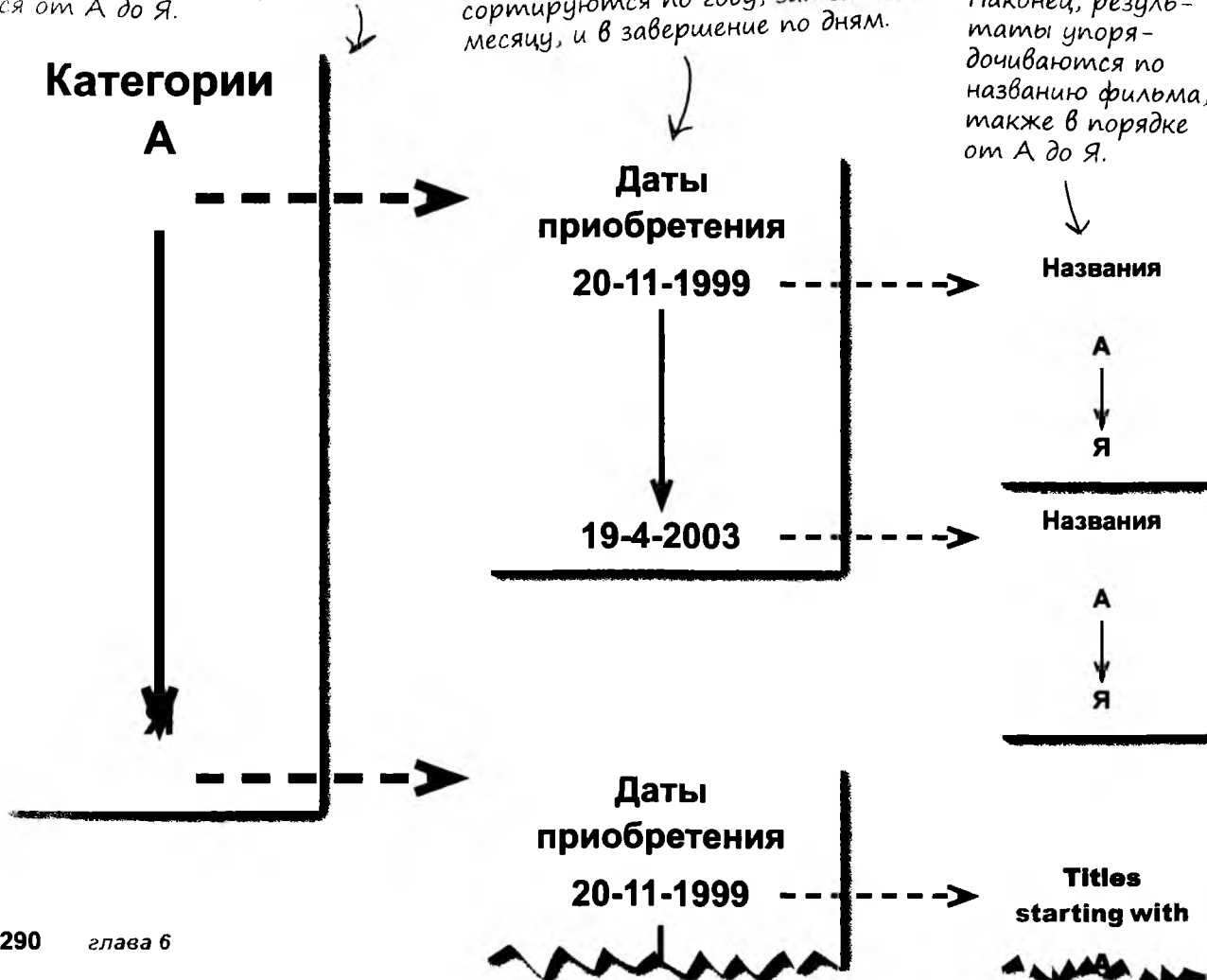
```
SELECT * FROM movie_table  
ORDER BY category, purchased, title;
```

Сначала результаты упорядочиваются по столбцу «category», потому что он находится на первом месте в списке ORDER BY. Категории упорядочиваются от А до Я.

Затем результаты (упорядоченные по категориям, начиная с А) сортируются по дате, начиная с самых старых дат. Даты всегда сортируются по году, затем по месяцу, и в завершение по дням.

Данные можно отсортировать по любому количеству столбцов.

Наконец, результаты упорядочиваются по названию фильма, также в порядке от А до Я.



Упорядоченная таблица

Давайте посмотрим, какие данные вернет команда SELECT для исходной таблицы фильмов.

Исходная таблица movie_table.

Никакого порядка здесь нет; фильмы перечисляются в порядке вставки записей в таблицу.

movie_id	title	rating	category	purchased
83	Большое приключение	G	семейное	6-3-2002
84	Грег: Неизвестные истории	PG	боевик	5-2-2001
85	Безумные клоуны	R	ужасы	20-11-1999
86	Параскеведекатриафобия	R	боевик	19-4-2003
87	Крыса по имени Дарси	G	семейное	19-4-2003
88	Конец очереди	R	разное	5-2-2001
89	Блестящие вещи	PG	драма	6-3-2002
90	Заберите обратно	R	комедия	5-2-2001
91	Наживка для акул	G	разное	20-11-1999
92	Разгневанный пират	PG	разное	19-4-2003
93	Планета пригодна для жизни	PG	фантастика	5-2-2001
94	Cows Go Wild	R	ужасы	3-8-2001

...и упорядоченные результаты нашего запроса:

Завершающая сортировка

Первичная сортировка

Вторичная сортировка

movie_id	title	rating	category	purchased
84	Грег: Неизвестные истории	PG	боевик	5-2-2001
86	Параскеведекатриафобия	R	боевик	19-4-2003
89	Блестящие вещи	PG	драма	6-3-2002
90	Заберите обратно	R	комедия	5-2-2001
91	Наживка для акул	G	разное	20-11-1999
88	Конец очереди	R	разное	5-2-2001
92	Разгневанный пират	PG	разное	19-4-2003
83	Большое приключение	G	семейное	6-3-2002
87	Крыса по имени Дарси	G	семейное	19-4-2003
85	Безумные клоуны	R	ужасы	20-11-1999



Не люблю старое кино... А если я захочу сначала увидеть новые фильмы? Неужели придется читать список от конца к началу?

В SQL есть ключевое слово для изменения направления сортировки.

По умолчанию SQL упорядочивает столбцы ORDER BY по возрастанию: от А к Я, от 1 к 99999 и т. д. Если вы предпочитаете получить данные в обратном порядке, укажите после имени столбца ключевое слово DESC.

Часто задаваемые вопросы

В: Но мы использовали ключевое слово DESC для получения ОПИСАНИЯ таблицы. Вы уверены, что оно может использоваться для изменения порядка?

О: Да, все зависит от контекста. Если поставить DESC перед именем таблицы — например, `DESC movie_table;` — то вы получите описание таблицы. В этом случае оно интерпретируется как сокращение от DESCRIBE.

В условии ORDER оно интерпретируется как сокращение от DESCENDING, и определяет порядок результатов.

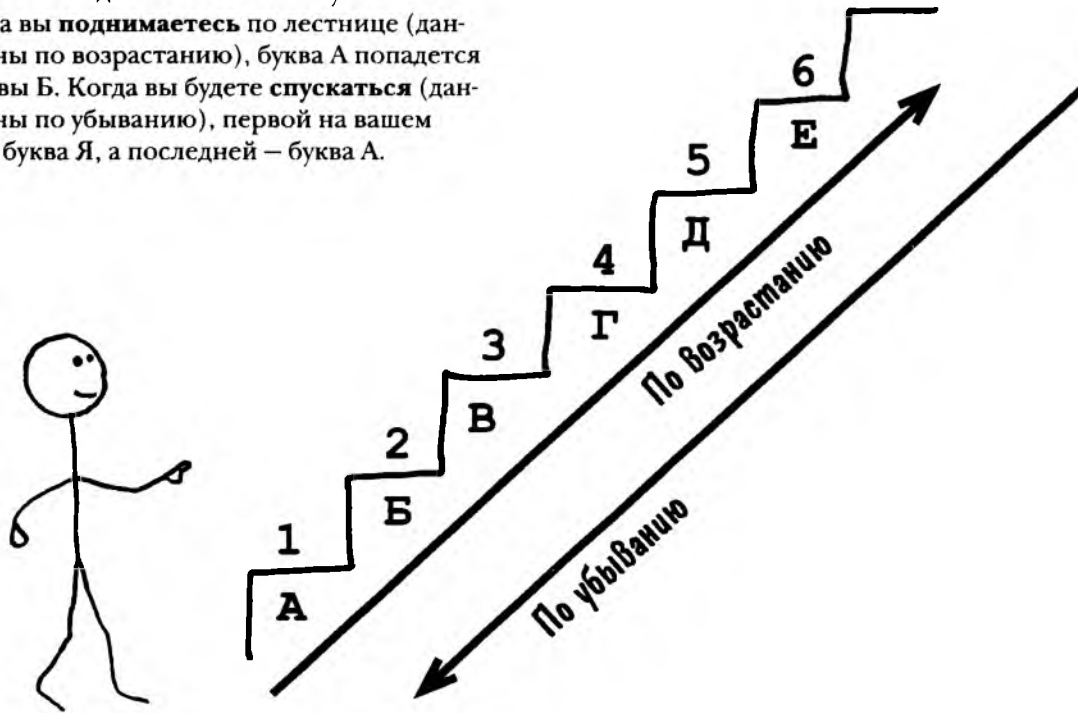
В: Я могу использовать в своих запросах полные слова DESCRIBE и DESCENDING, чтобы избежать путаницы?

О: Вы можете использовать DESCRIBE, но DESCENDING работать не будет.

Ключевое слово DESC после имени столбца в условии ORDER BY упорядочивает результаты по убыванию.

DESC и изменение порядка данных

Представьте, что ваши данные стоят на ступеньках лестницы. Когда вы **поднимаетесь** по лестнице (данные упорядочены по возрастанию), буква А попадетс вам раньше буквы Б. Когда вы будете **спускаться** (данные упорядочены по убыванию), первой на вашем пути попадетс буква Я, а последней – буква А.



Следующий запрос возвращает список фильмов, упорядоченных по дате приобретения, начиная с самых **новых**. Для каждой даты фильма, приобретенные в этот день, перечисляются в алфавитном порядке.

```
SELECT title, purchased
FROM movie_table
ORDER BY title ASC, purchased DESC;
```

Здесь можно указать ключевое слово *ASC*, но это не обязательно. Достаточно помнить, что по умолчанию данные сортируются по возрастанию.

Чтобы данные были отсортированы от Я до А (или от 9 до 1), используйте ключевое слово *DESC*.

To: Персоналу видеотеки
From: Директор
Subject: Налетай!

Всем привет!

Все просто прекрасно! Фильмы стоят на нужных местах, и благодаря этим вашим хитрым условиям ORDER BY каждый клиент может легко найти именно то, что ему нужно.

Чтобы наградить вас всех за примерную работу, завтра в моем доме состоится вечеринка с пиццей. Собираемся к 18:00.

И не забудьте принести отчеты!
Ваш директор

P. S. И не слишком наряжайтесь, мне тут нужно передвинуть кое-какую мебель...

Проблемы с печеньем

Руководитель местной группы девочек-скаутов пытается разобраться, кто из ее подопечных продал больше всего печенья. Пока у нее есть таблица с данными о продажах каждой девочки за день.

Я должна как можно скорее определить победителя.



Эдвина, руководитель группы

Возьми в руку карандаш



Девочка,
продавшая
печенье

cookie_sales

Заработанная
сумма

Дата
продажи

ID	first_name	sales	sale_date
1	Линдси	32.02	6-3-2007
2	Пэрис	26.53	6-3-2007
3	Бритни	11.25	6-3-2007
4	Николь	18.96	6-3-2007
5	Линдси	9.16	7-3-2007
6	Пэрис	1.52	7-3-2007
7	Бритни	43.21	7-3-2007
8	Николь	8.05	7-3-2007
9	Линдси	17.62	8-3-2007
10	Пэрис	24.19	8-3-2007
11	Бритни	3.40	8-3-2007
12	Николь	15.21	8-3-2007
13	Линдси	0	9-3-2007
14	Пэрис	31.99	9-3-2007
15	Бритни	2.58	9-3-2007
16	Николь	0	9-3-2007
17	Линдси	2.34	10-3-2007
18	Пэрис	13.44	10-3-2007
19	Бритни	8.78	10-3-2007
20	Николь	26.82	10-3-2007
21	Линдси	3.71	11-3-2007
22	Пэрис	0.56	11-3-2007
23	Бритни	34.19	11-3-2007
24	Николь	7.77	11-3-2007
25	Линдси	16.23	12-3-2007
26	Пэрис	0	12-3-2007
27	Бритни	4.50	12-3-2007
28	Николь	19.22	12-3-2007

Девочка, продавшая больше всего печенья, награждается бесплатными уроками верховой езды. Все девочки хотят победить, поэтому Эдвине очень важно побыстрее определить победителя, пока дело не дошло до ссоры.

Используйте свои навыки обращения с **ORDER BY** и напишите запрос, который поможет Эдвине узнать имя победителя.

Возьми в руку карандаш Решение



Девочка, продавшая больше всего печенья, награждается бесплатными уроками верховой езды. Все девочки хотят победить, поэтому Эдвине очень важно побыстрее определить победителя, пока дело не дошло до ссоры.

Используйте свои навыки обращения с ORDER BY и напишите запрос, который поможет Эдвине узнать имя победителя.

```
SELECT first_name, sales  
FROM cookie_sales  
ORDER BY first_name;
```

← Это запрос...

first_name	sales
Николь	19.22
Николь	0.00
Николь	8.05
Николь	26.82
Николь	7.77
Николь	15.21
Николь	18.96
Бритни	3.40
Бритни	2.58
Бритни	4.50
Бритни	11.25
Бритни	8.78
Бритни	43.21
Бритни	34.19
Линдси	17.62
Линдси	9.16
Линдси	0.00
Линдси	32.02
Линдси	2.34
Линдси	3.71
Линдси	16.23
Пэрис	26.53
Пэрис	0.00
Пэрис	0.56
Пэрис	1.52
Пэрис	13.44
Пэрис	24.19
Пэрис	31.99

← ...а это его результаты.

96.03

107.91

81.08

98.23

Чтобы определить победителя, нам все равно придется складывать данные о продажах каждой девочки вручную.

SUM сложит числа за нас

Ставки высоки. Мы не можем допустить ошибку и рассердить девочек-скаутов. Однако числа не обязательно складывать вручную — «черную работу» можно поручить SQL.

В языке SQL есть специальные ключевые слова, называемые *функциями*. Каждая функция выполняет некоторую операцию с одним или несколькими значениями. Первая функция, которую мы вам покажем, выполняет математическую операцию со столбцом. Функция **SUM** *суммирует значения столбца*, указанного в круглых скобках. Давайте посмотрим, как она работает.

Функция **SUM** суммирует значения столбца «sales».

SUM — *функция*, то есть это ключевое слово выполняет операцию со столбцом, указанным в круглых скобках.

```
SELECT SUM(sales)
FROM cookie_sales
WHERE first_name = 'Николь';
```

Это условие ограничивает запрос, чтобы он складывал только данные продаж Николь. Без него запрос просуммирует все содержимое столбца «sales».

```
File Edit Window Help TheWinners1
> SELECT SUM(sales) FROM cookie_sales
-> WHERE first_name = 'Николь';
+-----+
| SUM(sales) |
+-----+
|      96.03 |
+-----+
1 row in set (0.00 sec)
```

Теперь осталось вычислить еще три суммы, и наша работа завершена. Однако было бы намного проще и удобнее сделать все в одном запросе...

Попробуйте сами



Упражнение

Попробуйте сделать это самостоятельно. Создайте таблицу, похожую на `cookie_sales`, вставьте в нее несколько чисел и проверьте, как работают запросы, приведенные на нескольких ближайших страницах.

Суммирование с использованием GROUP BY

Данные о продажах печенья всеми девочками можно просуммировать в одном запросе — для этого в команду SUM включается условие GROUP BY. Такая команда группирует все записи с именем каждой девочки и суммирует данные продаж в каждой группе.

```
SELECT first_name, SUM(sales)
FROM cookie_sales
GROUP BY first_name
ORDER BY SUM(sales) DESC;
```

Суммирует данные столбца «sales».

Группирует значения столбца «first_name».

Упорядочение по той же функции SUM, которая использовалась для выборки.

Значения должны выводиться по убыванию, чтобы нам было проще определить победителя.

Команда суммирует все значения столбца «sales» в группе каждого значения «first_name».

first_name	sales
Николь	19.22
Николь	0.00
Николь	8.05
Николь	26.82
Николь	7.77
Николь	15.21
Николь	18.96

first_name	sales
Пэрис	26.53
Пэрис	0.00
Пэрис	0.56
Пэрис	1.52
Пэрис	13.44
Пэрис	24.19
Пэрис	31.99

first_name	sales
Линдси	17.62
Линдси	9.16
Линдси	0.00
Линдси	32.02
Линдси	2.34
Линдси	3.71
Линдси	16.23

first_name	sales
Бритни	3.40
Бритни	2.58
Бритни	4.50
Бритни	11.25
Бритни	8.78
Бритни	43.21
Бритни	34.19

Побеждает... Бритни!

```
File Edit Window Help TheWinnerReallyIs
> SELECT first_name, SUM(sales)
-> FROM cookie_sales GROUP BY first_name
-> ORDER BY SUM(sales);
+-----+-----+
| first_name | sum(sales) |
+-----+-----+
| Бритни     |      107.91 |
| Пэрис      |       98.23 |
| Николь     |       96.03 |
| Линдси     |       81.08 |
+-----+-----+
4 rows in set (0.00 sec)
```

Функция AVG с GROUP BY

Другие девочки были огорчены, поэтому Эдвина решила вручить второй приз за высший средний объем продаж за день. Для его вычисления она использует функцию AVG.

Каждая девочка продавала печенье семь дней. Для каждой девочки функция AVG суммирует ее продажи, а затем делит их на 7.

И снова данные группируются по значению first_name...

... но на этот раз вычисляется не сумма, а среднее значение.

```
SELECT first_name, AVG(sales)
FROM cookie_sales
GROUP BY first_name;
```

AVG суммирует все значения группы и делит сумму на количество значений, чтобы определить среднее значение для группы.

first_name	sales
Николь	19.22
Николь	0.00
Николь	8.05
Николь	26.82
Николь	7.77
Николь	15.21
Николь	18.96

first_name	sales
Пэрис	26.53
Пэрис	0.00
Пэрис	0.56
Пэрис	1.52
Пэрис	13.44
Пэрис	24.19
Пэрис	31.99

first_name	sales
Линдси	17.62
Линдси	9.16
Линдси	0.00
Линдси	32.02
Линдси	2.34
Линдси	3.71
Линдси	16.23

first_name	sales
Бритни	3.40
Бритни	2.58
Бритни	4.50
Бритни	11.25
Бритни	8.78
Бритни	43.21
Бритни	34.19

И снова победила Бритни...
Нужно придумать другой способ определения второго места.

```
File Edit Window Help TheWinnerReallyIs
> SELECT first_name, AVG(sales)
-> FROM cookie_sales GROUP BY first_name;
+-----+-----+
| first_name | AVG(sales) |
+-----+-----+
| Николь     | 13.718571 |
| Бритни     | 15.415714 |
| Линдси     | 11.582857 |
| Пэрис      | 14.032857 |
+-----+-----+
4 rows in set (0.00 sec)
```


MIN и MAX

Не желая сдаваться, Эдвина применяет к своей таблице функции MIN и MAX. Она хочет узнать, не было ли у других девочек более высоких продаж за день – а может, в свой худший день Бритни заработала меньше других?

Для определения **наибольшего значения в столбце** используется функция MAX, а для **определения наименьшего значения** – функция MIN.

```
SELECT first_name, MAX(sales)
FROM cookie_sales
GROUP BY first_name;
```

MAX возвращает наибольшее значение из каждой группы.

Сюрприз! Самая большая выручка за день снова у Бритни.

first_name	sales
Николь	26.82
Бритни	43.21
Линдси	32.02
Пэрис	31.99

```
SELECT first_name, MIN(sales)
FROM cookie_sales
GROUP BY first_name;
```

MIN возвращает наименьшее значение из каждой группы.

Похоже, у всех остальных девочек был хотя бы один выходной, а у Бритни даже в худший день был заработок.

first name	sales
Николь	0.00
Бритни	2.58
Линдси	0.00
Пэрис	0.00

Это уже серьезно. Может, дать приз девочке, которая продавала печенье больше дней, чем другие?



COUNT и подсчет дней

Чтобы узнать, какая из девочек продавала печенье больше дней, чем другие, Эдвина пытается использовать для подсчета функцию COUNT. Функция COUNT возвращает *количество записей в столбце*.

```
SELECT COUNT(sale_date)
FROM cookie_sales;
```

Функция COUNT возвращает количество записей в столбце «sale_date». Если запись содержит NULL, она не включается в подсчет.

Возьми в руку карандаш



cookie_sales

ID	first_name	sales	sale_date
1	Линдси	32.02	6-3-2007
2	Пэрис	26.53	6-3-2007
3	Бритни	11.25	6-3-2007
4	Николь	18.96	6-3-2007
5	Линдси	9.16	7-3-2007
6	Пэрис	1.52	7-3-2007
7	Бритни	43.21	7-3-2007
8	Николь	8.05	7-3-2007
9	Линдси	17.62	8-3-2007
10	Пэрис	24.19	8-3-2007
11	Бритни	3.40	8-3-2007
12	Николь	15.21	8-3-2007
13	Линдси	0	9-3-2007
14	Пэрис	31.99	9-3-2007
15	Бритни	2.58	9-3-2007
16	Николь	0	9-3-2007
17	Линдси	2.34	10-3-2007
18	Пэрис	13.44	10-3-2007
19	Бритни	8.78	10-3-2007
20	Николь	26.82	10-3-2007
21	Линдси	3.71	11-3-2007
22	Пэрис	.56	11-3-2007
23	Бритни	34.19	11-3-2007
24	Николь	7.77	11-3-2007
25	Линдси	16.23	12-3-2007
26	Пэрис	0	12-3-2007
27	Бритни	4.50	12-3-2007
28	Николь	19.22	12-3-2007

Перед вами исходная таблица. Как вы думаете, какой результат вернет запрос?

Представляет ли это число количество дней, в течение которых продавалось печенье?

Напишите запрос, который будет возвращать количество дней, в течение которых каждая девочка продавала печенье.

Возьми в руку карандаш Решение

Перед вами исходная таблица. Как вы думаете, какой результат вернет запрос?

28 дней

Представляет ли это число количество дней, в течение которых продавалось печенье?

Нет. Оно представляет количество значений в столбце «sale_date».

Напишите запрос, который будет возвращать количество дней, в течение которых каждая девочка продавала печенье.

```
SELECT first_name, COUNT(sale_date)
FROM cookie_sales
GROUP BY first_name;
```



Чтобы узнать, сколько дней продавалось печенье, можно было упорядочить результат по sale_date, и вычесть из последней даты первую. Правильно?

Вообще-то нет. Мы не можем быть уверены в том, что между первой и последней датой не было пропущенных дней.

Существует гораздо более простой способ узнать, в течение сколько дней продавалось печенье. Задача решается при помощи ключевого слова DISTINCT. Оно поможет нам не только вычислить нужное значение COUNT, но и получить список дат, не содержащий дубликатов.

Команда **SELECT DISTINCT**

Для начала посмотрим, как работает ключевое слово **DISTINCT** без функции **COUNT**.

Так как **DISTINCT** — ключевое слово, а не функция, имя столбца «**sale_date**» не нужно заключать в круглые скобки.

```
SELECT DISTINCT sale_date
FROM cookie_sales
ORDER BY sale_date;
```

Условие **ORDER BY** упрощает поиск первой и последней даты продажи.

```
File Edit Window Help NoDupes
> SELECT DISTINCT sale_date
FROM cookie_sales
-> ORDER BY sale_date;
+-----+
| sale_date |
+-----+
| 2007-06-03 |
| 2007-07-03 |
| 2007-08-03 |
| 2007-09-03 |
| 2007-10-03 |
| 2007-11-03 |
| 2007-12-03 |
+-----+
7 rows in set (0.00 sec)
```

Смотрите:
ни одного
дубликата!

Теперь попробуем выполнить команду с функцией **COUNT**:

Обратите внимание: **DISTINCT** заключается в круглые скобки вместе с «**sale_date**».

```
SELECT COUNT(DISTINCT sale_date)
FROM cookie_sales;
```

Условие **ORDER BY** не нужно, потому что **COUNT** вернет одно число. Упорядочивать здесь нечего.



Попробуйте выполнить этот запрос.
Кто из девочек продавал печенье в течение большего количества дней?

Ответа: Бруши

кто я?

Компания функций и ключевых слов SQL, облаченных в маскарадные костюмы, развлекается игрой «Кто я?» Игрок дает подсказку, а остальные на основании сказанного им пытаются угадать, кого он изображает. Будем считать, что игроки всегда говорят правду о себе. Заполните пропуски справа именами одного или нескольких участников. Также для каждого участника укажите, является ли он функцией или ключевым словом.

Сегодняшние участники:

COUNT, DISTINCT, AVG, MIN, GROUP BY, SUM, MAX

Кто я?



Имя

функция или
ключевое слово

Мой результат не выглядит большим.

Мой результат больше любого из входных значений.

Мои результаты единственные и неповторимые.

Я скажу, сколько здесь было значений.

Используйте меня при вычислении суммы.

Меня интересуют только большие числа.

Как дела? Да так, средне.

Ответы на с. 309.



часть Задаваемые Вопросы

В: А могли ли мы добавить условие **ORDER BY** при поиске средних/наибольших/наименьших значений с использованием **AVG**, **MAX** и **MIN**?

О: Могли — более того, это была бы очень хорошая идея. Мы не стали включать **ORDER BY**, чтобы не усложнять запросы и упростить изучение новых функций. Вернитесь к этим функциям и представьте, к каким последствиям привело бы применение **ORDER BY**. Вы видите, как изменятся результаты?

В: Ключевое слово **DISTINCT** выглядит весьма полезным. Его можно использовать с любым столбцом?

О: Да, с любым. Оно особенно удобно, когда один столбец содержит одинаковые значения у нескольких

записей, а вы просто хотите просмотреть уникальные значения вместо длинного списка дубликатов.

В: Запрос с **MIN ()** никак не помог бы Эдвине определить победителя, верно?

О: Верно, но Эдвина смогла узнать, кто из девочек трудился хуже других. На следующий год она постарается обратить на лентяев особое внимание.

В: Раз уж мы заговорили о **MIN** — что произойдет, если в столбце встречаются **NULL**?

О: Хороший вопрос. Нет, ни одна из этих функций никогда не возвращает **NULL**, потому что **NULL** — это отсутствие значения, а не ноль.



Хм... **AVG**, **MAX** и **COUNT** так и не помогли мне определить второе место. Придется воспользоваться **SUM**, вычислить, кто из девочек оказался на втором месте по продажам, и наградить.

МОЗГОВОЙ ШТУРМ

Представьте, что в таблице хранятся данные не четырех, а сорока девочек. Как использовать **SUM** для определения второго места?

LIMIT и ограничение результатов

Итак, мы будем использовать SUM для определения второго места. Давайте вернемся к исходному запросу и результатам, чтобы понять, как получить нужную информацию.

```
SELECT first_name, SUM(sales)
FROM cookie_sales
GROUP BY first_name
ORDER BY SUM(sales) DESC;
```

Очень важно использовать здесь условие ORDER BY; в противном случае результаты будут следовать в случайном порядке.

first_name	sales
Бритни	107.91
Пэрис	98.23
Николь	96.03
Линдси	81.08

Нас интересуют только первые два результата.

Пэрис на втором месте! Николь с ней больше не разговаривает.

С четырьмя результатами нетрудно увидеть, кто оказался на втором месте. Но если вы хотите действовать еще точнее, ограничьте список данными двух девочек с наибольшими объемами продаж. Ключевое слово LIMIT позволяет указать количество записей, возвращаемых запросом из итогового набора.

```
SELECT first_name, SUM(sales)
FROM cookie_sales
GROUP BY first_name
ORDER BY SUM(sales) DESC
LIMIT 2;
```

Означает, что список ОГРАНИЧИВАЕТСЯ первыми двумя результатами.

Этот длинный запрос возвращает всего два числа.

first_name	sales
Бритни	107.91
Пэрис	98.23

В таблице хранятся данные всего четырех девочек, и ограничение их до двух особой пользы не принесет. Но представьте, что вы работаете с огромной таблицей. Допустим, в таблице хранятся описания 1000 самых популярных песен, а вы хотите отобразить из них первые 100 в порядке популярности. Условие LIMIT позволит получить только нужные вам песни, а остальные 900 останутся «за кадром».

LIMIT и Второе место

LIMIT даже позволит нам сразу определить второе место, без вывода первого места. Для этого LIMIT передаются два параметра:

Если вы попытаетесь угадать смысл этого условия, то скорее всего, ошибетесь. С двумя параметрами LIMIT работает совсем не так, как прежде.

LIMIT 0, 4

Номер начальной записи. В SQL нумерация начинается с 0.

Количество возвращаемых результатов.

first_name	sales
Бритни	107.91
Пэрис	98.23
Николь	96.03
Линдси	81.08

Бритни — 0,
Пэрис — 1,
Николь — 2,
Линдси — 3

Еще не забыли наш пример со 100 песнями? Допустим, мы хотим получить песни с 20 по 30. Это можно сделать при помощи LIMIT с дополнительным параметром. Упорядочите песни по популярности и добавьте условие LIMIT 19, 10. Параметр 19 указывает, что вывод начинается с 20 песни (раз в SQL нумерация начинается с 0), а параметр 10 — что запрос должен вернуть 10 записей.

Возьми в руку карандаш



Напишите запрос, который вернет второй результат — **и только второй результат**. В запросе должно использоваться условие LIMIT с двумя параметрами.

Возьми в руку карандаш Решение



Напишите запрос, который вернет второй результат — **и только второй результат**. В запросе должно использоваться условие LIMIT с двумя параметрами.

```
SELECT first_name, SUM(sales)
FROM cookie_sales
GROUP BY first_name
ORDER BY SUM(sales) DESC
LIMIT 1,1;
```

← Не забудьте, что в SQL нумерация начинается с 0. Так что 1 — это на самом деле 2.

С этими новыми ключевыми словами мои команды SQL стали такими длинными и сложными... Все это хорошо, конечно, но нельзя ли их как-нибудь упростить?



Ваши запросы становятся длиннее, потому что сами данные стали более сложными.

К таблице стоит присмотреться повнимательнее — возможно, она стала слишком сложной. Пора переходить к главе 7...

Компания функций и ключевых слов SQL, облаченных в маскарадные костюмы, развлекается игрой «Кто я?» Игрок дает подсказку, а остальные на основании сказанного им пытаются угадать, кого он изображает. Будем считать, что игроки всегда говорят правду о себе. Заполните пропуски справа именами одного или нескольких участников. Также для каждого участника укажите, является ли он функцией или ключевым словом.

Сегодняшние участники:

COUNT, DISTINCT, AVG, MIN, GROUP BY, SUM, MAX

Кто я?



см. с. 304

Имя	функция или ключевое слово
Мой результат не выглядит большим.	MIN функция
Мой результат больше любого из входных значений.	SUM функция
Мои результаты единственные и неповторимые.	DISTINCT ключевое слово
Я скажу, сколько здесь было значений.	COUNT функция
Используйте меня при вычислении суммы.	GROUP BY ключевое слово
Меня интересуют только большие числа.	MAX функция
Как дела? Да так, средне.	AVG функция



Новые инструменты

Глава 6 осталась позади. Теперь вы с легкостью управляетесь с функциями, ключевыми словами и расширенными запросами SELECT. Полный список инструментов приведен в приложении III.

ORDER BY

Упорядочивает результаты по заданному столбцу.

GROUP BY

Группирует записи по одинаковым значениям столбца.

DISTINCT

Возвращает только уникальные значения, без дубликатов.

SUM

Суммирует числовые значения в столбце.

LIMIT

Определяет, сколько именно записей должен вернуть запрос, и с какой записи следует начинать.

COUNT

Сообщает, сколько записей возвращает запрос SELECT, без вывода самих записей. COUNT возвращает одно целое число.

AVG

Возвращает среднее значение для числового столбца..

MAX и MIN

MAX возвращает наибольшее значение в столбце, а MIN — наименьшее.

Ваши новые инструменты: функции, ключевые слова и расширенные запросы SELECT!

7 Многотабличные базы данных

Когда в одной таблице тесно



Иногда в одной таблице становится попросту тесно. Данные стали более сложными, и с одной таблицей работать уже **неудобно**. Ваша единственная таблица забита избыточной информацией, которая только попусту расходует место и замедляет обработку запросов. Вы выжали из одной таблицы все, что только можно, но окружающий мир огромен, и для хранения данных и работы с ними нередко приходится использовать **несколько таблиц**.

Как найти Найджелу подружку

Ближайший друг Грега – Найджел – попросил ему подобрать подружку с похожими интересами. Для начала Грег извлекает из базы данных запись Найджела.

Вот как она выглядит:

```
contact_id: 341
last_name: Мур
first_name: Найджел
phone: 5552311111
email: nigelmoore@ranchersrule.com
gender: M
birthday: 1975-28-08
profession: Фермер
city: Остин
state: TX
status: Не женат
interests: животные, лошади, кино
seeking: Незамужняя женщина
```

Столбец `interests` не является атомарным; в нем хранится несколько однотипных информационных объектов. Грег обеспокоен: похоже, составить запрос будет непросто.

Грег включает просьбу Найджела в свой список текущих дел.

Найджел ↘



● Текущие дела

Написать запрос для Найджела: запрос должен выполнять поиск по столбцу `interests`. Что-то сложно, вроде нужно использовать **ИКЕ**, но на один раз сойдет...

Зачем что-то менять?

Грег решил не изменять столбец interests. Он предпочитает писать сложные запросы, потому что ему кажется, что это придется делать не так часто.

Грег использует поле даты рождения для поиска кандидатов, которые по возрасту отличаются от Найджела не более чем на 5 лет.

Возьми в руку карандаш



Допишите запрос Грега, чтобы он находил женщин, разделяющих все интересы Найджела. Укажите, что делает каждая строка кода.

```

SELECT * FROM my_contacts
WHERE gender = 'Ж'
AND status = 'Не замужем'
AND state='TX'
AND seeking LIKE '%Мужчина%'
AND birthday > '1970-28-08'
AND birthday < '1980-28-08'
AND interests LIKE .....
AND .....
AND .....
    
```

Возьми в руку карандаш Решение



Допишите запрос Грегa, чтобы он находил женщин, разделяющих все интересы Найджела. Укажите, что делает каждая строка кода.

Выбрать из таблицы my_contacts все записи, удовлетворяющие следующим условиям.

```

SELECT * FROM my_contacts
WHERE gender = 'Ж'
AND status = 'Не замужем'
AND state='TX'
AND seeking LIKE '%Мужчина%'
AND birthday > '1970-28-08'
AND birthday < '1980-28-08'
AND interests LIKE '%животные%'
AND interests LIKE '%лошади%'
AND interests LIKE '%кино%';

```

Ищем женщину...
 ...незамужнюю...
 ...чтобы жила в том же штате...
 ...и хотела познакомиться с мужчиной...
 Не более чем на 5 лет младше или старше Найджела...
 А здесь ищутся совпадения для увлечений Найджела. Также можно было использовать OR, но мы хотим найти совпадение по всем увлечениям.

Запрос прекрасно сработал...

Грег находит идеальную пару для Найджела:

```

contact_id: 1854
last_name: Фиоре
first_name: Карла
phone: 5557894855
email: cfioire@fioreanimalclinic.com
gender: Ж
birthday: 1974-01-07
profession: Ветеринар
city: Раунд-Рок
state: TX
status: Не замужем
interests: лошади, кино, животные, детективы, туризм
seeking: single M

```

подходит
 хорошая профессия
 и даже живет неподалеку
 интересы совпадают!

Карла и Триггер



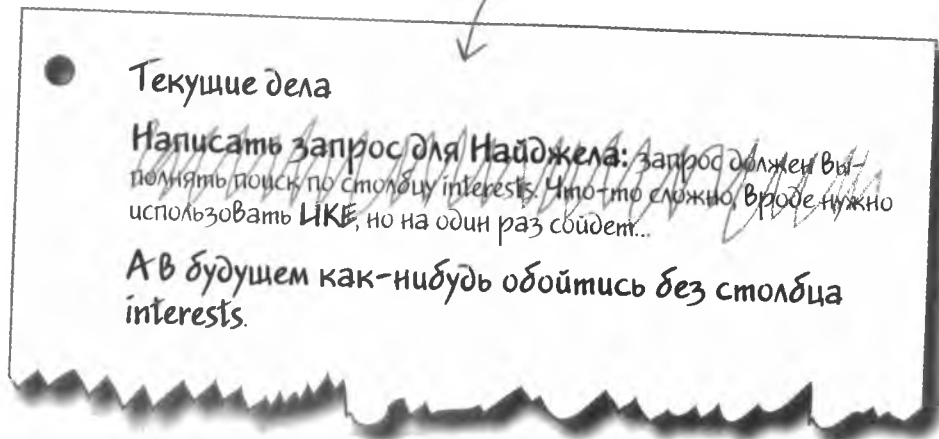
...даже слишком хорошо!

У Найджела и Карлы все срослось, и Грег стал жертвой собственного успеха: *все* неженатые друзья просят его найти им подругу жизни. А друзей у Грега много...



Всю «черную работу» должна выполнять база данных. Не пытайтесь обойти плохую структуру таблицы при помощи сложных запросов.

Написание запросов занимает слишком много времени. Грег включает в свой список дел новую запись.



Игнорировать проблему — не выход

Другой друг, Реджи, просит Грегa найти ему пару. Ему нужна женщина, которая отличается от него по возрасту не более чем на 5 лет. Реджи живет в Кембридже, штат Массачусетс, а его увлечения отличаются от увлечений Найджела.

Грег решает вообще не обращать внимания на столбец interests, чтобы не усложнять запросы.



Реджи →



Упражнение

Напишите для Реджи запрос, не использующий столбец interests.

```
contact_id: 873
last_name: Салливан
first_name: Реджи
phone: 5552311122
email: me@kathieleeisaflake.com
gender: M
birthday: 1955-20-03
profession: Комик
city: Кембридж
state: MA
status: Не женат
interests: животные, коллекционные карточки, геописк
seeking: Женщина
```

→ Ответ на с. 312.

Слишком много лишних вариантов

Грег отдает Реджи длинный список вариантов. Несколько недель спустя Реджи звонит Грегу и говорит, что от его списка нет никакого проку: ни одна из кандидатур не имеет с ним ничего общего.



Нельзя полностью игнорировать увлечения. Должен быть другой, лучший способ...

● Текущие дела

~~Написать запрос для Найджела: запрос должен выполнять поиск по столбцу interests. Что-то сложно, вроде нужно использовать LIKE, но на один раз сойдет...~~

~~А в будущем как-нибудь обойтись без столбца interests.~~

Проверять только первое увлечение, а на остальные не обращать внимания.

Увлечения ВАЖНЫ! Их нельзя игнорировать, это ценная информация.


Использовать только первое увлечение

Теперь Грег знает, что игнорировать все увлечения нельзя. Он предполагает, что люди перечисляют увлечения в порядке важности, и решает, что он будет проверять только первое из них. Запросы по-прежнему остаются сложными, но не настолько, как при включении LIKE для всех увлечений из столбца interests.

Возьми в руку карандаш



Используйте функцию `SUBSTRING_INDEX` для выделения первого увлечения из столбца interests.

 Возьми в руку карандаш
Решение

Используйте функцию SUBSTRING_INDEX для выделения первого увлечения из столбца interests.

```
SUBSTRING_INDEX(interests, ',', 1)
```

Вызов функции выделяет все символы, предшествующие запятой в столбце «interests».

Искомый символ — запятая.

«1» для поиска первой запятой. С параметром «2» функция выдела бы все символы до второй запятой, то есть текст первых двух увлечений.

Затем Грег пишет запрос, который поможет Реджи найти свою пару. В запросе используется функция SUBSTRING_INDEX, а первым увлечением должны быть 'животные'.

```
SELECT * FROM my_contacts
WHERE gender = 'Ж'
AND status = 'Не замужем'
AND state='MA'
AND seeking LIKE '%Мужчина%'
AND birthday > '1950-28-08'
AND birthday < '1960-28-08'
AND SUBSTRING_INDEX(interests, ',', 1) = 'животные';
```

В запросах будут отображаться только женщины, у которых в списке увлечений на первом месте стоят 'животные'.

Пара для Реджи

Наконец-то! Грег нашел пару для Реджи:

```
contact_id: 459
last_name: Фергюсон
first_name: Алексис
phone: 5550983476
email: alexangel@yahoo.com
gender: Ж
birthday: 1956-19-09 ← подходит
profession: Художник
city: Флиггервилль
state: MA ← живет близко
status: Не замужем
interests: животные ← подходящие увлечения
seeking: Мужчина
```

Трагическое несоответствие

Реджи договорился с Алексис о свидании, и Грег с нетерпением ждал его рассказа. Он уже начал представлять себе новую таблицу `my_contacts`, которая станет началом новой социальной сети.

На следующий день у двери Грега стоит Реджи — и притом очень сердитый.

Реджи кричит: «Конечно, она интересуется животными. Но ты не сказал мне, что она делает из них чучела! Там повсюду мертвые животные!»

● Текущие дела

Написать запрос для Найджела: запрос должен выполнять поиск по столбцу `interests`. Что-то сложное, вроде нужно использовать `LIKE`, но на один раз сойдет...

А в будущем как-нибудь обойтись без столбца `interests`.

Проверять только первое увлечение, а на остальные не обращать внимания.

Создать несколько столбцов для хранения увлечений, потому что хранение всех увлечений в одном столбце усложняет запросы.

В таблице была идеальная пара для Реджи, но Грег не нашел ее, потому что ее увлечения перечислялись в другом порядке.

Грег решает изменить структуру своей таблицы.

МОЗГОВОЙ ШТУРМ

Как будет выглядеть следующий запрос Грега после создания нескольких столбцов увлечений?

Создание новых столбцов interest

Грег понимает, что с одним столбцом увлечений написать правильный запрос слишком сложно. Приходится использовать LIKE, что иногда приводит к неверным совпадениям.

Но Грег умеет пользоваться командой ALTER для изменения таблиц, а также разбивать текстовые строки, поэтому он решает создать несколько столбцов с увлечениями и поместить каждое увлечение в отдельный столбец. Он решает, что четырех столбцов будет достаточно.

Возьми в руку карандаш



Используя команду ALTER и функцию SUBSTRING_INDEX, измените таблицу так, чтобы таблица состояла из перечисленных столбцов. Количество запросов не ограничивается.

```
contact_id
last_name
first_name
phone
email
gender
birthday
profession
city
state
status
interest1
interest2
interest3
interest4
seeking
```

→ Ответы на с. 311.

Начинаем заново

Грег чувствует себя виноватым за неудачу Реджи и решает попробовать еще раз. Для начала он извлекает из таблицы запись Реджи:

```
contact_id: 872
last_name: Салливан
first_name: Реджи
phone: 5554531122
email: regis@kathieleeisaflake.com
gender: Ж
birthday: 1955-20-03
profession: Комик
city: Кембридж
state: MA
status: Не женат
interest1: животные
interest2: коллекционные карточки
interest3: геописк
interest4: NULL
seeking: Женщина
```

} В измененной таблице информация об увлечениях хранится в четырех столбцах.



Упражнение

Грег пишет запрос, который должен вернуть Реджи подходящую пару. Он начинает с простых столбцов — gender, status, state, seeking и birthday — и только потом берется за столбцы interest.

Запишите его запрос.



Грег пишет запрос, который должен вернуть Реджи подходящую пару. Он начинает с простых столбцов — gender, status, state, seeking и birthday — и только потом берется за столбцы interest.

Запишите его запрос.

```
SELECT * FROM my_contacts
```

```
WHERE gender = 'Ж'
AND status = 'Не замужем'
AND state='MA'
AND seeking LIKE '%Мужчина%'
AND birthday > '1950-20-03'
AND birthday < '1960-20-03'
AND
```

```
(
interest1 = 'животные'
OR interest2 = 'животные'
OR interest3 = 'животные'
OR interest4 = 'животные'
)
```

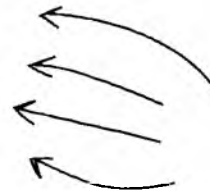
```
AND
```

```
(
interest1 = 'коллекционные карточки'
OR interest2 = 'коллекционные карточки'
OR interest3 = 'коллекционные карточки'
OR interest4 = 'коллекционные карточки'
)
```

```
AND
```

```
(
interest1 = 'геонюск'
OR interest2 = 'геонюск'
OR interest3 = 'геонюск'
OR interest4 = 'геонюск'
);
```

Реджи ищет незамужнюю женщину, родившуюся в заданный период времени, которая живет в Массачусетсе и хочет встречаться с неженатым мужчиной.



Чтобы найти совпадения с увлечениями Реджи, Грег вынужден проверить все четыре новых столбца «interest», потому что в каждом из них может найтись совпадение.

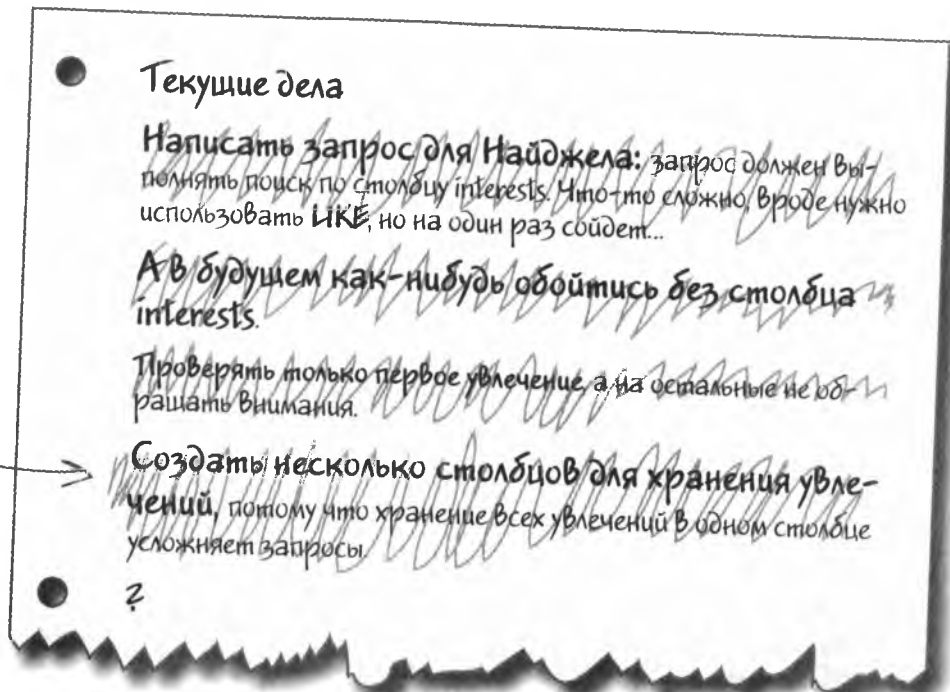


Столбец «interest4» у Реджи содержит NULL, поэтому проверяются только три увлечения вместо четырех.

Все без толку...

Добавление новых столбцов никак не помогло решить основную проблему: структура таблицы усложняет написание запросов к ней. Обратите внимание: в каждой версии таблицы нарушается правило атомарности данных.

Казалось бы, такое хорошее решение... Но с ним запросы стали еще сложнее.



...Один момент!



o o

А если создать отдельную таблицу, в которой хранится только информация об увлечениях?

МОЗГОВОЙ ШТУРМ

Какую пользу принесет создание новой таблицы?
И как связать данные из новой таблицы с существующей таблицей?

Одной таблицы недостаточно

Итак, если мы будем ограничиваться работой с текущей таблицей, хорошего решения не существует. Мы пытались обойти недостатки структуры данных разными способами, даже изменяя структуру всей таблицы. Ни один способ не сработал.

Рамки одной таблицы оказались слишком узкими. В действительности нам нужны **дополнительные таблицы**, которые *работают в сочетании* с текущей таблицей, позволяя нам связать **одного человека с несколькими увлечениями**. При этом существующие данные будут полностью сохранены.

Неатомарные столбцы из существующей таблицы следует переместить в новые таблицы.

```
File Edit Window Help MessyTable
> DESCRIBE my_contacts;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| contact_id | int(11)    | NO   | PRI | NULL     | auto_increment
| last_name  | varchar(30)| YES  |     | NULL     |
| first_name | varchar(20)| YES  |     | NULL     |
| phone      | varchar(10)| YES  |     | NULL     |
| email      | varchar(50)| YES  |     | NULL     |
| gender     | char(1)   | YES  |     | NULL     |
| birthday  | date      | YES  |     | NULL     |
| profession | varchar(50)| YES  |     | NULL     |
| city       | varchar(50)| YES  |     | NULL     |
| state      | varchar(2) | YES  |     | NULL     |
| status     | varchar(20)| YES  |     | NULL     |
| interests  | varchar(100)| YES  |     | NULL     |
| seeking    | varchar(100)| YES  |     | NULL     |
+-----+-----+-----+-----+-----+-----+
13 rows in set (0.01 sec) >
```

Многотабличная база данных с информацией о клоунах

Помните нашу таблицу с информацией о клоунах из главы 3? Проблем с клоунами становится все больше, поэтому мы преобразовали одну таблицу в более удобный набор из нескольких таблиц.

Так выглядела старая таблица clown_tracking.

clown_tracking

clown_info	name	last seen	activities
Элси	Дом престарелых Черри Хилл	Ж, рыжие волосы, зеленый костюм, огромные ботинки	шарики, машинки
Пиклз	Вечеринка Джека Грина	оранжевые волосы, синий костюм, огромные ботинки	мим
Снаглз	Болмарт	желтая рубашка, красные штаны	рожок, зонтик

То, что раньше было главной таблицей, теперь выглядит так.

clown_info
id
name
gender
description

info_activities
id
activity_id

activities
activity_id
activity

info_location
id
location_id
when

location
location_id
location

Скоро вы поймете, что означают эти линии со стрелками...

На нескольких ближайших страницах мы покажем, почему таблица была разбита именно так, а не иначе и что означают все эти стрелки и ключи. А после этого вы сможете по тем же принципам разбить таблицу Грега.



Как вы думаете, что означают линии со стрелками? А изображения ключей?

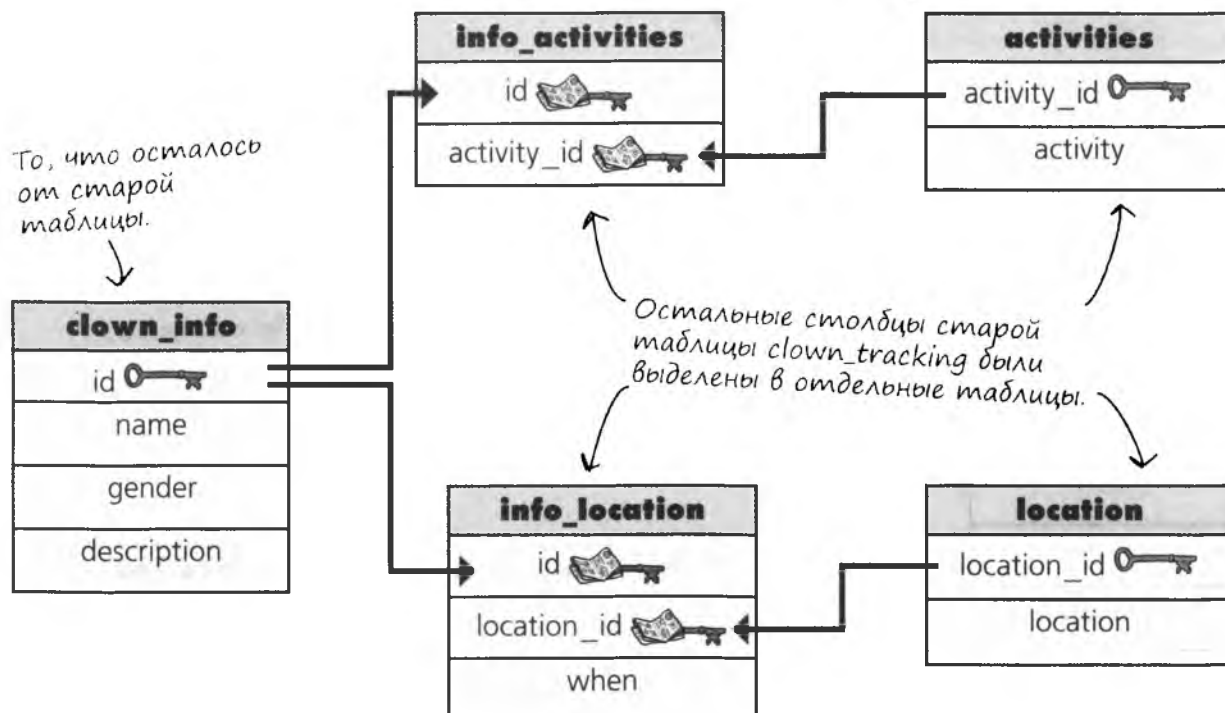
Схема базы данных clown_tracking

Представление всех структур базы данных (таблиц, столбцов и т. д.) и логических связей между ними называется **схемой**.

Наглядное представление базы данных поможет вам представить, как связаны между собой компоненты базы данных, однако схема также может быть записана и в виде текста.

Старая таблица.

clown_tracking			
clown_info	name	last seen	activities
Эпси	Дом престарелых Черри Хилл	Ж, рыжие волосы, зеленый костюм, огромные ботинки	шарики, машинки
Пиклз	Вечеринка Джека Грина	М, М, оранжевые волосы, синий костюм, огромные ботинки	мим
Снаглз	Болмарг	Ж, кашка, красные штаны	рожок, зонтик
Мистер Хоббс	Вечеринка Эрика Грея	Ж, Black hair, tiny hat	сиренка



Описание данных (столбцов и таблиц) вашей базы данных, включая все взаимосвязанные объекты и связи между ними, называется СХЕМОЙ.

Упрощенное представление таблиц

Вы видели, как была преобразована таблица с информацией о клоунах. Теперь давайте попробуем сделать то же самое с таблицей `my_contacts`.

До настоящего момента мы либо схематично изображали таблицы с именами столбцов в заголовках и данными внизу, либо выводили их описание в окне терминала командой `DESCRIBE`. Оба способа хорошо подходят для отдельных таблиц, но когда требуется построить диаграмму из нескольких таблиц, приходится искать что-то другое.

Ниже показано упрощенное представление таблицы `my_contacts`.

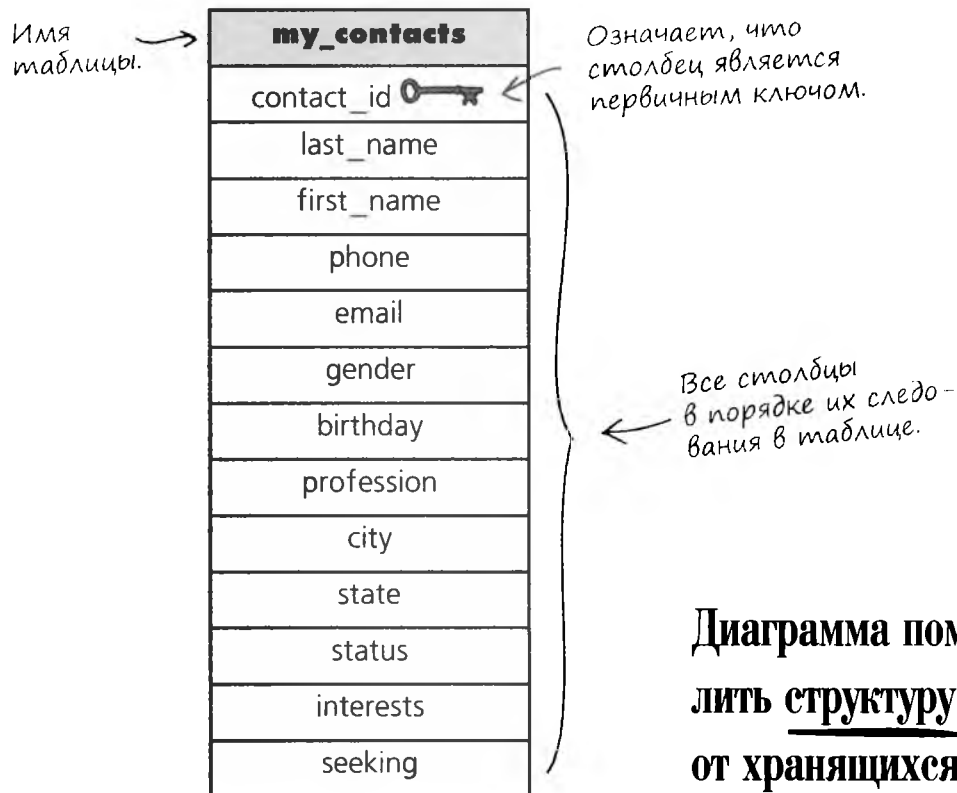


Диаграмма помогает отделить структуру таблицы от хранящихся в ней данных.

Как из одной таблицы сделать две

Мы знаем, что написать запрос для поиска информации в столбце `interests` в его текущем виде довольно затруднительно, потому что в одном столбце могут храниться сразу несколько значений. Впрочем, создание нескольких отдельных столбцов не особенно упростило нашу задачу.

Справа изображена таблица `my_contacts` в ее текущем состоянии. Столбец `interests` не атомарен, и существует только один действительно хороший способ сделать его атомарным: нам понадобится новая таблица, в которой будут храниться все увлечения.

Для начала нарисуем несколько диаграмм, которые покажут, как будут выглядеть новые таблицы. Только после того как будет готова новая схема, можно будет переходить к созданию новых таблиц или модификации данных.

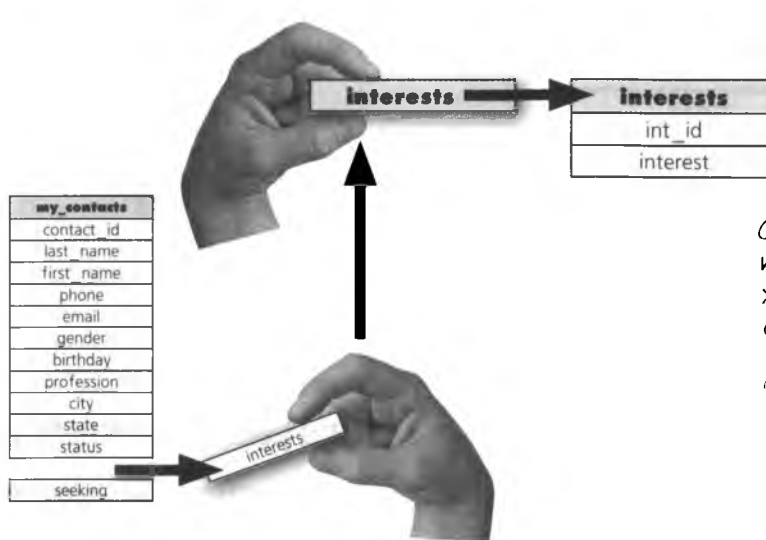
my_contacts
contact_id
last_name
first_name
phone
email
gender
birthday
profession
city
state
status
interests
seeking

Таблица `my_contacts` еще не атомарна

1

Удаляем столбец `interests` и размещаем его в отдельной таблице.

Столбец `interests` перемещается в новую таблицу.



Добавление поля `id` гарантирует, что в таблице не будет дубликатов.

Столбец «`interest`» типа `VARCHAR` содержит фактическое описание увлечения. В нем хранятся строки вида «туризм» или «кулинария».

В новой таблице `interests` будут храниться все увлечения из таблицы `my_contacts` (отдельная запись для каждого увлечения).

2

Добавляем столбцы, по которым можно будет узнать, какие увлечения принадлежат тому или иному человеку из таблицы my_contacts.

Мы вынесли увлечения из таблицы my_contacts, но как определить, кому какие увлечения принадлежат. Необходимо использовать информацию из таблицы my_contacts и разместить ее в таблице interests так, чтобы эти две таблицы были связаны между собой.

Например, для этого можно включить столбцы first_name и last_name в таблицу interests.

my_contacts
contact_id
last_name
first_name
phone
email
gender
birthday
profession
city
state
status
seeking

interests
int_id
interest
first_name
last_name

По этим двум столбцам можно узнать, кто какими увлечениями обладает.

Если эти значения совпадают, значит, человек обладает данным увлечением. Таблица interests содержит несколько записей с совпадением этих значений, что позволяет связать с одним человеком сразу несколько увлечений.

МОЗГОВОЙ ШТУРМ

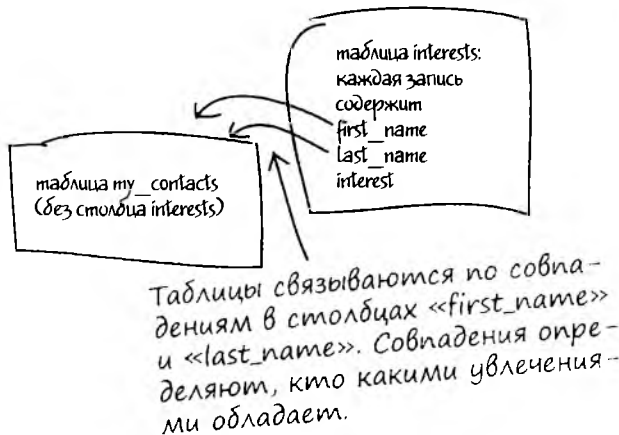
Мы движемся в верном направлении, но first_name и last_name — не лучшие столбцы для связывания таблиц.

Почему?

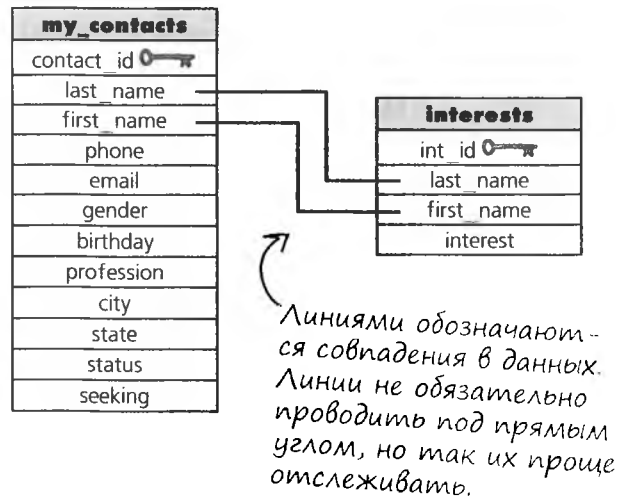
Связывание таблиц на диаграммах

К таблице `my_contacts` стоит присмотреться повнимательнее.

Вот ее исходный вариант.



А вот как выглядит новая схема.



Обратите внимание на линии между таблицами: они обозначают столбцы с совпадающими значениями. Диаграмма, представленная в таком виде, будет понятна для любого SQL-разработчика, потому что в ней используются стандартные обозначения.

А вот как выглядит серия команд `SELECT`, которая позволит нам использовать данные из обеих таблиц.

```
1 SELECT first_name, last_name  
FROM my_contacts  
WHERE (условия);
```

```
2 SELECT interest FROM interests  
WHERE first_name = 'Имя'  
AND last_name = 'Фамилия';
```

Вам кажется, что эта запись неэффективна? И правильно. Она всего лишь показывает, как использовать данные одной таблицы для извлечения данных из другой таблицы. (Вскоре мы покажем, как сделать то же самое более эффективно.)

Возьми в руку карандаш



Какие еще таблицы стоит добавить в базу данных `gregs_list` для хранения информации о нескольких увлечениях?

Не старайтесь нарисовать аккуратную схему; сейчас время собирать идеи. Одна идея уже изображена на рисунке, но у нее есть недостаток.



Таблицы связываются по совпадениям в столбцах «`first_name`» и «`last_name`». Совпадения определяют, кто какими увлечениями обладает.

Возьми в руку карандаш Решение



Какие еще таблицы стоит добавить в базу данных `gregs_list` для хранения информации о нескольких увлечениях?

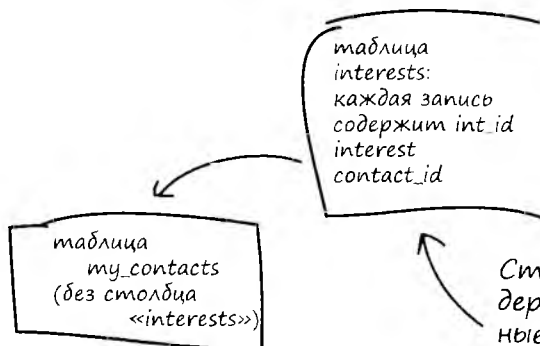
Не старайтесь нарисовать аккуратную схему; сейчас время собирать идеи. Одна идея уже изображена на рисунке, но у нее есть недостаток.



Однако связывание таблиц по имени и фамилии — не самый лучший способ. В списке `my_contacts` могут оказаться данные людей с одинаковым именем и фамилией, в результате чего может возникнуть путаница с увлечениями. Лучше использовать для связывания таблиц первичный ключ.

Таблицы связываются по совпадениям в столбцах «`first_name`» и «`last_name`». Совпадения определяют, кто какими увлечениями обладает.

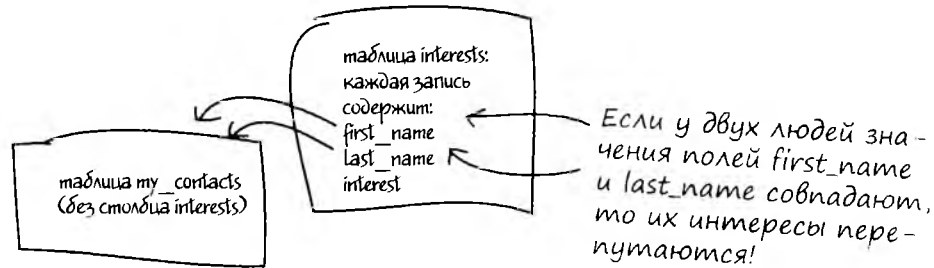
Вместо того, чтобы использовать «`first_name`» и «`last_name`», которые могут оказаться не уникальными, для связывания таблиц лучше взять «`contact_id`»:



Столбец «`contact_id`» содержит заведомо уникальные значения. Мы можем быть твердо уверены в том, что увлечения с некоторым «`contact_id`» принадлежат соответствующей записи из таблицы «`my_contacts`».

Связывание таблиц

У первой версии связанных таблиц был один серьезный недостаток: мы пытались использовать для связывания поля `first_name` и `last_name`. А если в таблице `my_contacts` появятся записи с одинаковыми значениями `first_name` и `last_name`?



Две таблицы должны связываться через **уникальный** столбец. К счастью, поскольку мы уже занялись нормализацией, в `my_contacts` такой столбец уже имеется: это **первичный ключ**.

Мы можем хранить значения первичного ключа из таблицы `my_contacts` в таблице `interests`. И что еще лучше, по этому столбцу можно будет определить, какие увлечения принадлежат тому или иному человеку из таблицы `my_contacts`. Такой способ связывания называется **внешним ключом**.



ВНЕШНИЙ КЛЮЧ — столбец таблицы, в котором хранятся значения **ПЕРВИЧНОГО КЛЮЧА** другой таблицы.

my_contacts	
contact_id	🔑
last_name	
first_name	
phone	
email	
gender	
birthday	
profession	
city	
state	
status	
seeking	

Чтобы новая таблица соответствовала правилам первой нормальной формы, каждой записи назначается уникальное значение первичного ключа.

interests	
int_id	🔑
interest	
contact_id	🔑

ВНЕШНИЙ КЛЮЧ определяет, какие увлечения принадлежат тому или иному человеку из таблицы `my_contacts`.

Что нужно знать о Внешних ключах




Имя внешнего ключа может отличаться от имени первичного ключа, с которым он связывается.

Первичный ключ, используемый внешним ключом, также называется *родительским ключом*. Таблица, которой принадлежит первичный ключ, называется *родительской таблицей*.

Внешний ключ может использоваться для установления соответствия между записями двух таблиц.

Внешний ключ может содержать значения NULL, хотя в первичном ключе они запрещены.

Значения внешнего ключа не обязаны быть уникальными — более того, чаще они уникальными не являются.



Понимаю, внешний ключ позволит мне связать две таблицы. Но какой прок от значений NULL во внешнем ключе? Можно ли сделать так, чтобы внешний ключ всегда был связан с родительским ключом?

Значение NULL во внешнем ключе означает, что в родительской таблице не существует соответствующего значения первичного ключа.

Однако мы можем сделать так, чтобы внешний ключ принимал только осмысленные значения, существующие в родительской таблице. Для этого следует воспользоваться **ограничением**.

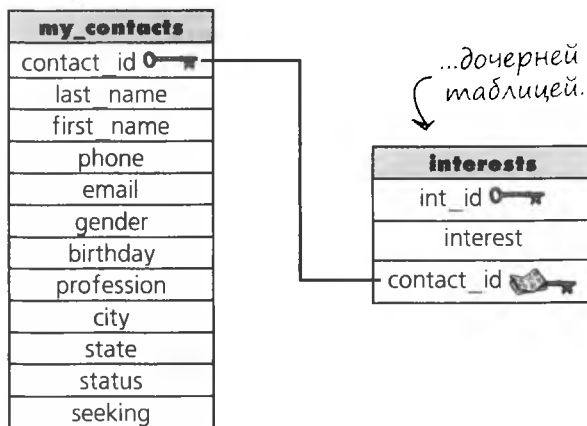
Ограничение Внешнего ключа

Хотя вы можете создать таблицу со столбцом, который будет выполнять функции внешнего ключа, такой столбец действительно станет внешним ключом только в том случае, если вы назначите его таковым в команде CREATE или ALTER. Ключ создается в структуре, называемой ограничением.

Ограничение — это своего рода правило, которое должно выполняться таблицей.

При вставке внешний ключ будет принимать только значения, существующие в первичном ключе родительской таблицы. Это требование называется целостностью данных.

Исходная таблица my_contacts стала родительской таблицей, потому что часть ее данных была перемещена в новую таблицу, называемую...



Термин «целостность данных» означает, что во внешнем ключе дочерней таблицы могут сохраняться только те значения, которые уже существуют в родительской таблице.

Внешний ключ должен быть связан с уникальным значением из родительской таблицы.

Это значение может и не быть значением первичного ключа, но оно обязательно должно быть **уникальным**.

Создание ВНЕШНЕГО КЛЮЧА как ограничения таблицы дает определенные преимущества.

При попытке нарушения правила вы получите сообщение об ошибке; таким образом предотвращаются случайные нарушения связей между таблицами.

Стоит ли возиться с внешними ключами?



Итак, упростить получение информации об увлечениях можно только одним способом: убрав их из таблицы `my_contacts`. И Реджи нужно подобрать нормальную пару... Осталось понять, КАК создать таблицу с внешним ключом.

Внешний ключ можно назначить при создании таблицы.

Внешние ключи также можно добавлять при выполнении команды `ALTER TABLE`. Синтаксис прост, но вы должны знать имя первичного ключа в родительской таблице, а также имя родительской таблицы. Давайте создадим таблицу `interests` с внешним ключом `contact_id` из таблицы `my_contacts`.

Часто задаваемые вопросы

В: Как написать запрос на выборку увлечений после того, как они будут извлечены из `my_contacts`?

О: Этим мы займемся в следующей главе. И вы увидите, что написать запрос на выборку данных из нескольких таблиц не так уж сложно. А пока необходимо изменить структуру `my_contacts`, чтобы запросы были простыми и эффективными.

Создание таблицы с внешним ключом

Теперь вы знаете, зачем создаются внешние ключи, и мы можем перейти непосредственно к способу их создания. Обратите внимание на имя, назначенное ограничению (CONSTRAINT): по нему можно легко определить, из какой таблицы берется ключ.

Включение команды PRIMARY KEY в строку с определением — другой (более быстрый) способ назначения первичного ключа.

```
CREATE TABLE interests (
  int_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  interest VARCHAR(50) NOT NULL,
  contact_id INT NOT NULL,
  CONSTRAINT my_contacts_contact_id_fk
  FOREIGN KEY (contact_id)
  REFERENCES my_contacts (contact_id);
```

Внешний ключ создается точно так же, как любой индексный столбец: с типом данных INT и условием NOT NULL.

Ограничению присваивается имя, по которому можно определить, из какой таблицы взят ключ (my_contacts), как он называется (contact_id) и что ключ является внешним (fk).

Если позднее мы захотим изменить свое решение, то используем это имя. Строго говоря, эта строка не обязательна, но ее рекомендуется включать в команду.

↑
Указывает, из какой таблицы взят внешний ключ...

↑
...и как он назывался в этой таблице.

В скобках указывается имя внешнего ключа. Вы можете назвать его так, как сочтете нужным.



Упражнение

А теперь попробуйте сами. Откройте окно консоли и введите приведенный выше код создания таблицы interests.

Когда таблица будет создана, просмотрите описание ее структуры. Какая новая информация в описании сообщает о наличии ограничения?



А теперь попробуйте сами. Откройте окно консоли и введите приведенный выше код создания таблицы `interests`.

Когда таблица будет создана, просмотрите описание ее структуры. Какая новая информация в описании сообщает о наличии ограничения?

```
File Edit Window Help
> DESC interests;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| int_id     | int(11)       | NO   | PRI | NULL    | auto_increment |
| interest   | varchar(50)   | NO   |     |         |                |
| contact_id | int(11)       | NO   | MUL |         |                |
+-----+-----+-----+-----+-----+-----+

```

«MUL» означает, что одно значение может храниться в столбце в нескольких экземплярах. Этот факт позволяет нам хранить несколько увлечений для каждого значения `contact_id` из таблицы `my_contacts`.

Часто задаваемые вопросы

В: Зачем столько хлопот с созданием ограничения внешнего ключа? Разве нельзя использовать ключ из другой таблицы в качестве внешнего ключа без создания ограничения?

О: Можно, но при создании ограничения в таблицу будут вставляться только значения, уже существующие в родительской таблице. Ограничение гарантирует корректность связи между таблицами.

В: «Гарантирует корректность связи»? Что это значит?

О: Ограничение внешнего ключа обеспечивает целостность данных (иначе говоря, оно следит за тем, чтобы запись с внешним ключом в одной таблице

всегда имела соответствующую запись в другой таблице). Если вы попытаетесь удалить запись в таблице с первичным ключом или изменить значение первичного ключа, задействованного в ограничении внешнего ключа другой таблицы, будет выдано сообщение об ошибке.

В: Выходит, что я никогда не смогу удалить из `my_contacts` запись с первичным ключом, который присутствует в таблице `interests` в качестве внешнего ключа?

О: Сможете, но сначала придется удалить запись внешнего ключа. В конце концов, если вы удаляете запись из `my_contacts`, знать увлечения этого человека вам уже не обязательно.

В: А почему нельзя просто оставить эти записи в таблице `interests`?

О: Они снижают эффективность работы с данными. Со временем такие записи накапливаются, и обработка запросов замедляется из-за необходимости поиска в бесполезной информации.

В: Ладно, убедили. Какие еще бывают ограничения?

О: Вы уже видели ограничение первичного ключа. Ключевое слово `UNIQUE` (при создании столбца) тоже считается ограничением. Также существует ограничение `CHECK`, не поддерживаемое в MySQL. В нем можно задать условие, которое должно выполняться для вставки значения в столбец. За дополнительной информацией о `CHECK` обращайтесь к документации своей РСУБД.

Связи между таблицами

Итак, вы знаете, как связать таблицы через внешний ключ, но мы по-прежнему должны разобраться в сути связей между таблицами. В таблице `my_contacts` проблема заключается в том, что **многих людей** нужно связать с **многими интересами**.

Это один из трех возможных типов связей, которые постоянно встречаются при работе с данными: «один-к-одному», «один-ко-многим» и «многие-ко-многим». Когда вы знаете, к какому типу относятся ваши данные, разработка структуры из нескольких таблиц (то есть **схемы**) становится достаточно простым делом.

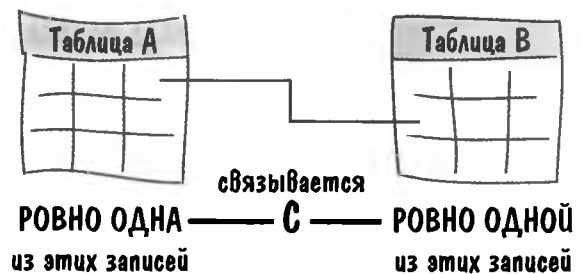
Типы связей: «один-к-одному»

Начнем с первого типа, «один-к-одному», и посмотрим, как он применяется на практике. В связях этого типа запись из таблицы А может быть связана НЕ БОЛЕЕ ЧЕМ С ОДНОЙ записью в таблице В.

Допустим, в таблице А хранится ваше имя, а в таблице В — информация о доходах и номера социального страхования (такая *изоляция* повышает безопасность данных).

В обеих таблицах присутствует поле `employee_id`. Поле `employee_id` родительской таблицы является первичным ключом, а поле `employee_id` дочерней таблицы — внешним ключом.

В схеме такая связь обозначается *простой* соединительной линией.



У каждого человека в таблице `employees` может быть только один номер социального страхования, а каждый номер может принадлежать только одному человеку. Следовательно, данная связь относится к типу «один-к-одному».

employees			salary		
employee_id	first_name	last_name	ssn	salary_level	employee_id
1	Бейонс	Ноулз	234567891	2	6
2	Шон	Картер	345678912	5	35
3	Шакира	Риполл	123456789	7	1

Эти таблицы тоже связаны отношением «один-к-одному», так как первичный ключ таблицы `employee` (`employee_id`) используется в качестве внешнего ключа таблицы `salary`.

Когда используются таблицы со связями типа «один-к-одному»



Выходит, все данные со связями «один-к-одному» следует выделять в новые таблицы?

Вообще-то нет. Таблицы со связями «один-к-одному» используются не так уж часто.

Есть несколько причин для установления связей типа «один-к-одному» между таблицами.

Когда используются связи типа «один-к-одному»

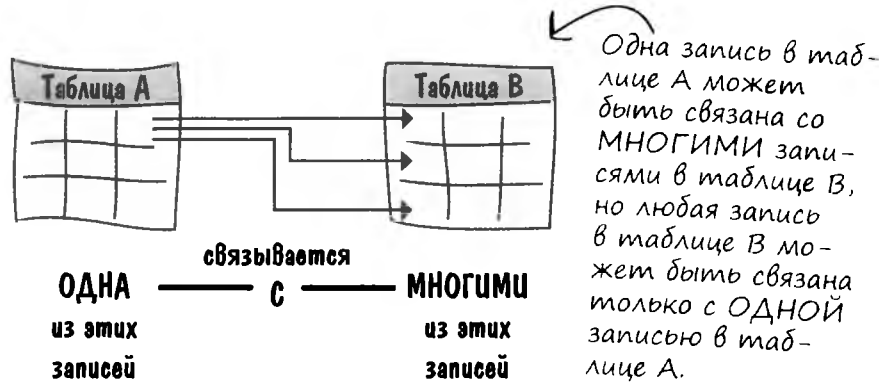
Обычно данные, связанные по типу «один-к-одному», разумнее хранить в основной таблице, однако выделение их в отдельную таблицу иногда приносит некоторые преимущества.

1. Выделение данных может ускорить обработку запросов. Например, если подавляющее большинство запросов извлекает только номер социального страхования и ничего более, лучше обращаться с запросом к меньшей таблице.
2. Если столбец может содержать неизвестные на данный момент значения, выделение его в отдельную таблицу позволит избежать хранения NULL в основной таблице.
3. Изоляция части данных помогает ограничить доступ к ним. Например, если у вас имеется таблица с записями работников, информацию о доходах лучше хранить отдельно от основной таблицы.
4. Большие блоки данных (например, тип BLOB) тоже лучше хранить в отдельной таблице.

«Один-к-одному»: ровно одна запись родительской таблицы связывается с одной записью дочерней таблицы.

Типы связей: «один-ко-многим»

В связях типа «один-ко-многим» запись в таблице может быть связана со **многими** записями в таблице В, но каждая запись в таблице В может быть связана только с **одной** записью в таблице А.



«Один-ко-многим»:
запись в таблице А может быть связана с **МНОГИМИ** записями в таблице В, но запись в таблице В может быть связана только с **ОДНОЙ** записью в таблице А.

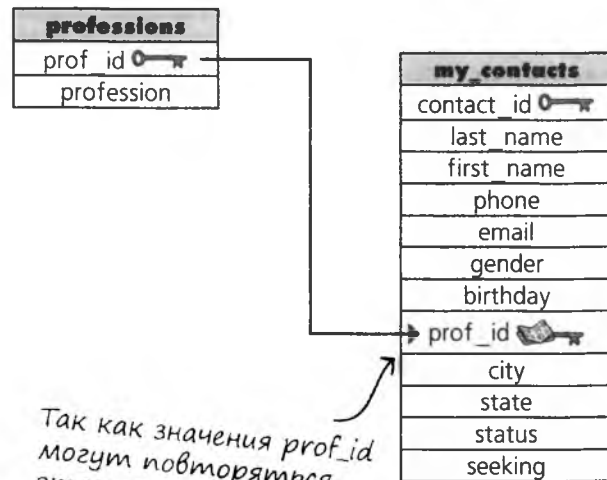
Данные о профессии в таблице `my_contacts` являются хорошим примером связей типа «один-ко-многим». Человек всегда имеет только одну профессию, но несколько человек из таблицы `my_contacts` могут иметь одинаковые профессии.

В этом примере мы вынесли столбец `profession` в новую дочернюю таблицу и заменили столбец `profession` внешним ключом `prof_id`. Для связывания таблиц используется столбец `prof_id`, присутствующий в обеих таблицах.

Соединительная линия помечена **треугольной стрелкой** на одном из концов; это означает, что **одна** запись связывается со **многими** записями.

Каждая запись таблицы `professions` может быть связана со многими записями `my_contacts`, но каждая запись `my_contacts` всегда связана только с одной записью в таблице `professions`.

Например, значение `prof_id` для профессии «Программист» может встретиться в `my_contacts` несколько раз, но у каждого человека в таблице `my_contacts` может быть указан только один код `prof_id`.

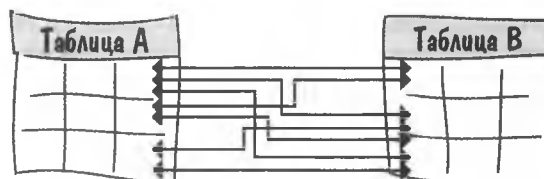


Так как значения `prof_id` могут повторяться, этот столбец не может быть первичным ключом. Это внешний ключ, связанный с ключом другой таблицы.

Типы связей: «многие-ко-многим»

Многие женщины держат в своем гардеробе **много** пар обуви. Если мы создаем две таблицы с информацией о женщинах и марках обуви, то между этими двумя таблицами будет существовать связь типа «многие-ко-многим», потому что обувь некоторого типа может принадлежать многим женщинам.

Предположим, Кэрри и Миранда купили шлепанцы и туфли «Прада», у Саманты и Миранды есть сандалии, а у Шарлотты есть вся эта обувь. Связь между таблицами women и shoes будет выглядеть так.



связываются

МНОГИЕ — С — **МНОГИМИ**
из этих записей из этих записей

woman_id	woman	shoe_id	shoe_name
1	Кэрри	1	Сандалии
2	Саманта	2	Сабо
3	Шарлотта	3	Шлепанцы
4	Миранда	4	Туфли «Прада»

А теперь представьте, что наши героини купили по паре каждой обуви, которой у них нет. В этом случае связь между таблицами примет следующий вид.

На обоих концах соединительных линий имеются стрелки; мы связываем многие записи со многими.

woman_id	woman	shoe_id	shoe_name
1	Кэрри	1	Сандалии
2	Саманта	2	Сабо
3	Шарлотта	3	Шлепанцы
4	Миранда	4	Туфли «Прада»

МОЗГОВОЙ ШТУРМ

Как изменить структуру таблиц без хранения нескольких значений в одном столбце (чтобы не столкнуться с теми же проблемами, что и Грег в своих запросах для Реджи)?

Возьми в руку карандаш



Взгляните на первую пару таблиц. Мы попытались решить проблему, включая столбец shoe_id в таблицу с информацией о женщинах в качестве внешнего ключа.

woman_id	woman	shoe_id
1	Кэрри	3
2	Саманта	1
3	Шарлотта	1
4	Миранда	1
5	Кэрри	4
6	Шарлотта	2
7	Шарлотта	3
8	Шарлотта	4
9	Миранда	3
10	Миранда	4

shoe_id	shoe_name
1	Сандалии
2	Сабо
3	Шлепанцы
4	Туфли «Прада»

Две таблицы связаны через столбец «shoe_id».

А теперь изобразите структуру таблиц, но на этот раз включите столбец woman_id в таблицу shoes в качестве внешнего ключа.

Когда это будет сделано, нарисуйте связи между таблицами.



Возьми в руку карандаш
Решение



Взгляните на первую пару таблиц. Мы попытались решить проблему, включая столбец `shoe_id` в таблицу с информацией о женщинах в качестве внешнего ключа.

woman_id	woman	shoe_id
1	Кэрри	3
2	Саманта	1
3	Шарлотта	1
4	Миранда	1
5	Кэрри	4
6	Шарлотта	2
7	Шарлотта	3
8	Шарлотта	4
9	Миранда	3
10	Миранда	4

shoe_id	shoe_name
1	Сандалии
2	Сабо
3	Шлепанцы
4	Туфли «Прада»

↑
← Две таблицы связаны через столбец `shoe_id`.

Обратите внимание на дубликаты в столбцах «woman» и «shoe_name».

А теперь изобразите структуру таблиц, но на этот раз включите столбец `woman_id` в таблицу `shoes` в качестве внешнего ключа.

Когда это будет сделано, нарисуйте связи между таблицами.

shoe_id	shoe_name	woman_id
1	Сандалии	3
2	Сабо	2
3	Шлепанцы	1
4	Туфли «Прада»	1
5	Сабо	3
6	Шлепанцы	3
7	Туфли «Прада»	3
8	Сандалии	4
9	Шлепанцы	4
10	Туфли «Прада»	4

woman_id	woman
1	Кэрри
2	Саманта
3	Шарлотта
4	Миранда



Нам нужна соединительная таблица

Как вы только что убедились, включение любого из первичных ключей другой таблицы в качестве внешнего ключа приводит к дублированию данных. Обратите внимание, сколько раз в таблице повторяются имена женщин. В идеале они должны встречаться в данных только один раз.

Нам понадобится дополнительная таблица, которая свяжет между собой эти две таблицы и упростит связи «многие-ко-многим» до «один-ко-многим». В этой таблице будут храниться все значения `woman_id` вместе со значениями `shoe_id`. Нам понадобится **соединительная таблица** со значениями первичных ключей двух связываемых таблиц.

Прямое связывание этих двух таблиц неэффективно, потому что из-за связей «многие-ко-многим» в данных появляется много дубликатов.

woman_id	woman
1	Кэрри
2	Саманта
3	Шарлотта
4	Миранда

shoe_id	shoe_name
1	Сандалии
2	Сабо
3	Шлепанцы
4	Туфли «Прада»

«многие-ко-многим»

Берем первичный ключ из этой таблицы...

...другой первичный ключ из этой...

...и размещаем их в соединительной таблице.

«один-ко-многим»

«один-ко-многим»

В соединительной таблице хранятся первичные ключи двух связываемых таблиц.

Столбцы первичных ключей обеих исходных таблиц связываются с соответствующими столбцами соединительной таблицы.

woman_id	shoe_id
1	3
1	4
2	1
3	1
3	2
3	3
3	4
4	1
4	3
4	4

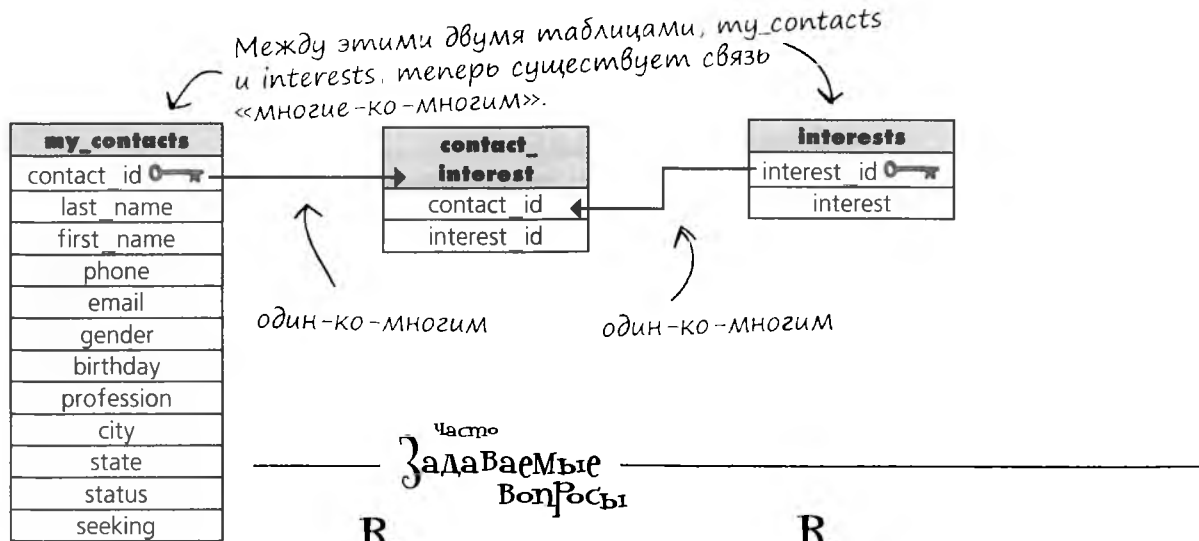
«Многие-ко-многим»: в соединительной таблице хранятся ключи из обеих таблиц.

Типы связей: «многие-ко-многим»

Теперь вам известен главный секрет связей «многие-ко-многим»: обычно они состоят из *двух* связей «один-ко-многим», объединенных при помощи *соединительной таблицы*.

ОДИН человек из таблицы `my_contacts` связывается со МНОГИМИ увлечениями из новой таблицы `interests`. Но так как каждое увлечение может принадлежать нескольким людям, такая связь относится к типу «многие-ко-многим».

В соответствии с этой схемой столбец `interests` может быть преобразован в связь «многие-ко-многим». У каждого человека может быть несколько увлечений, и каждое увлечение может принадлежать нескольким людям:



В: Всегда ли следует создавать соединительную таблицу в связях типа «многие-ко-многим»?

О: Да, всегда. Связи «многие-ко-многим» между двумя таблицами приводят к возникновению дубликатов, нарушающих требования первой нормальной формы (через пару страниц мы напомним, что такое нормализация).

Не существует веских причин в пользу нарушения первой нормальной формы, зато доводов «против» предостаточно. Самый серьезный из них — сложности с построением запросов при наличии дубликатов.

В: И какую пользу мне принесет такое изменение? Я с таким же успехом могу разместить все увлечения в таблице со столбцами `contact_id` и `interest_name`. Конечно, в ней будут дубликаты, но в остальном — почему бы и нет?

О: Вы поймете преимущества такой структуры в следующей главе, когда мы начнем строить запросы к связанным таблицам с использованием соединений. Кроме того, эти преимущества также могут зависеть от особенностей использования данных. Может оказаться так, что в таблице вас больше интересует именно связь «многие-ко-многим», а не данные в каждой из связываемых таблиц.

В: А если я все равно не против дубликатов?

О: Связывание таблиц помогает обеспечить целостность данных. Например, если вам потребуется удалить записи из `my_contacts`, изменятся только таблица `contact_interest`. Без отдельной таблицы вы можете случайно удалить лишние записи. Получается, что такая структура безопаснее.

Также упрощается обновление информации. Допустим, вы допустили ошибку в описании увлечения — например, написали «туризм». Чтобы исправить ее, будет достаточно изменить всего одну запись в таблице `interests`, а содержимое таблиц `contact_interest` и `my_contacts` останется неизменным.

ВЫБЕРИТЕ ТИП СВЯЗИ

В каждой из представленных ниже таблиц, знакомых вам по предыдущим главам, решите, какой тип связи лучше использовать для помеченного столбца — «один-ко-многим» или «многие-ко-многим».

(Не забудьте, что при наличии связи «один-ко-многим» и «многие-ко-многим» столбец выделяется из таблицы и связывается через идентификатор.)

СТОЛБЕЦ

СВЯЗЬ

doughnut_rating
doughnut_type
rating

clown_tracking
clown_id
activities
date

my_contacts
contact_id
state
interests

books
book_id
authors
publisher


fish_records
record_id
fish_species
state


ВЫБЕРИТЕ ТИП СВЯЗИ


В каждой из представленных ниже таблиц, знакомых вам по предыдущим главам, решите, какой тип связи лучше использовать для помеченного столбца — «один-ко-многим» или «многие-ко-многим».


(Не забудьте, что при наличии связи «один-ко-многим» и «многие-ко-многим» столбец выделяется из таблицы и связывается через идентификатор.)


СТОЛБЕЦ

doughnut_rating
doughnut_type

rating

clown_tracking
clown_id 
activities
date

my_contacts
contact_id 
state
interests

books
book_id 
authors
publisher

fish_records
record_id 
fish species
state

СВЯЗЬ

«один-ко-многим»

«многие-ко-многим»

«один-ко-многим»

«многие-ко-многим»

Вопрос с подвохом: у книги может быть несколько авторов, поэтому связь относится к типу «многие-ко-многим».

«многие-ко-многим»

«один-ко-многим»

«один-ко-многим»

«один-ко-многим»

Исправляем таблицу Грега

Я понял, к чему вы клоните. Мы преобразуем базу данных gregs_list и таблицу my_contacts в многотабличную форму, верно?

Почти. Теперь, когда вы разбираетесь в типах связей, мы почти готовы к переработке структуры gregs_list.

Мы знаем, что столбец interests можно связать с другой таблицей связью типа «один-к-многим». Столбец seeking тоже необходимо исправить аналогичным образом. После этих изменений таблица будет соответствовать критериям *первой нормальной формы**.

Но мы не можем остановиться на первой нормальной форме – нормализацию необходимо продолжить. Чем сильнее нормализуется таблица, тем проще получить из нее данные посредством запроса (или соединения – см. следующую главу). Но прежде чем создавать новую схему для базы данных gregs_list, мы познакомимся с другими уровнями нормализации.



my_contacts
contact_id 0
last_name
first_name
phone
email
gender
birthday
profession
city
state
status
interests
seeking

* Вам захотелось вернуться на несколько глав назад, чтобы вспомнить, что такое первая нормальная форма? Не нужно, мы напомним вам на следующей странице.

Не в первой нормальной форме

Мы упомянули о первой нормальной форме. Давайте еще раз вспомним, что это такое, а потом продолжим нормализацию до второй и даже третьей нормальной формы.

Итак, таблица, находящаяся в первой нормальной форме, должна удовлетворять следующим условиям.

Первая нормальная форма, или 1НФ:

Правило 1. Столбцы содержат только атомарные значения.

Правило 2. В таблице нет повторяющихся групп данных.

Изображенные ниже таблицы не соответствуют требованиям первой нормальной формы. Обратите внимание: во второй таблице добавились новые столбцы для цветов, но сами цвета при этом повторяются в записи.

Не находится в 1НФ

toy_id	toy	colors
5	мяч	белый, желтый синий
6	фрисби	зеленый, желтый
9	воздушный змей	красный, синий, зеленый
12	йо-йо	белый, желтый

Чтобы столбец «colors» был атомарным, он должен содержать только один из этих цветов, а не 2 или 3 в одной записи.

Все равно не находится в 1НФ

toy_id	toy	color1	color2	color3
5	мяч	белый	желтый	синий
6	фрисби	зеленый	желтый	
9	воздушный змей	красный	синий	зеленый
12	йо-йо	белый	желтый	

Эта таблица все еще не 1НФ, потому что столбцы все еще содержат те же типы данных, все VARCHAR с цветами игрушек.

Наконец-то — 1НФ...

Давайте посмотрим, что здесь нужно сделать.

В 1НФ

Первичный
ключ.

toy_id	toy
5	мяч
6	фрисби
9	воздушный змей
12	йо-йо

Внешний ключ.

toy_id	color
5	белый
5	желтый
5	синий
6	зеленый
6	желтый
9	красный
9	синий
9	зеленый
12	белый
12	желтый

Каждая запись
содержит только
один цвет, и все
записи уникальны.

Мы знаем, что эти
таблицы связаны
внешним ключом,
и рисовать линии
уже не обязательно.

Вместе значения столбцов
«toy_id» и «color» образуют
уникальный первичный ключ.

Столбец toy_id в отдельной таблице в качестве внешнего ключа — это нормально, потому что хранимые в нем значения не обязаны быть уникальными. При добавлении в эту таблицу значений color **все записи уникальны**, потому что цвет **В СОЧЕТАНИИ** с toy_id образует **уникальную комбинацию**.

Первичный ключ из нескольких столбцов? Но разве первичный ключ не должен быть только одним столбцом?



Нет. Ключ, состоящий из двух и более столбцов, называется составным ключом.

Рассмотрим еще несколько примеров использования составных ключей.

Составные ключи состоят из нескольких столбцов

До настоящего момента мы рассматривали связи данных таблицы с другими таблицами («один-к-одному», «один-ко-многим»). Однако пока ничего не было сказано о том, как столбцы таблицы связываются друг с другом. А без этого понять суть второй и третьей нормальных форм невозможно. Но зато потом потом намного упростится создание схем баз данных с запросами к нескольким таблицам.

Итак, что же такое «составной ключ»?

СОСТАВНЫМ КЛЮЧОМ называется **ПЕРВИЧНЫЙ КЛЮЧ**, состоящий из нескольких столбцов, комбинация которых образует уникальные значения.

← Для использования соединений (которыми мы займемся в следующей главе) таблицы должны иметь хорошо спроектированную структуру!



Возьмем следующую таблицу с информацией о супергероях. Таблица не имеет уникального ключа, но мы можем создать составной первичный ключ из столбцов name и power. Хотя в каждом из этих столбцов могут встречаться повторяющиеся значения, их комбинация всегда уникальна.

При создании этой таблицы можно указать, что эти два поля образуют составной первичный ключ. Предполагается, что супергерои с одинаковыми именами никогда не обладают одинаковыми суперспособностями, так что сочетание этих двух значений уникально.

super_heroes

name	power	weakness
Супер-Мусорщик	Моментально убирает мусор	отбеливатель
Брокер	Делает деньги из ничего	NULL
Супер-Парень	Летает	птицы
Чудо-Официант	Никогда не забывает заказы	насекомые
Грязнуля	Создает пыльные бури	отбеливатель
Супер-Парень	Обладает суперсилой	другие супермачо
Злая Тетка	Бывает очень, очень злой	NULL
Жаба	Язык справедливости	насекомые
Библиотекарь	Найдет все	NULL
Гусыня	Летает	NULL
Нарисованный Человек	Изображает людей	игра «Виселица»



Даже супергерои от чего-нибудь зависят

У наших супергероев много работы! Перед вами обновленная таблица `super_heroes`. Она соответствует требованиям ИНФ, но тут возникает другая проблема.

Столбец `initials` содержит сокращение, то есть начальные буквы значения столбца `name`. А что произойдет, если супергерой вдруг захочет сменить имя?

Точно, содержимое столбца `initials` тоже должно измениться. Говорят, что столбец `initials` **функционально зависим** от столбца `name`.

Эти два имени совпадают, но в сочетании со значением столбца «power» создается уникальный составной первичный ключ.

Если изменение содержимого одного столбца должно приводить к изменению другого, говорят, что второй столбец функционально зависим от первого.

`super_heroes`

name O+*	power O+*	weakness	city	country	arch_enemy	initials
Супер-Мусорщик	Моментально убирает мусор	отбеливатель	Готэм	США	Неряха	СМ
Брокер	Делает деньги из ничего	NULL	Нью-Йорк	США	Налоговый Инспектор	БР
Супер-Парень	Летает	птицы	Метрополис	США	Супер-Зануда	СП
Чудо-Официант	Никогда не забывает заказы	насекомые	Париж	Франция	Обжора	ЧО
Грязнуля	Создает пыльные бури	отбеливатель	Тулза	США	Гувер	ГР
Супер-Парень	Обладает суперсилой	алюминий	Метрополис	США	Плохиш	СП
Злая Тетка	Бывает очень, очень злой	NULL	Рим	Италия	Психоаналитик	ЗТ
Жаба	Язык справедливости	насекомые	Лондон	Англия	Цапля	ЖА
Библиотекарь	Найдет все	дети	Спрингфилд	США	Хаос	БИ
Гусыня	Летает	NULL	Миннеаполис	США	Охотник	ГУ
Нарисованный Человечек	Изображает людей	игра «Виселица»	Лондон	Англия	Ластик	НЧ

Возьми в руку карандаш



Итак, в таблице супергероев столбец `initials` зависим от столбца `name`. А вы видите еще какие-нибудь похожие зависимости? Запишите их здесь.

Возьми в руку карандаш Решение



Итак, в таблице супергероев столбец `initials` зависит от столбца `name`. А вы видите еще какие-нибудь похожие зависимости? Запишите их здесь.

`initials` зависит от `name` ←
`weakness` зависит от `name` ←
`arch_enemy` зависит от `name` ←
`city` зависит от `country` ←

Из этой записи не ясно, в какой таблице находятся столбцы; это может быть существенно при добавлении новых таблиц. Существует специальный сокращенный синтаксис для обозначения этих зависимостей и таблиц, в которых они находятся.

Сокращенная запись



Для компактного описания функциональных зависимостей часто используется следующая запись:

T.x → T.y

Это можно прочесть так: «В таблице с именем T столбец y функционально зависит от столбца x». Зависимый столбец указывается в правой части.

Применительно к нашим супергероям это выглядит так:

`super_heroes.name → super_heroes.initials`

«В таблице `super_heroes` столбец `initials` функционально зависит от столбца `name`».

`super_heroes.name → super_heroes.weakness`

«В таблице `super_heroes` столбец `weakness` функционально зависит от столбца `name`».

`super_heroes.name → super_heroes.arch_enemy`

«В таблице `super_heroes` столбец `arch_enemy` функционально зависит от столбца `name`».

`super_heroes.country → super_heroes.city`

«В таблице `super_heroes` столбец `city` функционально зависит от столбца `country`».

Супергеройские зависимости

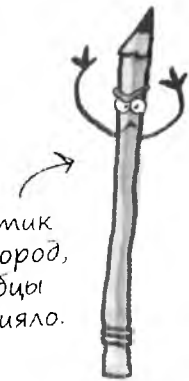
Итак, если наш супергерой меняет имя, столбец `initials` тоже должен измениться; это означает, что столбец **зависит** от столбца `name`.

Если заклятый враг супергероя решит переехать в другой город, то изменится его текущее местонахождение – и только. Таким образом, столбец `arch_enemy_city` в приведенной ниже таблице абсолютно **независим**.

Зависимым называется столбец с данными, которые могут измениться в случае изменения другого столбца. **Независимые** столбцы *существуют сами по себе*.



Если Нарисованный Человек вдруг захочет называть себя Дистрофиком, то и столбец «initials» тоже придется изменить.

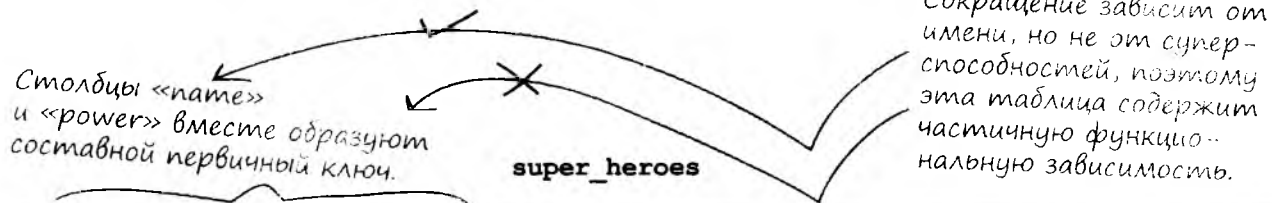


Тем временем Ластик переехал в новый город, но на другие столбцы это никак не повлияло.

Частичные функциональные зависимости

Частичная функциональная зависимость означает, что не-ключевой столбец зависит от некоторых, но не от всех столбцов составного первичного ключа.

В нашей таблице столбец `initials` **частично независим** от `name`, потому что в случае изменения имени супергероя столбец `initials` тоже изменится, а в случае изменения `power` (но не `name`!) столбец `initials` останется неизменным.



<code>name</code>	<code>power</code>	<code>weakness</code>	<code>city</code>	<code>initials</code>	<code>arch_enemy_id</code>	<code>arch_enemy_city</code>
Супер-Мусорщик	Моментально убирает мусор	отбеливатель	Готэм	ST	4	Готэм
Брокер	Делает деньги из ничего	NULL	Нью-Йорк	TB	8	Ньюарк
Супер-Парень	Летает	птицы	Метрополис	SG	5	Метрополис
Чудо-Официант	Никогда не забывает заказы	насекомые	Париж	WW	1	Париж
Грязнуля	Создает пыльные бури	отбеливатель	Тулза	D	2	Канзас-Сити
Супер-Парень	Обладает суперсилой	алюминий	Метрополис	SG	7	Готэм
Злая Тетка	Бывает очень, очень злой	NULL	Рим	FW	10	Рим
Жаба	Язык справедливости	насекомые	Лондон	T	16	Бат
Библиотекарь	Найдет все	дети	Спрингфилд	L	3	Луисвилль
Гусыня	Летает	NULL	Миннеаполис	GG	9	Миннеаполис
Нарисованный Человек	Изображает людей	игра «Виселица»	Лондон	S	33	Бородейл

Транзитивные функциональные зависимости

Также необходимо учесть и связи всех неключевых столбцов с другими столбцами. Если заклятый враг какого-либо супергероя переедет в другой город, его значение `arch_enemy_id` от этого не изменится.

Значение «`arch_enemy_id`» не изменилось, хотя Неряха и переехал в Канзас-Сити.

name 0+*	arch_enemy_id	arch_enemy_city
Супер-Мусорщик	4	Канзас-Сити
Брокер	8	Ньюарк
Супер-Парень	5	Метрополис
Чудо-Официант	1	Париж
Грязнуля	2	Канзас-Сити

Если изменение не-ключевого столбца может привести к изменению других столбцов, значит, существует транзитивная зависимость.

Предположим, супергерой захотел поменять себе заклятого врага. Значение `arch_enemy_id` при этом изменится, а это *может* привести к изменению `arch_enemy_city`.

Если изменение не-ключевого столбца приводит к изменению других столбцов, значит, существует *транзитивная зависимость*.

Если обновление «`arch_enemy_id`» приводит к изменению значения в столбце «`arch_enemy_city`»...

name 0+*	arch_enemy_id	arch_enemy_city
Супер-Мусорщик	2	Канзас-Сити
Брокер	8	Ньюарк
Супер-Парень	5	Метрополис
Чудо-Официант	1	Париж
Грязнуля	2	Канзас-Сити

...это называется транзитивной функциональной зависимостью, потому что не-ключевой столбец «`arch_enemy_city`» связан со столбцом «`arch_enemy_id`», который также является не-ключевым.

Транзитивная функциональная зависимость: не-ключевой столбец связан с другими не-ключевыми столбцами.



Упражнение

В следующей таблице хранится информация о книгах. Столбец `pub_id` определяет издателя, а столбец `pub_city` — город, в котором была опубликована книга.

<code>author</code> O+K	<code>title</code> O+K	<code>copyright</code>	<code>pub_id</code>	<code>pub_city</code>
Джон Дир	В мире с природой	1930	2	Нью-Йорк
Фред Мерц	Я ненавижу Люси	1968	5	Бостон
Лесси	Помогите Тимми!	1950	3	Сан-Франциско
Тимми	Лесси, успокойся	1951	1	Нью-Йорк

Напишите, что произойдет со значением столбца `copyright`, если столбец `title` в третьей записи примет значение «Вытащите Тимми из колодца!».

При изменении названия изменится и значение «copyright».

← Столбец «copyright» зависит от «title», поэтому его значение изменится.

Что произойдет со значением столбца `copyright`, если автор книги в третьей записи изменится, а название останется прежним?

Что произойдет с записью «В мире с природой», если ее полю `pub_id` будет присвоено значение 1?

Что произойдет со значением `pub_id` записи «Я ненавижу Люси», если издатель переместится в другой город?

Что произойдет со значением `pub_city` записи «Я ненавижу Люси», если ее полю `pub_id` будет присвоено значение 1?



Упражнение
Решение

В следующей таблице хранится информация о книгах. Столбец `pub_id` определяет издателя, а столбец `pub_city` — город, в котором была опубликована книга.

Напишите, что произойдет со значением столбца `copyright`, если столбец `title` в третьей записи примет значение «Вытащите Тимми из колодца!». Столбец «`copyright`» зависит от «`title`», поэтому и значение «`copyright`».

Что произойдет со значением столбца `copyright`, если автор книги в третьей записи изменится, а название останется прежним?

При изменении автора, но не названия изменится и значение «`copyright`».

Автор и название вместе образуют составной первичный ключ

«`Copyright`» зависит от названия и от автора.

<code>author 0+*</code>	<code>title 0+*</code>	<code>copyright</code>	<code>pub_id</code>	<code>pub_city</code>
Джон Дир	В мире с природой	1930	2	Нью-Йорк
Фред Мерц	Я ненавижу Люси	1968	5	Бостон
Лесси	Помогите Тимми!	1950	3	Сан-Франциско
Тимми	Лесси, успокойся	1951	1	Нью-Йорк

Что произойдет с записью «В мире с природой», если ее полю `pub_id` будет присвоено значение 1?

«`pub_city`» для «`pub_id 1`» и «`pub_id 2`» — Нью-Йорк, поэтому город не меняется («`pub_city`» не изменится. (даже когда «`pub_city`» зависит от «`pub_id`»)).

`pub_id` не зависит от столбца `pub_city`, и поэтому остается тем же.

Что произойдет со значением `pub_id` записи «Я ненавижу Люси», если издатель переместится в другой город?

«`pub_id`» остается тем же.

Что произойдет со значением `pub_city` записи «Я ненавижу Люси», если ее полю `pub_id` будет присвоено значение 1?

«`pub_city`» транзитивно зависит от «`pub_id`», поэтому значение меняется. «`pub_city`» становится Нью-Йорк.

«`pub_city`» зависит от значения в столбце «`pub_id`» — транзитивная функциональная зависимость.

<code>author 0+*</code>	<code>title 0+*</code>	<code>copyright</code>	<code>pub_id</code>	<code>pub_city</code>
Джон Дир	В мире с природой	1930	2	Нью-Йорк
Фред Мерц	Я ненавижу Люси	1968	5	Бостон
Лесси	Помогите Тимми!	1950	3	Сан-Франциско
Тимми	Лесси, успокойся	1951	1	Нью-Йорк

Часть
**Задаваемые
 Вопросы**

В: Существует ли простой способ устранения частичных функциональных зависимостей?

О: Использование столбца-идентификатора, как в таблице `my_contacts`, полностью решает все проблемы. Так как этот столбец представляет собой новый ключ, который создается только для индексирования этой таблицы, никакие другие столбцы от него не зависят.

В: Когда и зачем мне могут потребоваться составные ключи из столбцов таблицы (если не считать соединительных таблиц)? Почему нельзя всегда создавать столбец-идентификатор?

О: Безусловно, это решает проблему. Однако попробуйте провести поиск в Интернете по условию «синтетические или естественные ключи» — вы найдете убедительные аргументы в пользу обоих решений, а также немало горячих споров. Лучше, если вы примете решение самостоятельно. В этой книге в основном используется решение с синтетическим ключом, чтобы вы смогли понять суть концепции, не отвлекаясь на тонкости реализации.

Зависимости — это, конечно, хорошо. Но какое отношение они имеют к переходу от первой нормальной формы ко второй?



Включение столбцов первичных ключей в таблицы способствует выполнению требований 2НФ.

Для простоты и удобства, а также для обеспечения уникальности мы обычно включали во все свои таблицы столбцы, которые использовались как первичные ключи. Это способствует выполнению требований 2НФ, потому что вторая нормальная форма определяет *связь первичного ключа таблицы с хранящимися в ней данными*.

Вторая нормальная форма

Как покажут следующие две таблицы, используемые в системе складского учета магазинов игрушек, требования второй нормальной формы относятся к отношениям между первичным ключом таблицы и хранящимися в ней данными.

toy_id	toy
5	мяч
6	фрисби
9	воздушный змей
12	йо-йо

Составной ключ.

toy_id 0+*	store_id 0+*	color	inventory	store_address
5	1	белый	34	23 Мейпл
5	3	желтый	12	100 Норт-стрит
5	1	синий	5	23 Мейпл
6	2	зеленый	10	1902 Эмберлайн
6	4	желтый	24	17 Инглсайд
9	1	красный	50	23 Мейпл
9	2	синий	2	1902 Эмберлайн
9	2	зеленый	18	1902 Эмберлайн
12	4	белый	28	17 Инглсайд
12	4	желтый	11	17 Инглсайд

Столбец содержит много дубликатов, причем эти дубликаты не содержат полезной информации об игрушках: они относятся к магазину.

Над этим столбцом тоже стоит хорошенько подумать. Эти данные скорее должны храниться в таблице игрушек, а не в складских данных. Столбец «toy_id» должен идентифицировать как тип, ТАК И цвет игрушки.

Количество единиц товара зависит от обоих столбцов, образующих составной первичный ключ, поэтому частичная функциональная зависимость отсутствует.

Обратите внимание на дублирование store_address для игрушек, связанных с идентификатором магазина store_id. Если нам вдруг понадобится изменить адрес магазина, придется изменять каждую запись таблицы, в которой он присутствует. Чем больше записей обновляется с течением времени, тем выше вероятность того, что в данных появятся случайные ошибки.

С другой стороны, если выделить столбец store_address в отдельную таблицу, то адрес будет достаточно изменить только в одном месте.

Возможно, таблица уже находится в 2НФ...

Таблица 1НФ также находится в 2НФ, если все столбцы таблицы являются частью первичного ключа.

Мы можем создать новую таблицу с составным первичным ключом из столбцов `toy_id` и `store_id`. Тогда в одной таблице будет храниться вся информация об игрушках, в другой — вся информация о магазинах, а новая таблица будет связывать эти две таблицы.

Таблица 1НФ находится в 2НФ, если все столбцы таблицы являются частью первичного ключа

ИЛИ

она имеет одностолбцовый первичный ключ.

Вся информация о них.

Игрушки

toy_store
toy_id
store_id

Вся информация о них.

Магазины

Таблица 1НФ также находится в 2НФ, если она имеет одностолбцовый первичный ключ.

И это хорошая причина для создания столбца-идентификатора с условием `AUTO_INCREMENT`.

Вторая нормальная форма, или 2НФ:

Правило 1. Таблица находится в 1НФ.

Правило 2. Таблица не имеет частичных функциональных зависимостей.



Вряд ли в `my_contacts` есть частичные функциональные зависимости, хотя...

Значит, пора поиграть...



Станьте таблицей 2NF с частичными функциональными зависимостями

Представьте себя на месте таблицы и исключите из себя все частичные функциональные зависимости. В каждой из представленных таблиц вычеркните те столбцы, которые лучше переместить в отдельную таблицу.

Эти два столбца образуют уникальный составной первичный ключ.

toy_inventory
toy_id
store_id

singers
singer_id
last_name
first_name
agency
agency_state

cookie_sales
amount
girl_id
date
girl_name
troop_leader
total_sales

movies
movie_id
title
genre
rented_by
due_date
rating

salary
employee_id
last_name
first_name
salary
manager
employee_email
hire_date

dog_breeds
breed
description
avg_weight
avg_height
club_id
club_state

Возьми в руку карандаш



Преобразуйте эти таблицы в три таблицы, соответствующие требованиям 2НФ.

Одна таблица должна содержать информацию об игрушках, другая — о магазинах, а третья — содержать данные о наличии товара и связывать первые две между собой. Присвойте всем трем таблицам содержательные имена.

Добавьте в соответствующие таблицы столбцы `phone`, `manager`, `cost` и `weight`. Возможно, вам придется создать новые значения `toy_id`.

toy_id	toy
5	мяч
6	фрисби
9	воздушный змей
12	йо-йо

toy_id 0+K	store_id 0+K	color	inventory	store_address
5	1	белый	34	23 Мейпл
5	3	желтый	12	100 Норт-стрит
5	1	синий	5	23 Мейпл
6	2	зеленый	10	1902 Эмберлайн
6	4	желтый	24	17 Инглсайд
9	1	красный	50	23 Мейпл
9	2	синий	2	1902 Эмберлайн
9	2	зеленый	18	1902 Эмберлайн
12	4	белый	28	17 Инглсайд
12	4	желтый	11	17 Инглсайд

Станьте таблицей 2нф с частичными функциональными зависимостями. Ответ



Представьте себя на месте таблицы

и исключите из себя все частичные функциональные зависимости. В каждой из представленных таблиц вычеркните те столбцы, которые лучше переместить

в отдельную таблицу.

Эти два столбца образуют уникальный составной первичный ключ.

toy_inventory
toy_id
store_id

Первичный ключ.

singers
singer_id
last_name
first_name
agency
agency_state

Хотя здесь следовало бы разместить идентификатор из таблицы agency (два агентства могут иметь одинаковые названия), частичной функциональной зависимости нет.

cookie_sales
amount
girl_id
date
girl_name
treep_leader
total_sales

После исключения этих столбцов оставшиеся образуют составной первичный ключ.

Первичный ключ.

movies
movie_id
title
genre
rented_by
due_date
rating

Эти столбцы связаны только транзитивной функциональной зависимостью.

Первичный ключ.

salary
employee_id
last_name
first_name
salary
manager
employee_email
hire_date

Этим данным здесь не место, но частичной функциональной зависимости нет.

dog_breeds
breed
description
avg_weight
avg_height
club_id
club_state

Составной первичный ключ.

Столбец «club_id» может находиться в этой таблице (если это связь «один-к-одному»), столбцу «club_state» здесь явно не место. Но несмотря на это, среди столбцов нет частичных функциональных зависимостей.

Возьми в руку карандаш



Решение Преобразуйте эти таблицы в три таблицы, соответствующие требованиям 2НФ.

Одна таблица должна содержать информацию об игрушках, другая — о магазинах, а третья — содержать данные о наличии товара и связывать первые две между собой. Присвойте всем трем таблицам содержательные имена.

Добавьте в соответствующие таблицы столбцы phone, manager, cost и weight. Возможно, вам придется создать новые значения toy_id.

toy_id	toy
5	мяч
6	фрисби
9	воздушный змей
12	йо-йо

toy_id	store_id	color	inventory	store_address
5	1	белый	34	23 Мейпл
5	3	желтый	12	100 Норт-стрит
5	1	синий	5	23 Мейпл
6	2	зеленый	10	1902 Эмберлайн
6	4	желтый	24	17 Инглсайд
9	1	красный	50	23 Мейпл
9	2	синий	2	1902 Эмберлайн
9	2	зеленый	18	1902 Эмберлайн
12	4	белый	28	17 Инглсайд
12	4	желтый	11	17 Инглсайд

toy_info

toy_id	toy	color	cost	weight
1	мяч	белый	1.95	0.3
2	мяч	желтый	2.20	0.4
3	мяч	синий	1.95	0.3
4	фрисби	зеленый	3.50	0.5
5	фрисби	желтый	1.50	0.2
6	воздушный змей	красный	5.75	1.2
7	воздушный змей	синий	5.75	1.2
8	воздушный змей	зеленый	3.15	0.8
9	йо-йо	белый	4.25	0.4
10	йо-йо	желтый	1.50	0.2

store_inventory

toy_id	store_id	inventory
5	1	34
5	3	12
5	1	5
6	2	10
6	4	24
9	1	50
9	2	2
9	2	18
12	4	28
12	4	11

store_info

store_id	address	phone	manager
1	23 Мейпл	555-6712	Джо
2	1902 Эмберлайн	555-3478	Сьюзен
3	100 Норт-стрит	555-0987	Тара
4	17 Инглсайд	555-6554	Гордон

Составной первичный ключ состоит из столбцов «toy_id» и «store_id».

Третья нормальная форма (наконец-то!)

Так как в книге мы по возможности добавляем «синтетические» первичные ключи, с переводом таблиц во вторую нормальную форму обычно проблем не бывает. Любая таблица с **синтетическим первичным ключом**, не имеющая составного первичного ключа, всегда находится в 2НФ.

Как убедиться, что мы в 3НФ?

Если таблица имеет синтетический первичный ключ и не имеет составного первичного ключа, она находится в 3НФ.

Третья нормальная форма, или ЗНФ:
Правило 1. Таблица находится в 2НФ.
Правило 2. Таблица не имеет транзитивных зависимостей.

Еще не забыли? Транзитивная функциональная зависимость означает наличие связей между не-ключевыми столбцами.

Если изменение какого-либо не-ключевого столбца может привести к изменению других столбцов, имеет место транзитивная зависимость.

Что произойдет при изменении значения какого-либо из трех столбцов: **course_name** (название учебного курса), **instructor** (преподаватель) и **instructor_phone** (телефон преподавателя).

- ⇒ При изменении **course_name** ни **instructor**, ни **instructor_phone** не изменяются.
- ⇒ При изменении **instructor_phone** ни **instructor**, ни **course_name** не изменяются.
- ⇒ При изменении **instructor** значение **instructor_phone** изменится. Мы обнаружили транзитивную зависимость.

При рассмотрении ЗНФ на первичный ключ можно не обращать внимания.

courses
course_id
course_name
instructor
instructor_phone


Чтобы таблица соответствовала требованиям ЗНФ, из нее необходимо убрать столбец «instructor_phone».



Упражнение

Что делать с таблицей my_contacts?

В нее необходимо внести несколько изменений. Начните с текущей версии таблицы my_contacts и изобразите новую схему gregs_list. Обозначьте связи между внешними ключами линиями, а связи типа «один-ко-многим» — стрелками. Также обозначьте первичные и составные ключи.

my_contacts
contact_id 
last_name
first_name
phone
email
gender
birthday
profession
city
state
status
interests
seeking

Подсказка. Наша версия на следующей странице состоит из 8 таблиц. (Мы добавили столбец для почтового индекса, а до этого было 7.)



Упражнение
Решение

Что делать с таблицей my_contacts?

В нее необходимо внести несколько изменений. Начните с текущей версии таблицы my_contacts и изобразите новую схему gregs_list. Обозначьте связи между внешними ключами линиями, а связи типа «один-ко-многим» — стрелками. Также обозначьте первичные и составные ключи.

my_contacts
contact_id
last_name
first_name
phone
email
gender
birthday
profession
city
state
status
interests
seeking

Связь «многие-ко-многим» состоит из двух связей «один-ко-многим» и соединительной таблицы.

Эти три связи относятся к типу «один-ко-многим».

profession
prof_id
profession

zip_code
zip_code
city
state

status
status_id
status

my_contacts
contact_id
last_name
first_name
phone
email
gender
birthday
→ prof_id
→ zip_code
→ status_id

contact_interest
contact_id
0+∞
interest_id
0+∞

interests
interest_id
interest

В таблице «contact_interest» некоторое значение «interest_id» может встречаться многократно, а в таблице «interests» — только один раз.

contact_seeking
contact_id
0+∞
seeking_id
0+∞

seeking
seeking_id
seeking

Два столбца образуют составной ключ.

Связь «многие-ко-многим» состоит из двух связей «один-ко-многим» и соединительной таблицы.

...А далее Регжу (и gregs_list) ждало счастливое будущее...

Грег смог найти идеальную пару для Реджи по своей новой нормализованной базе данных — и не только для Реджи, но и многим своим друзьям, и его мечты сбылись. В общем, все кончилось хорошо.



Конец



Стоп, не так быстро! Теперь мне нужно составить запросы ко всем этим новым таблицам! Как получить данные из набора связанных таблиц без написания сотни-другой запросов?

Вас спасут соединения.

До встречи в следующей главе...



Новые инструменты

Поздравляем, вы одолели больше половины книги. Напоминаем ключевые термины, которые вы узнали в главе 7. Полный список инструментов приведен в приложении III.

Схема

Описание данных, хранящихся в базе данных, включающее все объекты и связи между ними.

Связь «один-к-одному»

Ровно одна запись родительской таблицы связывается с одной записью дочерней таблицы.

Связь «один-ко-многим»

Запись одной таблицы может быть связана со многими записями другой таблицы, но каждая запись последней может быть связана только с одной записью в первой.

Связь «многие-ко-многим»

Две таблицы связываются через соединительную таблицу, благодаря чему многие записи первой таблицы могут быть связаны со многими записями второй, и наоборот.

Первая нормальная форма (1НФ)

Столбцы содержат только атомарные значения и в них отсутствуют повторяющиеся группы данных.

Транзитивная функциональная зависимость

Не-ключевой столбец связан с другим не-ключевым столбцом (-ами).

Вторая нормальная форма (2НФ)

Таблица находится в 1НФ и не содержит частичных функциональных зависимостей.

Третья нормальная форма (3НФ)

Таблица находится в 2НФ и не имеет транзитивных зависимостей.

Внешний ключ

Столбец таблицы, значения которого ссылаются на первичный ключ другой таблицы.

Составной ключ

Первичный ключ, состоящий из нескольких столбцов, комбинация которых образует уникальное значение ключа.

Возьми в руку карандаш
Решение

Прежде всего необходимо создать новые столбцы:

```
ALTER TABLE my_contacts
ADD COLUMN interest1 VARCHAR(50),
ADD COLUMN interest2 VARCHAR(50),
ADD COLUMN interest3 VARCHAR(50),
ADD COLUMN interest4 VARCHAR(50);
```

Затем первое увлечение переносится в новый столбец «interest1». Это можно сделать так:

```
UPDATE my_contacts
SET interest1 = SUBSTRING_INDEX(interests, ',', 1);
```

Первое увлечение, сохраненное в «interest1», нужно удалить из столбца «interests». Удаляются все символы до первой запятой включительно:

Функция TRIM удаляет пробелы у левого края строки после того, как мы удалим все символы вплоть до запятой.

Функция RIGHT возвращает правую часть строкового значения.

```
UPDATE my_contacts SET interests = TRIM(RIGHT(interests,
(LENGTH(interests) - LENGTH(interest1) - 1)));
```

А эта устрашающая конструкция вычисляет длину нужной части столбца «interests». Из общей длины «interests» вычитаемся длина части, перемещенной в «interest1». Затем мы вычитаем еще 1, чтобы усеченное значение начиналось после запятой.

Эти действия повторяются для остальных столбцов увлечений:

```
UPDATE my_contacts SET interest2 = SUBSTRING_INDEX(interests, ',', 1);
UPDATE my_contacts SET interests = TRIM(RIGHT(interests, (LENGTH(interests) -
LENGTH(interest2) - 1)));
UPDATE my_contacts SET interest3 = SUBSTRING_INDEX(interests, ',', 1);
UPDATE my_contacts SET interests = TRIM(RIGHT(interests, (LENGTH(interests) -
LENGTH(interest3) - 1)));
```

Для последнего столбца осталось только одно значение:

```
UPDATE my_contacts SET interest4 = interests;
```

Теперь столбец «interests» можно полностью удалить. Также можно было переименовать его в «interest4» и избавиться от лишней команды ADD COLUMN (предполагается, что увлечений не более четырех).

contact_id
last_name
first_name
phone
email
gender
birthday
profession
city
state
status
interest1
interest2
interest3
interest4
seeking



Упражнение

Решение

Со с. 316.

Напишите для Реджи запрос, не использующий столбец interests.

```
SELECT * FROM my_contacts
WHERE gender = 'Ж'
AND status = 'Не замужем'
AND state='MA'
AND seeking LIKE '%Неженатый
мужчина%'
AND birthday > '1950-20-03'
AND birthday < '1960-20-03';
```

← Фактически это тот же запрос, который Грег использовал для Найджела, только без проверки столбца «interests».

8 Соединения и Многотабличные операции

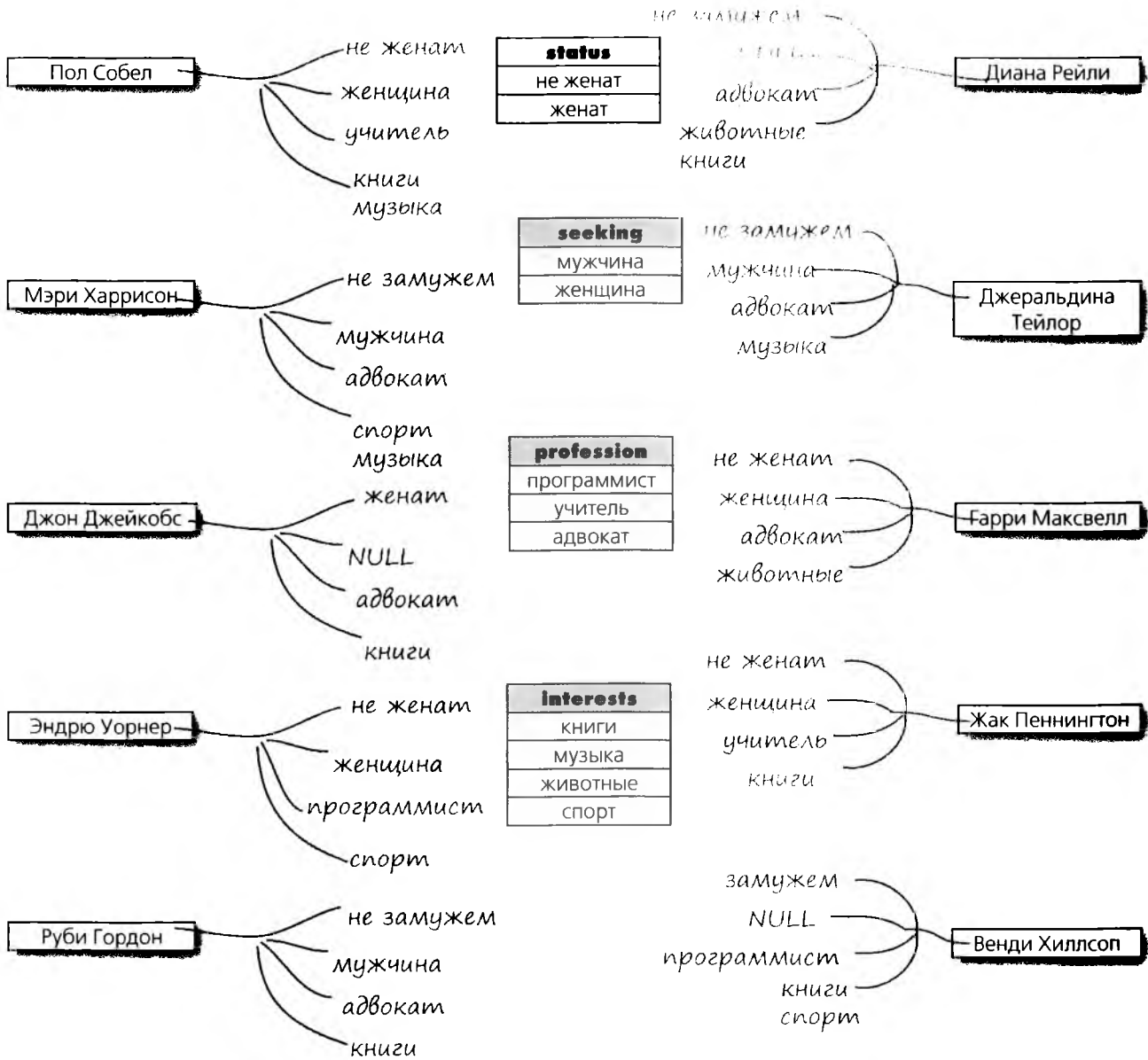
* Не могли бы мы остаться в одиночестве? *



Добро пожаловать в многотабличный мир! Базы данных, состоящие из **нескольких таблиц**, удобны, но чтобы успешно работать с ними, вам придется освоить некоторые новые **инструменты и приемы**. При работе с несколькими таблицами может возникнуть путаница, поэтому вам понадобятся **псевдонимы**. А **соединения** помогут установить связь между таблицами, чтобы снова собрать воедино информацию, разбросанную по разным таблицам. Приготовьтесь, пора **снова взять базу данных под свой полный контроль!**

И все равно повторения, повторения...

Грег заметил, что в столбцах **profession**, **interests** и **seeking** постоянно встречаются одни и те же значения.



Заполнение таблиц

Обилие повторяющихся значений упрощает заполнение таблиц `profession`, `interests` и `seeking`. Грег хочет заполнить эти таблицы значениями, уже хранящимися в старой таблице `my_contacts`.

Но сначала необходимо составить запрос и узнать, какие данные уже хранятся в таблице — и каким-то образом обойтись без огромного списка дубликатов.



Получить набор значений, хранящихся в таблице, — вполне стандартная задача, верно?

Возьми в руку карандаш



Напишите запросы, возвращающие список значений столбцов `profession`, `interests` и `seeking values` из старой таблицы `my_contacts`. В списке не должно быть дубликатов. Вспомните задачу с продажей печенья из главы 6.



Возьми в руку карандаш

Решение

Напишите запросы, возвращающие список значений столбцов profession, interests и seeking values из старой таблицы my_contacts. В списке не должно быть дубликатов. Вспомните задачу с продажей печенья из главы 6.

```
SELECT profession FROM my_contacts
```

```
GROUP BY profession  
ORDER BY profession;
```

Условие GROUP BY соединяет дубликаты в одно значение для каждой группы.

Затем условие ORDER BY упорядочивает список по алфавиту.

```
SELECT seeking FROM my_contacts
```

```
GROUP BY seeking  
ORDER BY seeking;
```

При нарушении порядка условий вы получите сообщение об ошибке. Условие ORDER BY всегда должно стоять на последнем месте.

```
SELECT interests  
FROM my_contacts  
GROUP BY interests  
ORDER BY interest;
```

Но этот запрос не подойдет для столбца interests. Ведь в этом столбце хранится несколько значений, помните?



Мы не сможем воспользоваться простым запросом SELECT для вывода информации об увлечениях.

С такими значениями аналогичная команда SELECT работать не будет.

interests
книги, спорт
музыка, животные, книги
животные, книги
спорт, музыка

Проблемы с нормализацией

Ненормализованная структура таблицы порождает массу проблем. Не существует простого способа извлечь эти значения из столбца `interests` по одному.

То, что у нас есть:

<code>interests</code>
один, два, три, четыре

← Столбец из таблицы `my_contacts`

То, что должно быть:

<code>interests</code>
один
два
три
четыре

← Столбец новой таблицы `interests`.

МОЗГОВОЙ ШТУРМ

Как выделить не-атомарные значения в один столбец таблицы `interests`?

Может, вручную? Мы можем просмотреть каждую запись `my_contacts`, а потом ввести каждое значение в новой таблице.

Прежде всего, это гигантский объем работы. Представьте, что таблица содержит тысячи записей.

Во-вторых, ручная обработка сильно усложнит поиск дубликатов. С сотнями разных увлечений вам придется при каждом вводе смотреть, не было ли данное увлечение введено ранее.

Зачем выполнять всю «черную работу» самостоятельно, рискуя натворить ошибок? Лучше поручить ее SQL.



Особые увлечения (столбец)

В одном довольно прямолинейном решении в таблице `my_contacts` создаются четыре новых столбца для временного хранения обрабатываемых значений. После завершения обработки столбцы будут удалены.

Возьми в руку карандаш



Вы уже умеете пользоваться командой `ALTER`; создайте в таблице `my_contacts` четыре новых столбца. Присвойте им имена `interest1`, `interest2`, `interest3` и `interest4`.

→ Ответы на с. 408.

Вот как будет выглядеть столбец `interests` с новыми столбцами `interest` в таблице `my_contacts` после выполнения `ALTER`.

<code>interests</code>	<code>interest1</code>	<code>interest2</code>	<code>interest3</code>	<code>interest4</code>
один, два, три, четыре				

Первое увлечение копируется в новый столбец `interest1` при помощи функции `SUBSTRING_INDEX` (см. главу 5):

```
UPDATE my_contacts
SET interest1 = SUBSTRING_INDEX(interests, ',', 1);
```

↑
Имя столбца.

↑
Искомый символ (запятая).

↑
...Поиск первого вхождения.

После выполнения команды таблица будет выглядеть так.

<code>interests</code>	<code>interest1</code>	<code>interest2</code>	<code>interest3</code>	<code>interest4</code>
один, два, три, четыре	один			

Разделение увлечений

А теперь самое сложное: мы воспользуемся другой функцией для удаления из текущего значения `interests` данных, скопированных в столбец `interest1`. После этого можно будет продолжить заполнение остальных столбцов по тому же принципу.

<code>interests</code>	<code>interest1</code>	<code>interest2</code>	<code>interest3</code>	<code>interest4</code>
один, два, три, четыре	один			

Мы удалим текст первого увлечения, следующую за ним запятую и пробел, следующий за запятой в столбце «`interests`».

Функция `SUBSTR` получает текст столбца `interests` и возвращает заданную его часть. Мы выделяем символы, которые были скопированы в `interest1`, а также еще два символа (запятая и пробел).

Заменить содержимое столбца `interests` тем, что в нем хранится сейчас, с удалением символов, скопированных в «`interest1`», запятой и пробела.

UPDATE my_contacts

SET interests = SUBSTR(interests, LENGTH(interest1)+2);

Функция `SUBSTR` возвращает часть исходного текста этого столбца. Она «отрезает» от него первую часть, описанную в круглых скобках, и возвращает вторую.

Как вы могли заметить, работа некоторых функций зависит от используемой разновидности SQL. Так вот, это одна из таких функций. За описанием вашей конкретной разновидности SQL обращайтесь к документации.

Длина текста в поле «`interest1`»... и еще 2 символа: запятая и пробел.

Функция `LENGTH` возвращает число — длину строки, указанной в круглых скобках.

В нашем примере длина строки «один» равна 4.

Итак, сумма будет равна $4+2$, или 6 — именно столько символов будет удалено в начале старого содержимого столбца `interests`.

Обновление столбцов

После выполнения команды UPDATE таблица будет выглядеть так, как показано ниже.

Однако работа еще не закончена. Теперь нужно проделать то же самое для столбцов interest2, interest3 и interest4.

interests	interest1	interest2	interest3	interest4
два, три, четыре	один			

Возьми в руку карандаш



Заполните пропуски в команде update. Мы привели пару подсказок, чтобы немного упростить вашу задачу.

Подсказка. С каждым вызовом SUBSTR текст столбца «interests» становится все короче.»

```

UPDATE my_contacts SET
interest1 = SUBSTRING_INDEX(interests, ',', 1),
interests = SUBSTR(interests, LENGTH(interest1)+2),
interest2 = SUBSTRING_INDEX(.....),
interests = SUBSTR(.....),
interest3 = SUBSTRING_INDEX(.....),
interests = SUBSTR(.....),
interest4 = .....
    
```

После удаления первых трех увлечений из столбца «interests» остается последнее, четвертое увлечение. Что с ним нужно сделать?

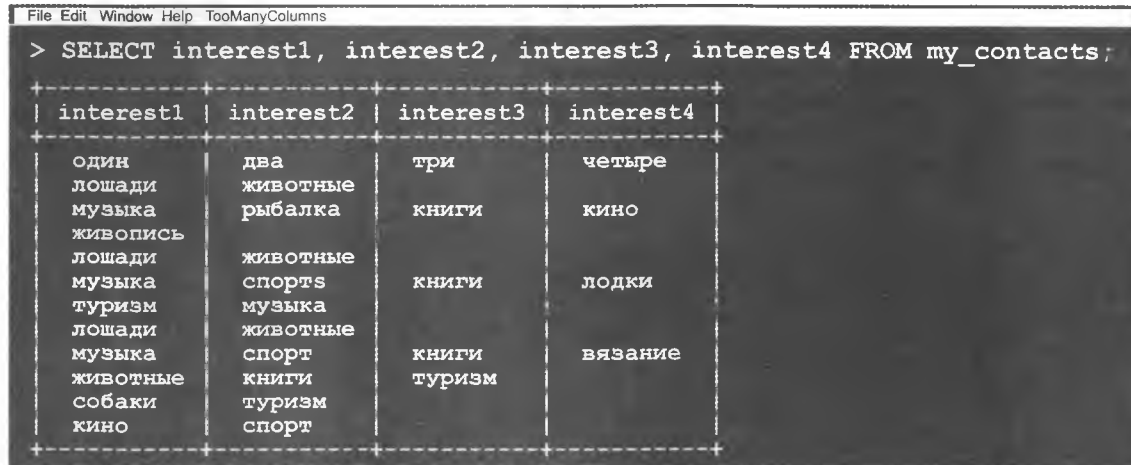
Заполните содержимое всех столбцов после выполнения большой команды.

interests	interest1	interest2	interest3	interest4
два, три, четыре	один			

ОТВЕТЫ на с. 408.

Вывод списка

Наконец-то все увлечения разделены по разным столбцам. Для вывода можно воспользоваться простой командой `SELECT` — но не для всех одновременно. И команда не позволит легко извлечь их в один итоговый набор, потому что увлечения хранятся в четырех столбцах. Результат будет выглядеть примерно так.



```
File Edit Window Help TooManyColumns
> SELECT interest1, interest2, interest3, interest4 FROM my_contacts;
```

interest1	interest2	interest3	interest4
один	два	три	четыре
лошади	животные	книги	кино
музыка	рыбалка		
живопись			
лошади	животные		
музыка	спорт	книги	лодки
туризм	музыка		
лошади	животные		
музыка	спорт	книги	вязание
животные	книги	туризм	
собаки	туризм		
кино	спорт		

Конечно, мы можем написать четыре отдельные команды `SELECT` для вывода всех значений:

```
SELECT interest1 FROM my_contacts;    SELECT interest3 FROM my_contacts;
SELECT interest2 FROM my_contacts;    SELECT interest4 FROM my_contacts;
```

Остается лишь понять, как вставить результат выполнения этих команд в новую таблицу. К счастью, это можно сделать, причем способ не один — их не менее трех!



Упражнение

Попробуйте сами

Вспомните команду `SELECT` для столбца `profession`, написанную нами на с. 375:

```
SELECT profession FROM my_contacts GROUP BY profession
ORDER BY profession;
```

На следующей странице представлены **ТРИ СПОСОБА** использования команд `SELECT` для автоматического заполнения новой таблицы `interests`.

Поразмыслите над командами `SELECT`, `INSERT` и `CREATE`. Затем переверните страницу и посмотрите описания трех способов.

Ваша задача — не угадать правильный синтаксис, а обдумать имеющиеся возможности.

Дороги, которые мы выбираем

Возможность сделать одно и то же тремя (и более) разными способами кому-то может показаться веселой, но нормальных людей такое изобилие обычно сбивает с толку.

И все же это полезно. Зная три решения одной задачи, вы всегда сможете выбрать то решение, которое лучше подходит для ваших потребностей. А по мере роста объема данных вы заметите, что некоторые запросы быстрее выполняются вашей РСУБД. Запросы к очень большим таблицам желательно оптимизировать, и умение решать одну задачу разными способами вам в этом поможет.

На ближайших страницах представлены все три способа создания и заполнения таблицы уникальными значениями, упорядоченными по алфавиту.

profession
prof_id 0 ←
profession



(Почти) одновременное выполнение CREATE, SELECT и INSERT

1. CREATE TABLE, затем INSERT с SELECT

Этот способ вам уже известен! Сначала таблица table создается командой CREATE, а затем столбцы заполняются значениями, возвращаемыми командой SELECT на с. 375.

```
CREATE TABLE profession  
(  
  id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  profession varchar(20)  
);
```

Создание таблицы profession со столбцом первичного ключа и столбцом VARCHAR для описаний профессий.

```
INSERT INTO profession (profession)  
SELECT profession FROM my_contacts  
GROUP BY profession  
ORDER BY profession;
```

Заполнение столбца «profession» таблицы profession значениями, выдаваемыми командой SELECT.

2. CREATE TABLE с SELECT, добавление первичного ключа командой ALTER

Второй способ: таблица `profession` создается командой `CREATE` с использованием данных команды `SELECT`, возвращающей значения столбца `profession` таблицы `my_contacts`. Затем таблица изменяется командой `ALTER` с добавлением первичного ключа.

```
CREATE TABLE profession AS
  SELECT profession FROM my_contacts
  GROUP BY profession
  ORDER BY profession;

ALTER TABLE profession
  ADD COLUMN id INT NOT NULL AUTO_INCREMENT FIRST,
  ADD PRIMARY KEY (id);
```

Создание таблицы `profession` с единственным столбцом. Таблица заполняется значениями, полученными от `SELECT`

...после чего команда `ALTER` добавляет в таблицу столбец первичного ключа.

Одновременное выполнение CREATE, SELECT и INSERT

3. CREATE TABLE с первичным ключом и SELECT в одной команде

А этот способ состоит всего из одного шага: команда `CREATE` создаст таблицу `profession` со столбцом первичного ключа и столбцом `VARCHAR` для хранения профессий, одновременно с заполнением последнего данными запроса `SELECT`. Условие `AUTO_INCREMENT` сообщает РСУБД, что значения столбца `id` должны генерироваться автоматически, а следовательно, данные будут направлены в единственный оставшийся столбец.

```
CREATE TABLE profession
(
  id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  profession varchar(20)
) AS
SELECT profession FROM my_contacts
GROUP BY profession
ORDER BY profession;
```

Создание таблицы `profession` с первичным ключом и столбцом «`profession`», которой немедленно заполняется данными от `SELECT`.

Я еще не видел ключевое слово `AS`. Похоже, оно используется для обозначения результатов запроса, вставляемых в новую таблицу.

Да, ключевое слово `AS` делает именно это.

Оно используется при определении псевдонимов, которыми мы сейчас займемся!



Зачем нужно AS?

AS заполняет новую таблицу результатами SELECT. Таким образом, при использовании AS во втором и третьем примерах мы указываем РСУБД, что данные, полученные из таблицы my_contacts в результате выполнения SELECT, следует поместить в только что созданную таблицу profession.

Если бы мы не указали, что новая таблица содержит два столбца с новыми именами, то условие AS создало бы всего один столбец с таким же типом данных и именем, как у столбца из команды SELECT.

Если бы мы не создали таблицу с двумя столбцами, то команда создала бы один столбец с таким же типом данных и именем, как у результата SELECT.

Создаем в новой таблице столбец типа VARCHAR с именем profession.

```
CREATE TABLE profession  
(  
  id INT(11) NOT NULL AUTO INCREMENT PRIMARY KEY,  
  profession varchar(20)  
) AS  
SELECT profession FROM my_contacts  
GROUP BY profession  
ORDER BY profession;
```

Это короткое ключевое слово играет важную роль: оно направляет весь вывод SELECT в новую таблицу.

Все эти имена относятся к столбцу «profession» таблицы my_contacts.

Так как таблица profession была создана с автоматически увеличиваемым первичным ключом, значения могут добавляться только во второй столбец таблицы, которому также присвоено имя profession.



Я совсем запуталась — «profession» встречается в этом запросе целых пять раз! Возможно, РСУБД отличает одно вхождение от другого, но как их различу я?

Для предотвращения путаницы столбцу можно назначить альтернативное имя.

Это одна из причин, по которым SQL позволяет назначать столбцам и таблицам временные имена, называемые *псевдонимами*.

Псевдонимы столбцов

Создать псевдоним очень просто. Он указывается после первого использования имени столбца в запросе с другим ключевым словом AS. Оно сообщает РСУБД, что столбец `profession` таблицы `my_contacts` может временно называться новым именем, чтобы пользователю было проще разобраться в происходящем.

Мы присвоим данным, выбранным из таблицы `my_contacts`, имя **`mc_prof`** (`mc` – сокращение от `my_contacts`).

```
CREATE TABLE profession
(
  id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  profession varchar(20)
) AS
SELECT profession AS mc_prof FROM my_contacts
GROUP BY mc_prof
ORDER BY mc_prof;
```

Укажите псевдоним после первого использования исходного имени столбца в запросе. Тем самым вы сообщаете своей РСУБД, что в дальнейшем к столбцу возможны обращения не только по имени, но и по псевдониму.

Этот запрос делает то же самое, но благодаря псевдониму в нем проще разобраться.

Между двумя запросами существует одно неочевидное различие. Все запросы возвращают результаты в форме таблиц. **Псевдоним изменяет имя столбца в результатах, но не изменяет исходного имени столбца.** Иначе говоря, псевдоним действует временно.

Но так как мы указали, что новая таблица состоит из двух столбцов (первичный ключ и столбец `profession`), столбцу таблицы будет присвоено имя `profession`, а не `mc_prof`.

profession
programmer
teacher
lawyer

Результаты исходного запроса с исходным именем столбца.

Результаты запроса, использующего псевдоним. Имя столбца совпадает с псевдонимом.

mc_prof
programmer
teacher
lawyer

Кому нужны псевдонимы таблиц?

Вам и нужны! Мы сейчас займемся соединениями с выборкой данных из нескольких таблиц. Без псевдонимов вам придется вводить имена таблиц снова и снова и вам это быстро надоест.

Псевдонимы таблиц создаются почти так же, как псевдонимы столбцов. Псевдоним таблицы указывается после первого использования имени таблицы в запросе с ключевым словом AS. В следующем примере оно сообщает, что таблица `my_contacts` в дальнейшем будет также доступна по имени `mc`.

```
SELECT profession AS mc_prof
FROM my_contacts AS mc
GROUP BY mc_prof
ORDER BY mc_prof;
```

Псевдонимы таблиц создаются так же, как и псевдонимы столбцов.

И я должен использовать «AS» каждый раз, когда потребуется создать псевдоним?

Нет, существует сокращенный синтаксис назначения псевдонимов.

Просто не указывайте ключевое слово AS. Следующий запрос делает то же самое, что и запрос в начале страницы.

Эти два запроса делают одно и то же.

```
SELECT profession mc_prof
FROM my_contacts mc
GROUP BY mc_prof
ORDER BY mc_prof;
```

Назначение псевдонима без ключевого слова AS. Псевдоним должен указываться сразу же после имени таблицы или столбца, с которым он связывается.

Псевдонимы таблиц также называются параллельными именами.



Все, что Вы хотели знать о внутренних соединениях

Каждый, кому доводилось слышать разговоры о SQL, наверняка слышал слово «соединение». Эта тема не так сложна, как может показаться на первый взгляд. Мы покажем вам, что такое соединения, как они работают, в каких ситуациях их следует применять и в какой ситуации применяется та или иная разновидность соединений.

Но начнем мы с рассмотрения простейшей разновидности соединений (которая и полноценным соединением-то не является!).

Она известна под разными именами. В этой книге мы будем называть ее **перекрестным соединением**, хотя также встречается термин «перекрестное произведение» и «декартово соединение».



...вот откуда на самом деле берутся таблицы результатов.

Предположим, имеются две таблицы: с именами детей и названиями игрушек, которые есть у этих детей. Ваша задача — узнать, какие игрушки можно подарить каждому ребенку.

toys

toy_id	toy
1	обруч
2	самолет
3	солдатики
4	губная гармошка
5	бейсбольные карточки

boys

boy_id	boy
1	Дейви
2	Бобби
3	Бивер
4	Ричи

Перекрестное соединение

Результат следующего запроса представляет собой перекрестное соединение. Мы запрашиваем данные из обеих таблиц: столбец toy из таблицы toys и столбец boy из таблицы boys.

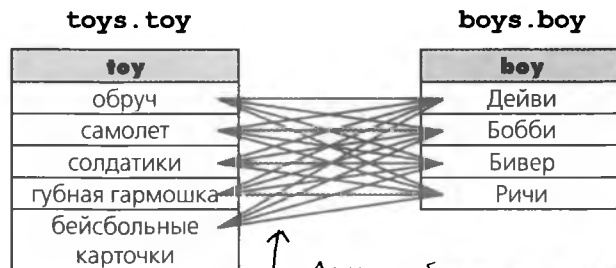
```
SELECT t.toy, b.boy
FROM toys AS t
CROSS JOIN
boys AS b;
```

Здесь тоже используются псевдонимы таблиц.

Помните сокращенную запись из предыдущей главы? Перед точкой указывается имя таблицы, после точки — имя столбца таблицы. Только на этот раз вместо полных имен таблиц используются псевдонимы.

Запрос читает данные из столбца «boy» таблицы boys и из столбца «toy» таблицы «toys». Оставшаяся часть запроса соединяет эти два столбца в новую таблицу.

Перекрестное соединение создает пару из каждого значения первой таблицы и каждого значения из второй таблицы.



Линии обозначают результаты соединения. Каждой игрушке ставится в соответствие каждый мальчик. Результат не содержит дубликатов.

Результат соединения состоит из 20 записей (5 игрушек * 4 мальчиков), то есть всех возможных комбинаций.

Такая группировка данных объясняется только тем, что toys.toy содержит больше записей. Если бы в таблице «boys» было 5 записей, а в таблице «toys» — 4 записи, то результаты группировались бы по именам мальчиков. Но помните, что порядок результатов для этого запроса не имеет значения.

toy	boy
обруч	Дэйви
обруч	Бобби
обруч	Бивер
обруч	Ричи
самолет	Дэйви
самолет	Бобби
самолет	Бивер
самолет	Ричи
самолет	Дэйви

Перекрестное соединение (CROSS JOIN) возвращает комбинации каждой записи первой таблицы с каждой записью второй таблицы.

Часто
**Задаваемые
 Вопросы**

В: И зачем мне это нужно?

О: О перекрестных соединениях важно знать, потому что при экспериментах с соединениями можно случайно получить перекрестный результат. Это поможет вам исправить неправильно написанный запрос. Поверьте, такое случается. Кроме того, перекрестные соединения иногда используются для тестирования скорости РСУБД и ее конфигурации. Их обработка занимает относительно много времени, что упрощает анализ и сравнения.

В: А если использовать запрос вида:
SELECT * FROM toys CROSS JOIN boys;
 Что произойдет при использовании **SELECT ***?

О: Попробуйте сами. Вы получите те же 20 записей, но в них будут включены все 4 столбца.

Внутренним соединением (INNER JOIN) называется перекрестное соединение, из результатов которого часть записей исключается по условию запроса.

В: Что произойдет при перекрестном соединении двух очень больших таблиц?

О: Вы получите *огромное количество записей*. С перекрестным соединением лучше не экспериментировать — при таком гигантском объеме возвращаемых данных ваш компьютер может «зависнуть»!

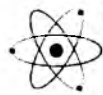
В: Существует ли другой синтаксис у таких запросов?

О: Да, существует. Вместо ключевых слов **CROSS JOIN** можно поставить запятую:

```
SELECT toys.toy, boys.boy
FROM toys, boys;
```

В: Ранее я слышал термины «внутреннее соединение» и «внешнее соединение». Это то же самое, что и перекрестное соединение?

О: Перекрестное соединение является разновидностью внутреннего соединения. В сущности, внутреннее соединение — это перекрестное соединение, из результатов которого некоторые записи исключены по критерию запроса. Внутренние соединения вскоре будут описаны более подробно — а пока просто запомните!



МОЗГОВОЙ ШТУРМ

Как вы думаете, какой результат вернет следующий запрос:

```
SELECT b1.boy, b2.boy
FROM boys AS b1 CROSS JOIN boys AS b2;
```

Попробуйте сами.

Возьми в руку карандаш



profession	
prof_id	🔑
profession	

my_contacts	
contact_id	🔑
last_name	
first_name	
phone	
email	
gender	
birthday	
prof_id	🔑
zip_code	🔑
status_id	🔑

Перед вами две таблицы из базы данных gregs_list: profession и my_contacts. Просмотрите код запроса и запишите, что, по вашему мнению, делает каждая строка.

```
SELECT mc.last_name,  
  
mc.first_name,  
  
p.profession  
  
FROM my_contacts AS mc  
  
INNER JOIN  
  
profession AS p  
  
ON mc.contact_id = p.prof_id;
```

Допустим, данные из трех карточек, приведенных ниже, занесены в таблицы.
Изобразите таблицу с результатами.

Джоан Эверетт
Не замужем
4-3-1978
Солт-Лейк-сити, УТ
Художник
Ж
jeverett@mightygumball.net
парусный спорт, туризм, кулинария
555 555-9870

Тара Болдуин
Замужем
9-1-1970
Бостон, МА
Шеф-повар
Ж
tara@breakneckpizza.com
кино, книги, кулинария
555 555-3432

Пол Сингх
Женат
12-10-1980
Нью-Йорк, NY
Профессор
М
ps@tikibeanlounge.com
собаки,
555 555-8222

Возьми в руку карандаш
Решение



profession
prof_id
profession

my_contacts
contact_id
last_name
first_name
phone
email
gender
birthday
prof_id
zip_code
status_id

Перед вами две таблицы из базы данных gregs_list: profession и my_contacts. Просмотрите код запроса и запишите, что, по вашему мнению, делает каждая строка.

SELECT mc.last_name, Выборка столбца «last_name» таблицы «my_contacts» (псевдоним mc)
mc.first_name, и столбца «first_name» таблицы my_contacts
p.profession и столбца «profession» таблицы profession (псевдоним p)
FROM my_contacts AS mc из таблицы my_contacts (псевдоним mc)
INNER JOIN внутреннее соединение результатов выборки
profession AS p с таблицей profession (псевдоним p)
ON mc.contact_id = p.prof_id; при условии, что значение столбца «contact_id» таблицы my_contacts совпадает со значением столбца «id» таблицы profession

Допустим, данные из трех карточек, приведенных ниже, занесены в таблицы. Изобразите таблицу с результатами.

last_name	first_name	profession
Эверетт	Джоан	Художник
Сингх	Пол	Профессор
Болдуин	Тара	Шеф-повар

Открой свое внутреннее соединение

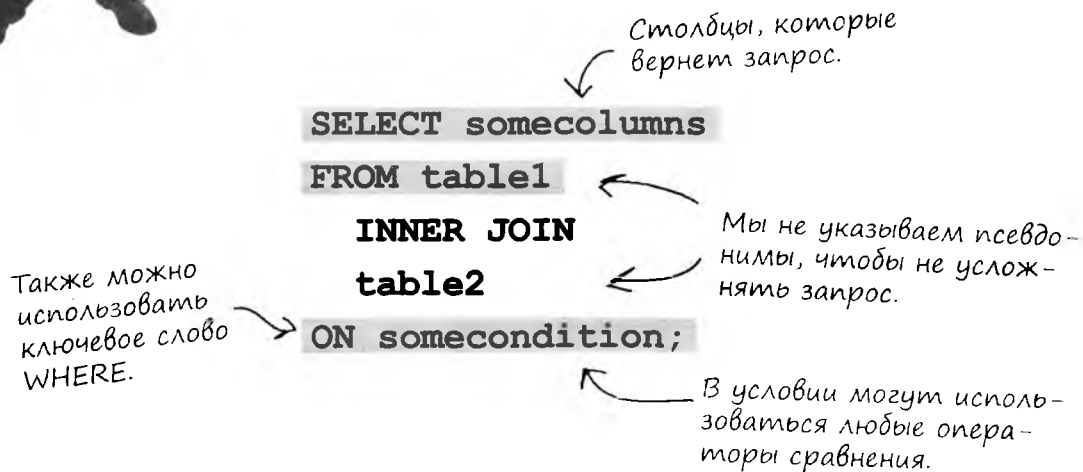


Понял! Так я могу связать новые таблицы с новой версией my_contacts. Мне не нужно писать десяток SELECT, достаточно включить таблицы во внутреннее соединение!

Все только начинается

Думаете, это все? Мы рассмотрели только одну разновидность одного типа соединений. И вам еще предстоит узнать много всего об этом и других видах соединений, прежде чем вы сможете эффективно и разумно применять их на практике.

Внутреннее соединение комбинирует записи двух таблиц в соответствии с заданным условием. Столбцы включаются в выходной набор только в том случае, если соединенная запись удовлетворяет условию. Давайте повнимательнее рассмотрим синтаксис.



Внутреннее соединение комбинирует записи из двух таблиц в соответствии с заданным условием.

Внутреннее соединение в действии: эквисоединение

Рассмотрим следующие таблицы. У каждого мальчика есть только одна игрушка. Связь относится к типу «один-к-одному», а `toy_id` – внешний ключ.

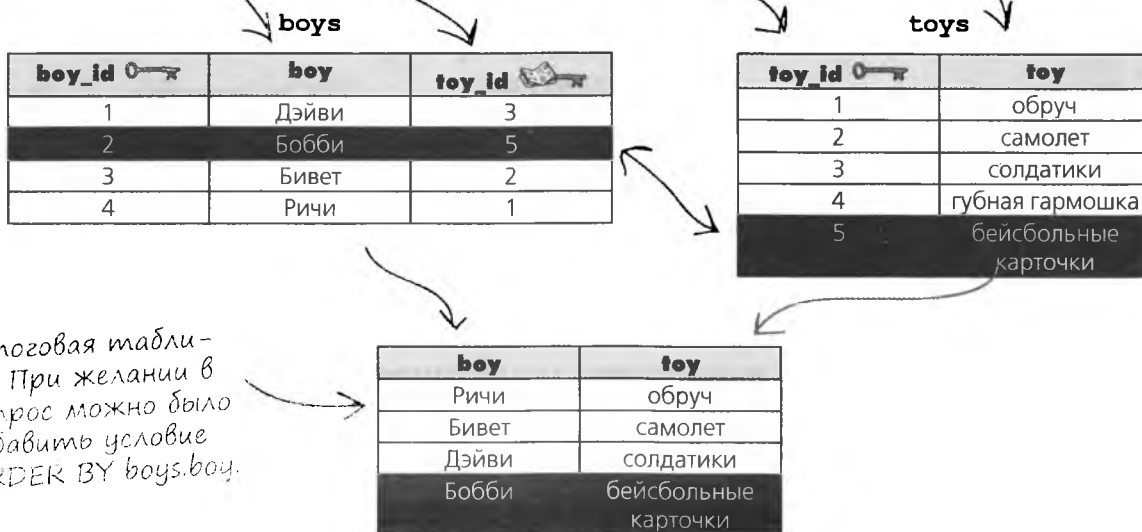
boy_id	boy	toy_id
1	Дэйви	3
2	Бобби	5
3	Бивер	2
4	Ричи	1

toy_id	toy
1	обруч
2	самолет
3	солдатики
4	губная гармошка
5	бейсбольные карточки

Все, что требуется – определить, какая игрушка принадлежит каждому из мальчиков. Мы можем воспользоваться внутренним соединением с оператором `=` для поиска совпадений внешнего ключа `boys` с первичным ключом `toys`.

```
SELECT boys.boy, toys.toy
FROM boys
INNER JOIN
toys
ON boys.toy_id = toys.toy_id;
```

Эквивалентное соединение – внутреннее соединение с проверкой равенства.



Итоговая таблица. При желании в запрос можно было добавить условие `ORDER BY boys.boy`.

Возьми в руку карандаш



Напишите следующие запросы эквивалентных соединений для базы данных gregs_list.

Запрос, который возвращает адреса электронной почты (email) и профессии (profession) каждого человека в my_contacts.

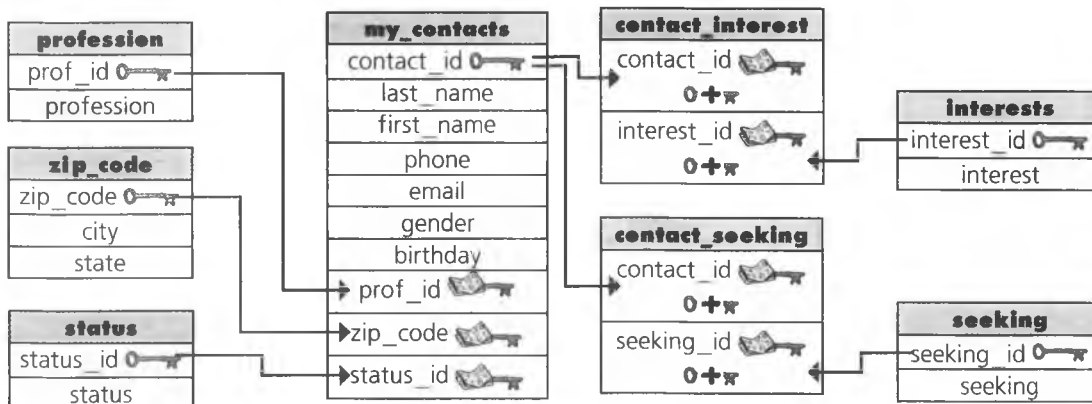
.....

Запрос, который возвращает имя (first_name), фамилию (last_name) и семейное положение (status) каждого человека в my_contacts.

.....

Запрос, который возвращает имя (first_name), фамилию (last_name) и штат (state) каждого человека в my_contacts.

.....



Возьми в руку карандаш Решение

Напишите следующие запросы эквивалентных соединений для базы данных gregs_list.

Запрос, который возвращает адреса электронной почты (email) и профессии (profession) каждого человека в my_contacts.

```
SELECT mc.email, p.profession FROM my_contacts mc
INNER JOIN profession p ON mc.prof_id = p.prof_id;
```

Внешний ключ prof_id связывается со столбцом «prof_id» таблицы profession.

Запрос, который возвращает имя (first_name), фамилию (last_name) и семейное положение (status) каждого человека в my_contacts.

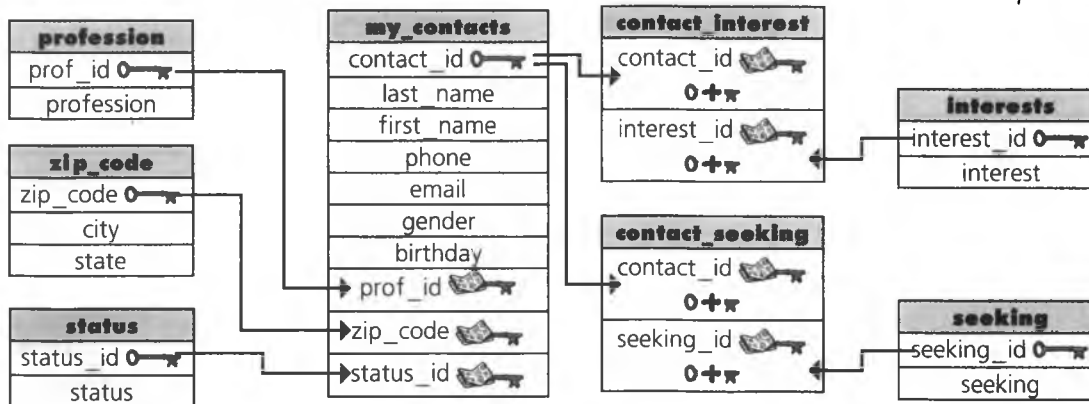
```
SELECT mc.first_name, mc.last_name, s.status FROM my_contacts mc
INNER JOIN status s ON mc.status_id = s.status_id;
```

Внешний ключ status_id связывается со столбцом «status_id» таблицы «status».

Запрос, который возвращает имя (first_name), фамилию (last_name) и штат (state) каждого человека в my_contacts.

```
SELECT mc.first_name, mc.last_name, z.state FROM my_contacts mc
INNER JOIN zip_code z ON mc.zip_code = z.zip_code;
```

На этот раз в качестве ключа, связывающего две таблицы, используется столбец «zip_code».



Внутреннее соединение в действии: неэквивалентное соединение

Неэквивалентное соединение возвращает записи, у которых заданные значения столбцов не равны. Для примера рассмотрим те же две таблицы, boys и toys. Используя неэквивалентное соединение, мы можем точно узнать, каких игрушек *нет* у каждого из мальчиков (такой результат более удобен при поиске подарка на день рождения).

```
SELECT boys.boy, toys.toy
FROM boys
  INNER JOIN
  toys
ON boys.toy_id <> toys.toy_id
ORDER BY boys.boy;
```

Упорядочение результатов упростит их чтение.

Оператор «не равно» — отсюда и название соединения.

boy_id	boy	toy_id
1	Дэйви	3
2	Бобби	5
3	Бивер	2
4	Ричи	1

toy_id	toy
1	обруч
2	самолет
3	солдатики
4	губная гармошка
5	бейсбольные карточки

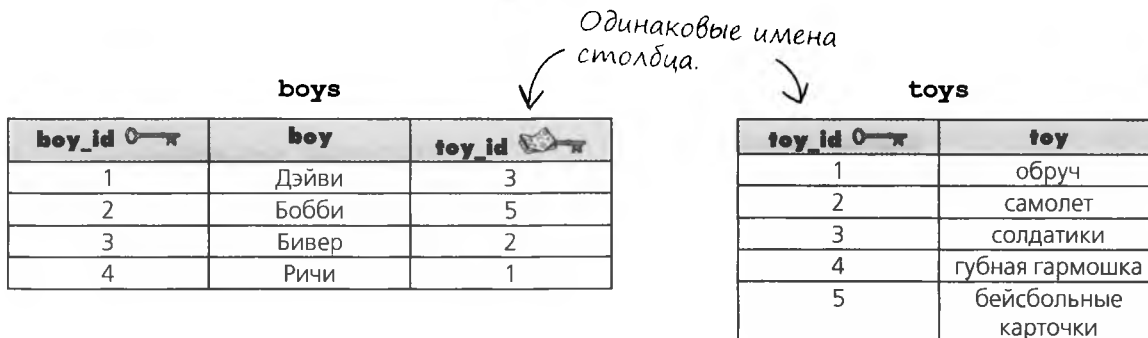
boy	toy
Бивер	обруч
Бивер	солдатики
Бивер	губная гармошка
Бивер	бейсбольные карточки
Бобби	солдатики
Бобби	губная гармошка
Бобби	обруч
Бобби	самолет
Дэйви	обруч
Дэйви	самолет
Дэйви	губная гармошка
Дэйви	бейсбольные карточки
Ричи	самолет
Ричи	солдатики
Ричи	губная гармошка
Ричи	бейсбольные карточки

Четыре игрушки, которых еще нет у Бивера.

Неэквивалентное соединение проверяет несовпадение значений.

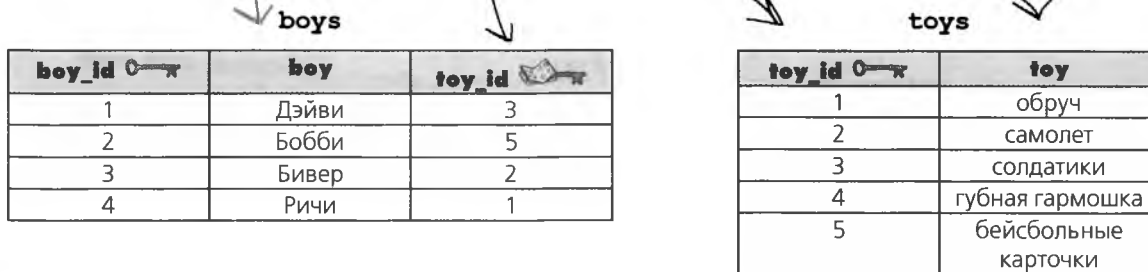
Последнее внутреннее соединение: естественное соединение

Осталась всего одна разновидность внутренних соединений — так называемые **естественные соединения**. Естественные соединения возможны только в том случае, если столбец, по которому выполняется соединение, имеет **одинаковые имена в обеих таблицах**. Давайте еще раз рассмотрим эти две таблицы.



Как и прежде, мы хотим знать, какая игрушка есть у каждого из мальчиков. Естественное соединение распознает совпадающие имена столбцов в двух таблицах и вернет соответствующие комбинации.

```
SELECT boys.boy, toys.toy
FROM boys
NATURAL JOIN
toys;
```



Мы получили тот же самый результат, как и в случае первого внутреннего соединения, — эквивалентное

boy	toy
Ричи	обруч
Бивер	самолет
Дэйви	солдатики
Бобби	губная гармошка

Естественное соединение связывает записи по значениям одноименных столбцов.

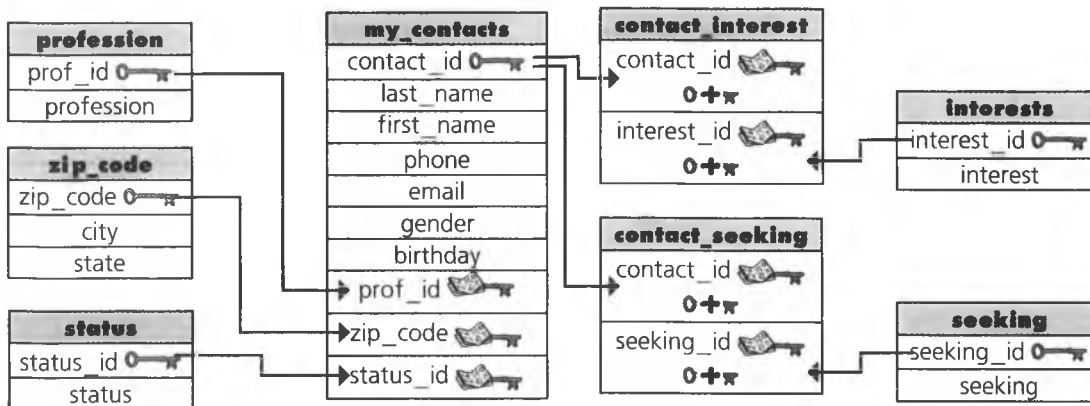
Возьми в руку карандаш

Напишите следующие запросы для базы данных `gregs_list` с использованием естественных или неэквивалентных соединений.

Запрос, который возвращает адреса электронной почты (`email`) и профессии (`profession`) каждого человека в `my_contacts`.

Запрос, который возвращает имя (`first_name`), фамилию (`last_name`) и семейное положение (`status`), которым не обладает каждый человек в `my_contacts`.

Запрос, который возвращает имя (`first_name`), фамилию (`last_name`) и штат (`state`) каждого человека в `my_contacts`.



Возьми в руку карандаш Решение



Напишите следующие запросы для базы данных gregs_list с использованием естественных или неэквивалентных соединений.

Запрос, который возвращает адреса электронной почты (email) и профессии (profession) каждого человека в my_contacts.

```
SELECT mc.email, p.profession FROM my_contacts mc
INNER JOIN profession p;
```

Запрос, который возвращает имя (first_name), фамилию (last_name) и семейное положение (status), которым не обладает каждый человек в my_contacts.

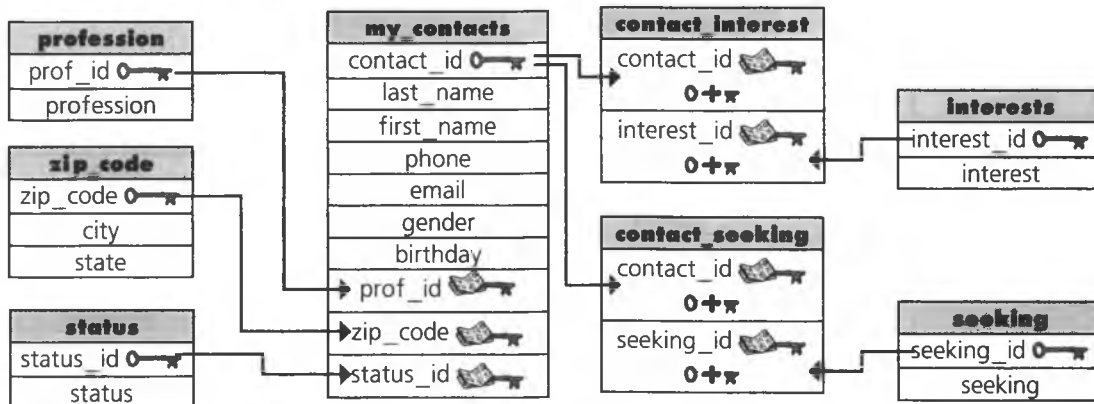
```
SELECT mc.first_name, mc.last_name, s.status FROM my_contacts mc
INNER JOIN status s ON mc.status_id <> s.status_id;
```

Для каждого человека будет создано несколько записей со всеми вариантами семейного положения, с которыми этот человек не связан по status_id.

Запрос, который возвращает имя (first_name), фамилию (last_name) и штат (state) каждого человека в my_contacts.

```
SELECT mc.first_name, mc.last_name, z.state FROM my_contacts mc
INNER JOIN zip_code z;
```

Условие ON в первом и третьем запросах не обязательно, потому что имена внешнего и первичного ключей в них совпадают.



* КТО И ЧТО ДЕЛАЕТ? *

Проведите линию от каждого вида соединения к его описанию. Некоторые виды соединений могут соответствовать сразу нескольким описаниям.

естественное соединение

Я возвращаю все записи, у которых значение столбца таблицы не совпадает со значением столбца другой таблицы.

эквивалентное соединение

Для меня важен порядок соединения таблиц.

перекрестное соединение

Я возвращаю все записи, у которых значение столбца таблицы совпадает со значением столбца другой таблицы, и при этом использую ключевое слово ON.

внешнее соединение

Я соединяю две таблицы, содержащие одноименные столбцы.

неэквивалентное соединение

Количество возвращаемых мною записей может быть равно произведению количества записей двух таблиц.

внутреннее соединение

Я возвращаю все возможные комбинации без проверки условия.

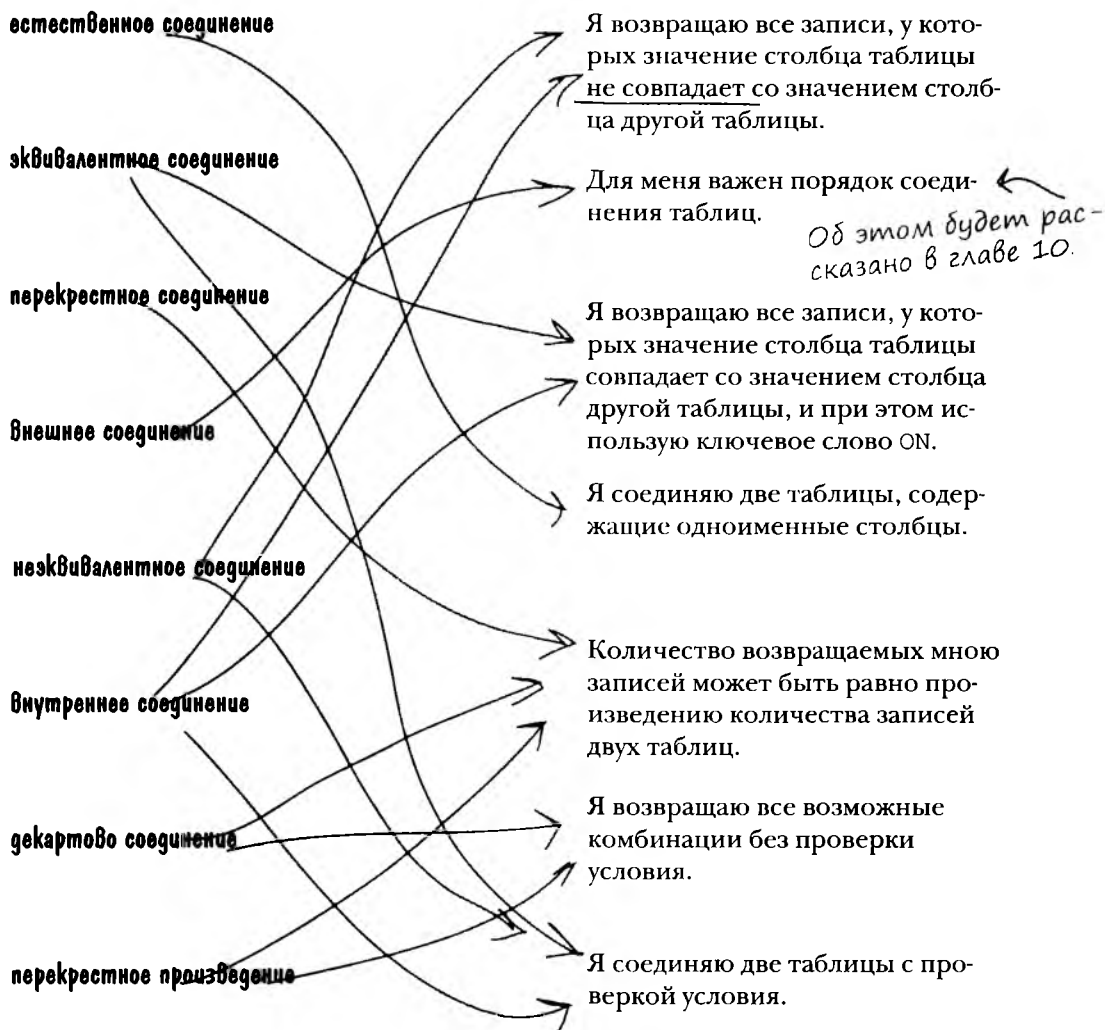
декартово соединение

перекрестное произведение

Я соединяю две таблицы с проверкой условия.

* КТО И ЧТО ДЕЛАЕТ? *

Проведите линию от каждого вида соединения к его описанию. Некоторые виды соединений могут соответствовать сразу нескольким описаниям.





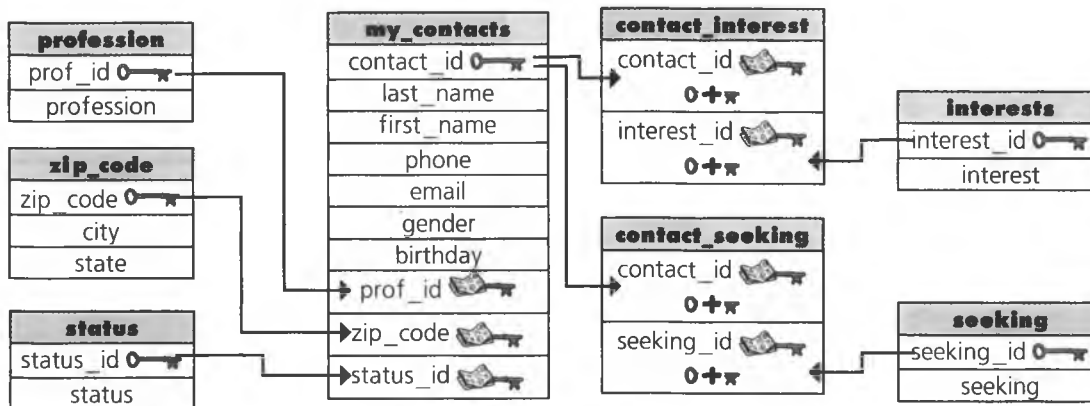
Упражнение

Для приведенной ниже схемы базы данных `gregs_list` напишите запросы SQL, возвращающие указанную информацию.

Напишите два запроса с разными соединениями для получения парных записей из таблиц `my_contacts` и `contact_interest`.

Напишите запрос для получения всех возможных комбинаций записей из таблиц `contact_seeking` и `seeking`.

Получите список профессий людей из таблицы `my_contacts`, но без дубликатов и в алфавитном порядке.





Для приведенной ниже схемы базы данных gregs_list напишите запросы SQL, возвращающие указанную информацию.

Напишите два запроса с разными соединениями для получения парных записей из таблиц my_contacts и contact_interest.

```
SELECT mc.first_name, mc.last_name, ci.interest_id FROM my_contacts mc
INNER JOIN contact_interest ci ON mc.contact_id = ci.contact_id;
```

```
SELECT mc.first_name, mc.last_name, ci.interest_id FROM my_contacts mc
NATURAL JOIN contact_interest ci;
```

Напишите запрос для получения всех возможных комбинаций записей из таблиц contact_seeking и seeking.

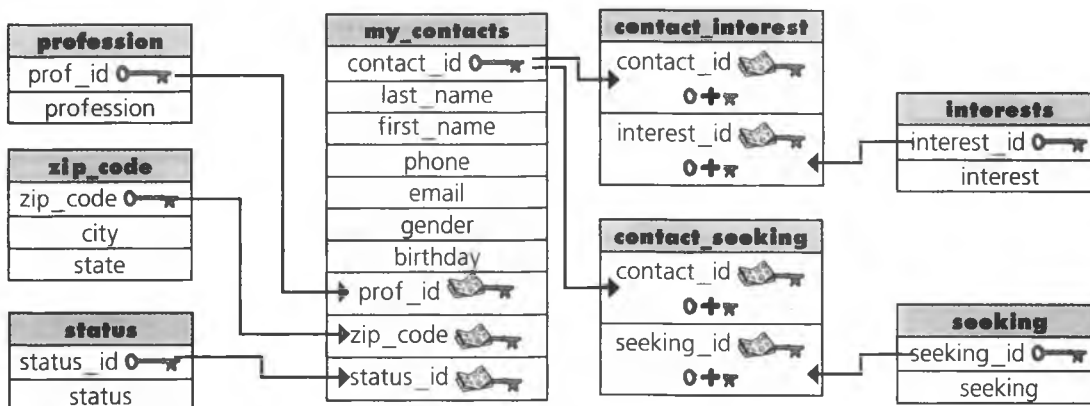
```
SELECT * FROM contact_seeking CROSS JOIN seeking;
```

```
SELECT * FROM contact_seeking, seeking;
```

Два способа выполнения одного перекрестного соединения.

Получите список профессий людей из таблицы my_contacts, но без дубликатов и в алфавитном порядке.

```
SELECT p.profession FROM my_contacts mc
INNER JOIN profession p ON mc.prof_id = p.prof_id GROUP BY profession
ORDER BY profession;
```



Часть
**Задаваемые
 Вопросы**

В: Можно ли включить в соединение более двух таблиц?

О: Можно, но об этом чуть позднее. Пока нас интересуют только общие концепции соединений.

В: Вроде как соединения считаются сложной темой?

О: С псевдонимами и соединениями запросы SQL все меньше напоминают текст на естественном языке. В сокращенной записи (скажем, при замене ключевых слов INNER JOIN запятыми) они выглядят еще более запутанными. По этой причине в книге в основном используются более понятные, а не более компактные запросы.

В: Значит ли это, что существуют другие варианты синтаксиса внутренних соединений?

О: Да, существуют. Но если вы поймете описанный нами синтаксис внутренних соединений, разобраться с другими будет намного проще. Сами концепции намного важнее, чем тонкости использования WHERE или ON.

В: Вы использовали в соединении конструкцию ORDER BY. Означает ли это, что в соединениях можно использовать и другие конструкции?

О: Да, в соединениях можно использовать конструкции GROUP BY, WHERE, а также функции SUM, AVG и т. д.

Встроенные запросы?

Грег постепенно начинает понимать возможности соединений. Он видит, что разбиение базы данных на таблицы имеет смысл, а работать с хорошо спроектированными таблицами не так уж сложно. Грег даже планирует расширить базу данных `gregs_list`.



Но мне по-прежнему часто приходится вводить один запрос, а потом использовать его результаты на входе другого запроса, хотя удобнее было бы разместить один запрос внутри другого. Но это только мечты...

Запрос внутри другого запроса?
 Такое возможно?



ОТКРОВЕННО О ПСЕВДОНИМАХ ТАБЛИЦ И СТОЛБЦОВ

Интервью недели: Что они скрывают?

Head First: Добро пожаловать, Псевдоним Таблицы и Псевдоним Столбца. Мы рады, что вы сегодня с нами. Надеемся, вы поможете нам прояснить некоторое недопонимание.

Псевдоним Таблицы: Еще бы, я тоже очень рад. И вы можете для краткости называть нас ПТ и ПС во время этого интервью (смеется).

Head First: Ха-ха! Да, это будет уместно. Итак, ПС, начнем с вас. Для чего такая секретность? Вы что-то пытаетесь скрыть?

Псевдоним Столбца: Вовсе нет! Если уж на то пошло, я стараюсь все прояснить. Ведь я сейчас говорю за нас обоих — верно, ПТ?

ПТ: Конечно. В случае ПС и так понятно, что он старается сделать: он берет длинные или избыточные имена столбцов и упрощает работу с ними. Просто для удобства. Кроме того, он предоставляет таблицы результатов с понятными именами столбцов. Со мной дело обстоит немного иначе.

Head First: Надо признать, мы не настолько хорошо знакомы с вами, ПТ. Мы видели, как вы работаете, но еще не до конца понимаем, что именно вы делаете. Ведь когда вас используют в запросах, вы не отображаетесь в результатах.

ПТ: Да, это правда. Но по-моему, вы не улавливаете моего более высокого предназначения.

Head First: Высокого предназначения? Интересно, продолжайте.

ПТ: Я существую для того, чтобы упростить написание запросов.

ПС: И еще ты помогаешь мне в соединениях, ПТ.

Head First: Ничего не понимаю. Может, приведете пример?

ПТ: Давайте рассмотрим синтаксис. Думаю, вам будет предельно понятно, что я делаю:

```
SELECT mc.last_name, mc.first_name,
       p.profession
FROM my_contacts AS mc
      INNER JOIN
      profession AS p
WHERE mc.contact_id = p.id;
```

Head First: Понятно! Повсюду, где мне пришлось бы вводить `my_contacts`, достаточно ввести `mc`. А `profession` заменяется на `p`. Так гораздо проще и намного удобнее, когда мне приходится включать два имени столбцов в один запрос.

ПТ: Особенно когда таблицы имеют похожие имена. Упрощение помогает не только написать пужный запрос, но и понять его, когда вы вернетесь к нему через какое-то время.

Head First: Большое спасибо, ПТ и ПС. Нам было очень... э... куда они пропали?



Новые инструменты

После главы 8 вы можете строить соединения, как настоящий SQL-профессионал. Ниже перечислены основные понятия этой главы. Полный список инструментов приведен в приложении III.

Внутреннее соединение

Любое соединение, комбинирующее записи двух таблиц по некоторому условию.

Естественное соединение

Внутреннее соединение без «ON». Работает только при соединении двух таблиц, содержащих одноименные столбцы.

Эквивалентное и неэквивалентное соединение

Две разновидности внутренних соединений. Эквивалентное соединение возвращает комбинации с равными значениями, а неэквивалентные — с неравными значениями столбцов.

Перекрестное соединение

Набор всех комбинаций записей одной таблицы с записями другой таблицы. Также встречаются другие названия — «декартово соединение», «декартово произведение» и др.

Соединение через запятую

То же, что перекрестное соединение — за исключением того, что ключевые слова CROSS JOIN заменяются запятой.

Возьми в руку карандаш



Решение
Со с. 378.

Вы уже умеете пользоваться командой ALTER; создайте в таблице my_contacts четыре новых столбца. Присвойте им имена interest1, interest2, interest3 и interest4.

```
ALTER TABLE my_contacts
ADD (interest1 VARCHAR(20), interest2 VARCHAR(20),
interest3 VARCHAR(20), interest4 VARCHAR(20));
```

Возьми в руку карандаш



Решение
Со с. 380.

Заполните пропуски в команде update. Мы привели пару подсказок, чтобы немного упростить вашу задачу.

Не путайте SUBSTRING_INDEX с SUBSTR: функция SUBSTRING_INDEX ищет заданный текст (в данном случае запятую) *внутри* столбца «interests» и возвращает все предшествующие символы. Функция SUBSTR усекает столбец «interests» до текста, следующего за первым увлечением, запятой и пробела (+2) до конца строки.

```
UPDATE my_contacts SET
interest1 = SUBSTRING_INDEX(interests, ',', 1),
interests = SUBSTR(interests, LENGTH(interest1)+2),
interest2 = SUBSTRING_INDEX(interests, ',', 1),
interests = SUBSTR(interests, LENGTH(interest2)+2),
interest3 = SUBSTRING_INDEX(interests, ',', 1),
interests = SUBSTR(interests, LENGTH(interest3)+2),
interest4 = interests;
```

После выполнения команды столбец «interests» остается пустым.

После удаления первых трех увлечений из столбца «interests» остается только четвертое увлечение. Эта команда просто переносит его в новый столбец. Также вместо этого можно было бы переименовать столбец «interests» в «interest4».

interests	interest1	interest2	interest3	interest4
second, third, fourth	first	second	third	fourth

Запросы внутри запросов



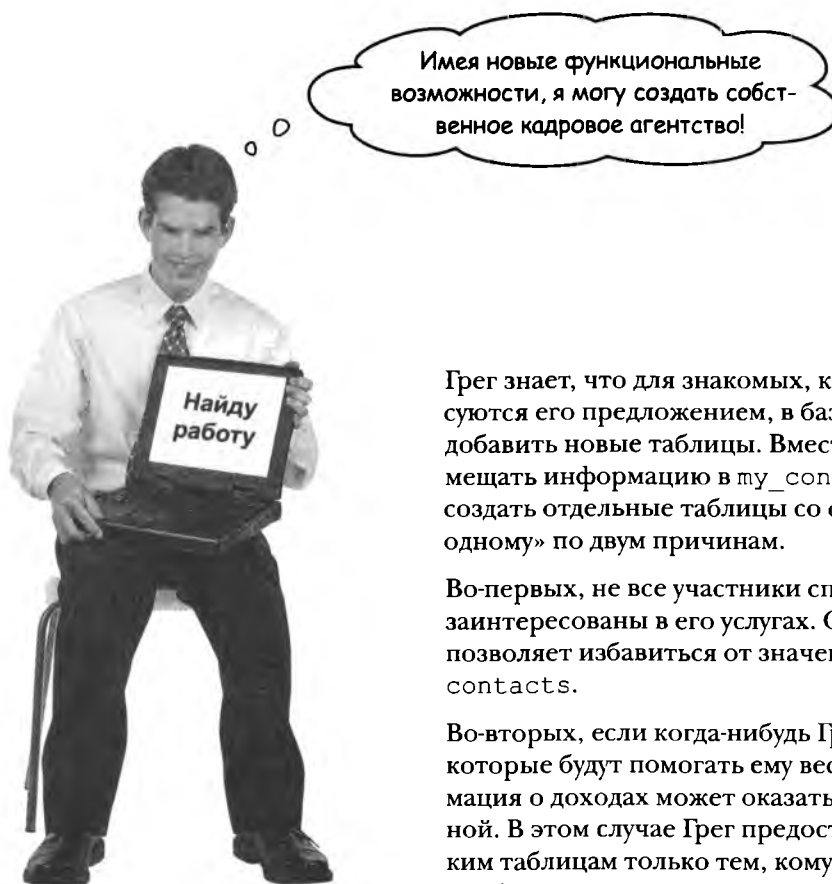
И все заметят, что я полна... (Как это называется? Утонченность? Изысканность? Элегантность?)

Мне, пожалуйста, запрос из двух частей. Соединения — хорошая штука, но иногда возникает необходимость обратиться к **базе данных сразу с несколькими вопросами**. Или **взять результат одного запроса и использовать его в качестве входных данных другого запроса**. В этом вам помогут **подзапросы**, также называемые **подчиненными запросами**. Они предотвращают **дублирование данных**, делают **запросы более динамичными** и даже помогут вам попасть на вечеринку в высшем обществе. (А может, и нет — но два из трех тоже неплохо!)

Грег берется за поиски работы

До настоящего момента база данных `gregs_list` была сугубо бескорыстным делом. Она помогала Грегу подбирать пары для своих друзей, но заработка не приносила.

Внезапно Грег сообразил, что он мог бы открыть собственное кадровое агентство, в котором подбирал бы людям из своего списка различные варианты работы.



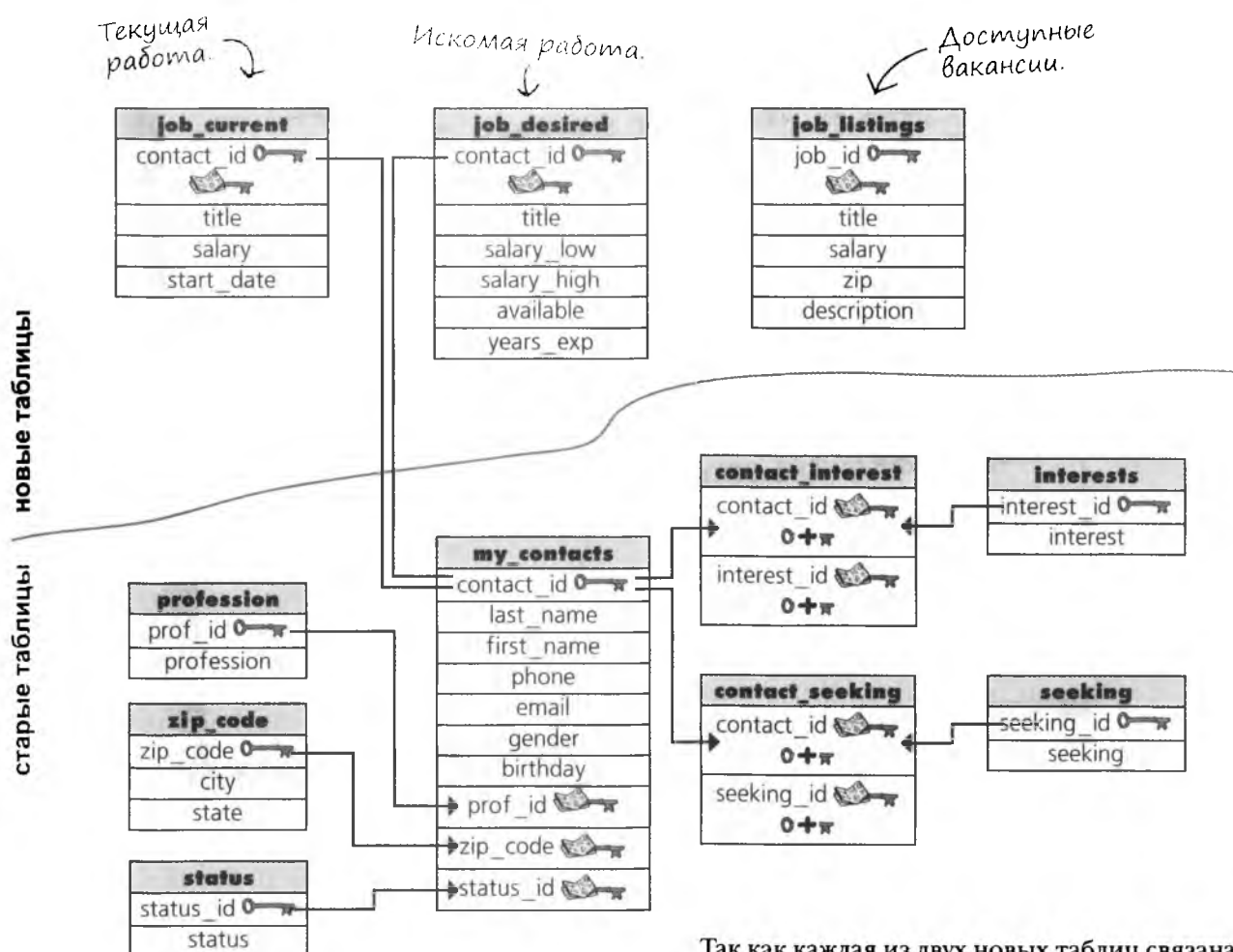
Грег знает, что для знакомых, которые заинтересуются его предложением, в базу данных придется добавить новые таблицы. Вместо того чтобы размещать информацию в `my_contacts`, он решает создать отдельные таблицы со связями «один-к-одному» по двум причинам.

Во-первых, не все участники списка `my_contacts` заинтересованы в его услугах. Отдельная таблица позволяет избавиться от значений `NULL` в `my_contacts`.

Во-вторых, если когда-нибудь Грег наймет людей, которые будут помогать ему вести бизнес, информация о доходах может оказаться конфиденциальной. В этом случае Грег предоставит доступ к таким таблицам только тем, кому он действительно необходим.

В списке Грега появляются новые таблицы

Грег добавил в свою базу данных новые таблицы для хранения информации об ожидаемой должности и диапазоне заработка, а также текущей должности и зарплате. Также Грег создает простую таблицу для хранения информации об имеющихся вакансиях.



Так как каждая из двух новых таблиц связана с my_contacts связью типа «один-к-одному», для получения данных очень удобно использовать естественные соединения.

Грег использует Внутреннее соединение

Грег получил информацию об отличной вакансии и теперь пытается найти кандидатов на нее в своей базе данных. Он хочет найти наилучшее совпадение, поскольку в случае найма его кандидата он получит премиальные.

Требуется: Веб-разработчик

Компания примет на работу веб-разработчика с отличным знанием HTML & CSS для работы с группой визуального дизайна. Специалисту, хорошо разбирающемуся в веб-стандартах, предоставляется уникальная возможность проявить себя в чрезвычайно перспективной компании, которой руководят умные люди, любящие свое дело.

Зарплата: \$95 000-\$10 5000

Опыт работы: 5+ лет

Когда Грег найдет нескольких кандидатов, он сможет обзвонить их и провести дальнейший отбор. Но сначала необходимо найти в базе данных всех веб-разработчиков с опытом работы не менее 5 лет, запрашивающих не более 10 5000.

Возьми в руку карандаш



Напишите запрос для выборки из базы данных кандидатов, удовлетворяющих поставленным условиям.

job_current
contact_id
title
salary
start_date

Наименьшая зарплата, на которую согласен кандидат.

Зарплата, которую кандидат надеется получать на новой работе.

job_desired
contact_id
title
salary_low
salary_high
available
years_exp

job_listings
job_id
title
salary
zip
description

Но он хочет опробовать другие запросы

Пока у Грега больше вакансий, чем претендентов. Он намерен провести поиск по таблице `professions` и выяснить, удастся ли ему найти совпадения для открытых вакансий. Далее он собирается выполнить естественное соединение с таблицей `my_contacts`, получить контактные данные и узнать, заинтересуются ли кандидаты его предложением.

Сначала он получает все вакантные должности из таблицы `job_current`.

```
SELECT title FROM job_listings  
GROUP BY title ORDER BY title;
```

Мы используем конструкцию `GROUP BY`, чтобы для каждой должности возвращалась только одна запись. Кроме того, данные упорядочиваются по алфавиту.

Результаты.

title
Веб-дизайнер
Веб-разработчик
Официант
Парикмахер
Повар

Лишь малая часть должностей в таблице `job_current`.

Возьми в руку карандаш



Напишите запрос для выборки из базы данных кандидатов, удовлетворяющих поставленным условиям.

```
SELECT mc.last_name, mc.first_name, mc.phone
```

```
FROM my_contacts AS mc
```

```
NATURAL JOIN
```

```
job_desired AS jd
```

```
WHERE jd.title = 'Веб-разработчик'
```

```
AND jd.salary_low < 105000;
```

Нас интересуют контактные данные людей, ищущих работу веб-разработчика.

Так как в таблицах `my_contacts` и `job_desired` в качестве первичного ключа используется столбец «`contact_id`», их можно связать при помощи естественного соединения.

Нас интересуют только люди, готовые работать за предложенную сумму. По столбцу «`salary_low`» мы убеждаемся в том, что предложенная зарплата не ниже запрашиваемого минимума.

А теперь Грег использует ключевое слово IN, чтобы узнать, имеются ли кандидаты на эти должности среди его подопечных.

```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title
FROM job_current AS jc NATURAL JOIN my_contacts AS mc
WHERE
jc.title IN ('Веб-дизайнер', 'Веб-разработчик', 'Официант', 'Парикмахер', 'Повар');
```

Помните ключевое слово IN? С ним запрос возвращает запись в том случае, если значение jc.title входит в группу значений в круглых скобках.

Результаты первого запроса.

← Работает

mc.first_name	mc.last_name	mc.phone	jc.title
Джо	Лонниган	(555) 555-3214	Повар
Венди	Хиллерман	(555) 555-8976	Официант
Шон	Миллер	(555) 555-4443	Веб-дизайнер
Джаред	Колуэй	(555) 555-5674	Веб-разработчик
Хуан	Гарза	(555) 555-0098	Веб-разработчик

Но нам все равно придется вводить два разных запроса...



Попробуйте объединить два запроса в один. Запишите здесь этот запрос.

Подзапросы

Чтобы сделать то, что делается двумя запросами, всего в одном запросе, нам понадобится включить в него **подзапрос**.

Второй запрос, в котором извлекаются совпадения из таблицы `professions`, мы назовем **ВНЕШНИМ** запросом, потому что в него «упакован» другой, **ВНУТРЕННИЙ** запрос. Давайте посмотрим, что происходит.

ВНЕШНИЙ запрос

```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title
FROM job_current AS jc NATURAL JOIN my_contacts AS mc
WHERE
jc.title IN ('Sales Representative', 'Sales Representative', 'Sales Representative', 'Sales Representative');
```

Это внешний запрос.

Эту часть можно удалить и заменить частью первого запроса, которая станет внутренним запросом.

Все профессии из списка в скобках были получены в результате *первого* запроса — того, который выбирал все вакантные должности из таблицы `job_current`. Таким образом (будьте внимательны, это тонкий момент!), мы можем **заменить эту часть внешнего запроса частью первого запроса**. Он по-прежнему вернет результаты, перечисленные в скобках, но на этот раз будет инкапсулирован в подзапросе:

ВНУТРЕННИЙ запрос

```
SELECT title FROM job_listings
```

← Эта часть первого запроса превращается во внутренний запрос (или подзапрос).

Подзапрос представляет собой запрос, «упакованный» в другом запросе. Также он может называться «внутренним запросом».

Два запроса преобразуются в запрос с подзапросом

Фактически мы всего лишь объединяем два запроса в один. Первый запрос называется **внешним**, а второй – **внутренним**.

ВНЕШНИЙ запрос

+

ВНУТРЕННИЙ запрос

=

Запрос с подзапросом

Внешний
запрос

```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title
FROM job_current AS jc NATURAL JOIN my_contacts AS mc
WHERE jc.title IN (SELECT title FROM job_listings);
```

Два объединенных за-
проса превращаются
в запрос с подзапросом.

Нам больше не придется вводить вруч-
ную все должности, возвращаемые пер-
вым запросом, потому что внутренний
запрос подставит данные за нас!

Результаты, полученные при выполнении этого запроса, в точности совпадают с результатами при явном перечислении *всех вакансий* в условии WHERE, но набирать приходится намного меньше.

Те же результаты, но всего
с одним запросом!

mc.first_name	mc.last_name	mc.phone	jc.title
Джо	Лонниган	(555) 555-3214	Повар
Венди	Хиллерман	(555) 555-8976	Официант
Шон	Миллер	(555) 555-4443	Веб-дизайнер
Джаред	Колуэй	(555) 555-5674	Веб-разработчик
Хуан	Гарза	(555) 555-0098	Веб-разработчик

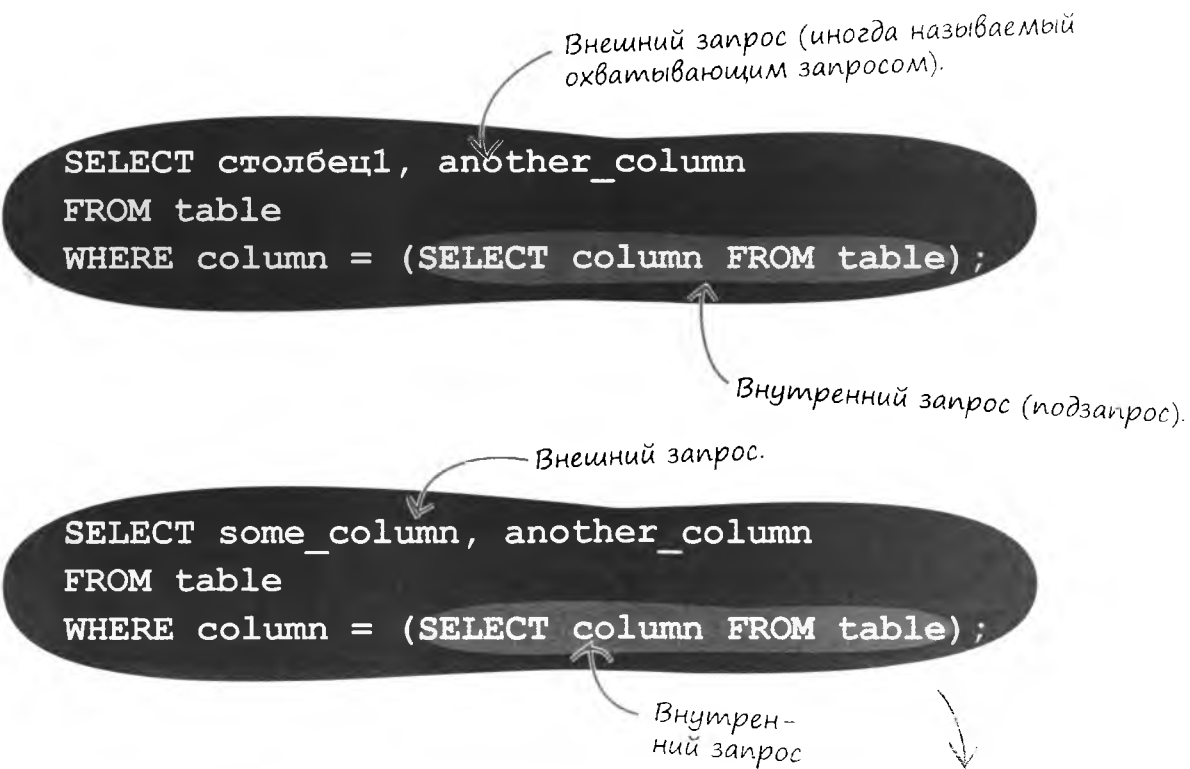


Код под микроскопом

Подзапросы: если одного запроса недостаточно

Подзапрос — не что иное, как запрос внутри другого запроса.

«Охватывающий» запрос называется внешним, а «вложенный» — внутренним запросом, или подзапросом.



Так как подзапрос использует оператор =, он возвращает *одно значение*, одну запись из одного столбца (иногда называется «ячейкой», но в SQL используется термин *скалярное значение*). Это значение сравнивается со столбцами в условии WHERE.

значение

Наш подзапрос возвращает скалярное значение (один столбец, одна запись), которое затем сравнивается со столбцами в условии WHERE.

Подзапрос в действии

Давайте посмотрим, как работает аналогичный запрос к таблице `my_contacts`. РСУБД читает скалярное значение из таблицы `zip_code` и сравнивает его со столбцами в условии `WHERE`.

```
(SELECT zip_code FROM
zip_code WHERE city =
'Мемфис' AND state = 'TN')
```

значение

```
SELECT last_name, first_name
FROM my_contacts
WHERE zip_code = (SELECT zip_code FROM
zip_code WHERE city =
'Мемфис' AND state = 'TN')
```

Запрос выбирает из `my_contacts` имена людей, живущих в Мемфисе (штат Теннесси).

Часто Задаваемые Вопросы

В: Почему то же самое нельзя сделать с использованием соединения?

О: Можно, но некоторые люди считают, что работать с подзапросами проще, чем с соединениями. Хорошо иметь свободу выбора синтаксиса.

Тот же запрос можно реализовать следующим образом:

```
SELECT last_name, first_name
FROM my_contacts mc
NATURAL JOIN zip_code zc
WHERE zc.city = 'Мемфис'
AND zc.state = 'TN'
```


Беседа у камина



Внутренний или внешний?

Внешний запрос

Знаешь, Внутренний Запрос, ты мне вообще-то не нужен. Я прекрасно обойдусь и без тебя.

Да, конечно. Ты даешь мне один маленький результат, а пользователям нужны данные, и притом МНОГО. Я даю им эти данные. Думаю, если бы тебя не было, это бы их вполне устроило.

Не придется, если добавить условие WHERE.

Нужен, еще как. Какая польза от одного столбца одной записи? Он просто не содержит достаточной информации.

Конечно, но я работаю сам по себе.

Внутренний запрос

Да и я без тебя обойдусь. Думаешь, это так весело — давать тебе конкретный, точный результат только для того, чтобы ты превратил его в набор подходящих записей? Количество не заменяет качество, знаешь ли.

Нет, я придаю твоим результатам некое подобие специализации. Без меня тебе придется возиться со всеми данными в таблице.

Я И ЕСТЬ твоё условие WHERE, и притом предельно конкретное. Собственно, ты мне не так уж нужен.

Ладно. Возможно, нам все же стоит работать вместе. Я определяю направление поиска твоих результатов.

Как и я.



Упражнение

Правила для подзапросов

Ниже перечислены некоторые правила, которым должны удовлетворять подзапросы. Заполните пропуски словами из следующего набора (некоторые слова могут использоваться многократно).

SELECT **ТОЧКА С ЗАПЯТОЙ** **СПИСОК СТОЛБЦОВ** **КОНЕЦ**
UPDATE **КРУГЛЫЕ СКОБКИ** **INSERT** **DELETE**

Кодекс SQL

Подзапрос всегда представляет собой одну команду

Подзапросы всегда заключаются в

Подзапросы не имеют собственного символа

..... Как обычно, один такой символ обозначает всего запроса.

Кодекс SQL

Подзапросы могут использоваться в четырех местах запросов:, **SELECT** как один из столбцов, условие и условие

Подзапросы могут использоваться в командах,, и, конечно,



Упражнение
Решение

Правила подзапросов

Помните об этих правилах, когда будете рассматривать примеры подзапросов в этой главе.

Когекс SQL

Подзапрос всегда представляет собой одну команду **SELECT**.

Подзапросы всегда заключаются в **КРУГЛЫЕ СКОБКИ**.

Подзапросы не имеют собственного символа **ТОЧКА С ЗАПЯТОЙ**. Как обычно, один такой символ обозначает **КОНЕЦ** всего запроса.

Когекс SQL

Подзапросы могут использоваться в четырех местах запросов: **SELECT**, **SELECT СПИСОК СТОЛБЦОВ** как один из столбцов, условие **FROM** и условие **WHERE**.

Подзапросы могут использоваться в командах **INSERT**, **DELETE**, **UPDATE** и, конечно, **SELECT**.

Часто задаваемые вопросы

В: Так что же может возвращать внутренний запрос? А как насчет внешнего запроса?

О: В большинстве случаев внутренний запрос может возвращать только одно значение, то есть один столбец с одной записью. Внешний запрос берет это значение и использует его для сравнения со всеми значениями столбца.

В: Почему вы говорите об «одном значении», когда в примере на с. 418 возвращается целый столбец значений?

О: Потому что оператор **IN** просматривает набор значений. При использовании оператора сравнения (как **=** в разделе «Анатомия») имеется всего одно значение, которое сравнивается с каждым значением в столбце.

В: Мне все еще неясно, сколько значений может возвращать подзапрос — одно или несколько. Что по этому поводу сказано в официальных правилах?

О: В общем случае подзапрос должен возвращать одно значение. **IN** — исключение из правила. Обычно подзапросы возвращают одно значение.

В: Что произойдет, если подзапрос возвращает несколько значений, но не использует условие **WHERE** с набором значений?

О: Хаос и разрушение!.. На самом деле вы просто получите сообщение об ошибке.

Правила — это, конечно, хорошо, но я хочу знать, как мне избавиться от длинных имен в столбцах результатов — таких, как mc.last_name. На этот счет есть какие-нибудь правила?



Существует два способа упрощения результатов.

Вы можете определить псевдонимы для своих столбцов в списке SELECT. Возвращаемая таблица сразу становится намного более понятной.

Вот как выглядит только что созданный нами подзапрос с короткими псевдонимами столбцов.

Столбцу «first_name» таблицы my_contacts в результатах запроса назначается псевдоним «firstname»...

...а столбцу «last_name» таблицы my_contacts назначается псевдоним «lastname».

```
SELECT mc.first_name AS firstname, mc.last_name AS lastname, mc.phone AS phone, jc.title AS jobtitle
```

```
FROM job_current AS jc NATURAL JOIN my_contacts AS mc WHERE jobtitle IN (SELECT title FROM job_listings);
```

Столбцу «phone» таблицы my_contacts в результатах назначается псевдоним «phone» и так далее. В общем, вы поняли!

Не забывайте: ключевое слово AS не является обязательным. При создании псевдонима его можно опустить.

Вот как выглядят результаты, возвращаемые запросом:

С псевдонимами столбцов результаты стали намного более понятными.

А поскольку псевдонимы существуют лишь временно, они никак не влияют на имена используемых таблиц и столбцов.

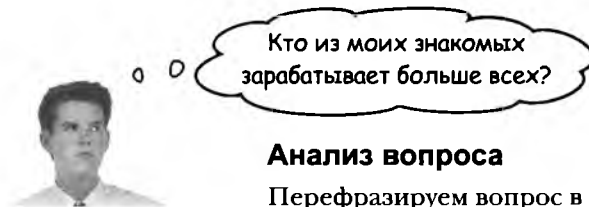
firstname	lastname	phone	jobtitle
Джо	Лонниган	(555) 555-3214	Повар
Венди	Хиллерман	(555) 555-8976	Официант
Шон	Миллер	(555) 555-4443	Веб-дизайнер
Джаред	Колуэй	(555) 555-5674	Веб-разработчик
Хуан	Гарза	(555) 555-0098	Веб-разработчик

Построение подзапроса

Самое сложное в подзапросах – не структура, а определение того, какую часть запроса следует преобразовать в подзапрос (и нужно ли это делать вообще).

Анализ запросов имеет много общего с решением простых арифметических задач. Вы находите в формулировке задачи то, что соответствует известным значениям (таблицы и имена столбцов), и разбиваете сложные утверждения на отдельные компоненты.

Давайте проанализируем вопрос, который нужно задать базе данных, и попробуем преобразовать его в код запроса.



Анализ вопроса

Перефразируем вопрос в контексте таблиц и столбцов базы данных.

«Кто» означает, что вас интересуют столбцы имени и фамилии из таблицы `my_contacts`. «Зарбатывает больше всех» означает, что вы хотите определить максимальное значение столбца `salary` из таблицы `job_current`.

Кто из моих знакомых зарабатывает больше всех?

← Столбцы «`first_name`» и «`last_name`» таблицы `my_contacts`.

← `MAX(salary)` из таблицы `job_current`.

Построение запроса, отвечающего на часть вопроса

Разделим вопрос на части и построим запрос, который возвращает ответ на одну из частей.

Похоже, хорошим кандидатом для первого запроса будет выборка `MAX(salary)`:

```
SELECT MAX(salary) FROM job_current;
```

←
Помните функцию `MAX`? Она возвращает наибольшее значение из столбца, указанного в скобках.

Продолжаем анализировать запрос.

С первой частью запроса тоже все ясно; из таблицы выбираются столбцы имени и фамилии:

```
SELECT mc.first_name, mc.last_name
FROM my_contacts AS mc;
```

Выборка имени и фамилии.

Остается понять, как связать эти два запроса.

Нам нужны не только имена людей из my_contacts, но и данные об их зарплате для сравнения с MAX (salary). Для выборки данных о зарплате каждого человека будет использоваться естественный внутренний запрос:

```
SELECT mc.first_name, mc.last_name, jc.salary
FROM my_contacts AS mc
NATURAL JOIN job_current AS jc;
```

Использование естественного соединения для выборки данных о зарплате каждого человека.

А теперь добавляем условие WHERE для связывания двух запросов.

Мы создаем один большой запрос на выборку, который отвечает на вопрос: «Кто зарабатывает больше всех?»

Та часть, которую мы только что написали, — извлекает сведения о зарплате каждого человека в таблице.

```
SELECT mc.first_name, mc.last_name, jc.salary
FROM my_contacts AS mc NATURAL JOIN job_current AS jc
WHERE jc.salary =
(SELECT MAX(jc.salary) FROM job_current jc);
```

Первая часть, которая стала подзапросом для определения максимального значения salary. Полученное значение используется для сравнения с внешней частью запроса для получения результата.

mc.first_name	mc.last_name	jc.salary
Майк	Скала	187000

Майк? Я так и думал — он никогда не проверяет чеки.





Похоже, то же самое можно было сделать и без подзапроса.

Верно, подзапрос — не единственное решение этой задачи.

К тому же результату можно было прийти с использованием естественного внутреннего соединения и команды `LIMIT`. Как это часто бывает в SQL, задачу можно решить несколькими разными способами.

МОЗГОВОЙ ШТУРМ

Напишите другой запрос, который покажет, кто из знакомых Грега больше всех зарабатывает.

Мне все равно, сколько решений имеет задача. Я хочу знать лучший способ. Или по крайней мере причины, по которым выбирается то или иное решение.



Логично.

Тогда почему бы вам не обратиться к интервью на с. 430?

Подзапрос как столбец SELECT

Подзапрос может использоваться в качестве одного из столбцов SELECT. Рассмотрим следующий пример.

```
SELECT mc.first_name, mc.last_name,
(SELECT state
FROM zip_code
WHERE mc.zip_code = zip_code) AS state
FROM my_contacts mc;
```

Определяем
псевдоним для
столбца «state».

Наш анализ этого запроса начинается с подзапроса. Подзапрос всего лишь устанавливает соответствие почтового индекса с обозначением штата по таблице zip_code.

В упрощенном виде запрос делает следующее:

Перебрать все записи в таблице my_contacts. Для каждой записи получить имя, фамилию и штат (для чего запрос получает почтовый индекс и сопоставляет его с обозначением штата по таблице zip_code).

Не забудьте, что подзапрос может вернуть только одно значение, поэтому при каждом его выполнении возвращается одна запись. Результат выглядит примерно так.

mc.first_name	mc.last_name	state
Джо	Лонниган	TX
Венди	Хиллерман	CA
Шон	Миллер	NY
Джаред	Колуэй	NJ
Хуан	Гарза	CA

Подзапрос, используемый в качестве выражения столбца в команде SELECT, может возвращать только одно значение из одного столбца.

Другой пример: подзапрос с естественным соединением

Друг Грег по имени Энди хвастается своим замечательным заработком. Он не сообщил подробностей, но Грег полагает, что вся необходимая информация хранится у него в таблице. Он быстро ищет ее, используя естественное соединение по адресу электронной почты Энди.

```
SELECT jc.salary  
FROM my_contacts mc NATURAL JOIN job_current jc  
WHERE email = 'andy@weatherorama.com';
```

← Запрос возвращает одно значение — зарплату Энди.

Этот запрос будет внутренним.

Грег замечает, что запрос возвращает всего один результат. Вместо того чтобы получить значение и подставить его в другой запрос, он решает преобразовать его в подзапрос.

Он пишет один запрос, который:

- получает зарплату Энди;
- сравнивает ее с другими зарплатами;
- возвращает имена и фамилии людей, которые зарабатывают больше Энди.

← С использованием оператора сравнения >.

← У которых значение «salary» больше, чем у Энди.

Запрос получается длинным, но он позволит мне сравнить значение, которого я пока не знаю, с другими значениями в базе данных.

Внешний запрос:

```
SELECT mc.first_name, mc.last_name, jc.salary  
FROM  
my_contacts AS mc NATURAL JOIN job_current AS jc  
WHERE  
jc.salary > (ЗДЕСЬ БУДЕТ ЗАПРОС С ЗАРПЛАТОЙ ЭНДИ)
```



Некоррелированный подзапрос

Итак, все компоненты собраны воедино; запрос готов. Сначала РСУБД однократно выполняет внутренний запрос, а затем использует полученное значение для вычисления результата внешнего запроса.

РСУБД начинает выполнение запроса с этой части.

Вывод данных только тех людей, у которых зарплата выше чем у Энди.

РСУБД обрабатывает эти два запроса по отдельности.

Небольшая часть результатов. Запрос не содержит ORDER BY, поэтому данные не упорядочены.

mc.first_name	mc.last_name	jc.salary
Гас	Логан	46500
Брюс	Хилл	78000
Тереза	Семел	48000
Рэнди	Райт	49000
Джулия	Мур	120000

```
SELECT mc.first_name, mc.last_name, jc.salary
FROM
my_contacts AS mc NATURAL JOIN job_current AS jc
WHERE
jc.salary > (SELECT jc.salary
FROM my_contacts mc NATURAL JOIN job_current jc
WHERE email = 'andy@weatherorama.com');
```

Выборка имени, фамилии и зарплат.

Подзапрос получает зарплату Энди, которая используется внешним запросом для сравнения.

Выполняется первым.

Автономный подзапрос, не содержащий ссылок на данные внешнего запроса, называется некоррелированным подзапросом.

Все подзапросы, которые встречались нам до настоящего момента, были **некоррелированными подзапросами**. Внутренний запрос обрабатывается первым, а полученный результат используется в условии WHERE внешнего запроса. **Но внутренний запрос никоим образом не зависит от данных внешнего запроса; его можно выполнить отдельно, как самостоятельный запрос.**



Внешний запрос обрабатывается вторым. Его результаты зависят от значения, возвращенного внутренним запросом.

Внутренний запрос не зависит от внешнего и выполняется первым.

(а если вы произнесете слова «некоррелированный подзапрос» в разговоре, это произведет большое впечатление на собеседников).



ОТКРОВЕННО ОБ SQL

Интервью недели:

Выбор оптимального построения запроса из нескольких вариантов

Head First SQL: Добро пожаловать, SQL. Спасибо, что не пожалели времени. Мы знаем, что дела у вас идут непросто.

SQL: Непросто? Теперь это так называется? Я бы сказал, что ситуация крайне тревожная, тяжелая и плохо поддающаяся объективному анализу при крайней запутанности.

Head First SQL: Да уж... Собственно, мы как раз об этом. Иногда приходится слышать жалобы на вашу чрезмерную гибкость. Когда мы задаем вам вопрос, вы даете слишком много вариантов ответа.

SQL: Действительно, я гибок. Один и тот же вопрос мне можно задать разными способами — и я дам одинаковые ответы.

Head First SQL: Кое-кто считает, что вам не хватает решительности.

SQL: Не стану оправдываться. Вы меня выставлете каким-то злодеем.

Head First SQL: Нет, мы знаем, что это не так — просто вы немного... неточны.

SQL: ХА! Я — неточен! С меня довольно. (*встает*)

Head First SQL: Нет, не уходите. Мы хотим всего лишь задать несколько вопросов. Иногда вы позволяете нам спрашивать одно и то же по-разному.

SQL: И что в этом плохого?

Head First SQL: Ничего, мы просто хотим понять, КАК ИМЕННО это нужно делать. Это существенно, если ответы все равно одинаковые?

SQL: Ну конечно, существенно. Иногда мне приходится очень долго думать, чтобы дать ответ.

Иногда БАХ — и ответ готов! Главное — правильно спросить.

Head First SQL: Значит, все дело во времени получения ответа? Нужно выбирать из этих соображений?

SQL: Ну конечно. Все дело в том, как спросить. Я просто пытаюсь ответить на ваши вопросы, когда они точно сформулированы.

Head First SQL: Скорость? В ней секрет?

SQL: Послушайте, я вам подскажу: базы данных РАСТУТ со временем. Ваши вопросы должны быть как можно проще, чтобы для выдачи ответа на заданный вопрос мне приходилось как можно меньше думать. Задавайте простые вопросы, и я быстро отвечу на них.

Head First SQL: Понятно. А как определить, какой вопрос простой, а какой — нет?

SQL: Например, перекрестные соединения обрабатываются очень долго. Коррелированные подзапросы тоже не отличаются быстротой...

Head First SQL: Еще что-нибудь?

SQL: Ну...

Head First SQL: Пожалуйста, продолжайте.

SQL: Экспериментируйте. Иногда лучше всего создать тестовые таблицы, опробовать разные запросы и сравнить время их обработки. Да, и соединения обрабатываются эффективнее подзапросов.

Head First SQL: Спасибо, SQL. Невероятно, такая тайна — и вдруг...

SQL: Ага. Спасибо, что не пожалели моего времени.

ПРАКТИКУМ ПОСТРОЕНИЯ ПОДЗАПРОСОВ

Прочитайте каждый из следующих сценариев. Напишите два запроса по приведенным инструкциям, а затем объедините их в запрос с подзапросом.

- 16 Грег хочет узнать среднюю зарплату веб-разработчиков в своей таблице `job_current`. Затем он хочет получить информацию о том, кто как зарабатывает относительно среднего уровня. Люди, зарабатывающие менее среднего, могут быть более заинтересованы в поиске новой работы.

Напишите запрос для определения средней зарплаты веб-разработчиков в таблице `job_current`.

- 17 Грег желает знать имя, фамилию и зарплату каждого веб-разработчика в таблице `job_current`.

Напишите запрос для получения имен, фамилий и зарплат всех веб-разработчиков в таблице `job_current`.

- 18 Используя среднюю зарплату (и немного математических вычислений) с подзапросом, Грег выводит информацию обо всех веб-разработчиках и о том, насколько больше (или меньше) среднего уровня они зарабатывают.

Объедините два запроса. Используйте подзапрос в списке столбцов `SELECT`.

ПРАКТИКУМ ПОСТРОЕНИЯ ПОДЗАПРОСОВ

Прочитайте каждый из следующих сценариев. Напишите два запроса по приведенным инструкциям, а затем объедините их в запрос с подзапросом.

- 1 Грег хочет узнать среднюю зарплату веб-разработчиков в своей таблице `job_current`. Затем он хочет получить информацию о том, кто как зарабатывает относительно среднего уровня. Люди, зарабатывающие менее среднего, могут быть более заинтересованы в поиске новой работы.

Напишите запрос для определения средней зарплаты веб-разработчиков в таблице `job_current`.

```
SELECT AVG(salary) FROM job_current WHERE title = 'Веб-разработчик';
```

↪ Среднее значение вычисляется функцией `AVG`.

- 2 Грег желает знать имя, фамилию и зарплату каждого веб-разработчика в таблице `job_current`.

Напишите запрос для получения имен, фамилий и зарплат всех веб-разработчиков в таблице `job_current`.

```
SELECT mc.first_name, mc.last_name, jc.salary
FROM my_contacts mc NATURAL JOIN job_current jc
WHERE jc.title = 'Веб-разработчик';
```

- 3 Используя среднюю зарплату (и немного математических вычислений) с подзапросом, Грег выводит информацию обо всех веб-разработчиках и о том, насколько больше (или меньше) среднего уровня они зарабатывают.

Объедините два запроса. Используйте подзапрос в списке столбцов `SELECT`.

```
SELECT mc.first_name, mc.last_name, jc.salary,
       jc.salary — (SELECT AVG(salary) FROM job_current WHERE title = 'Веб-разработчик')
FROM my_contacts mc NATURAL JOIN job_current jc
WHERE jc.title = 'Веб-разработчик';
```

↪ Наш подзапрос.

Некоррелированный подзапрос с несколькими значениями: IN, NOT IN

Вернемся к первому запросу, который был опробован Грегом на с. 17. Этот запрос позволил ему найти людей, должности которых *совпадают* с одной из предложенных вакансий. Запрос берет полный набор значений title, возвращаемых командой SELECT в подзапросе, и проверяет по этому набору каждую запись таблицы job_current для поиска возможных совпадений.

```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title
FROM job_current AS jc NATURAL JOIN my_contacts AS mc
WHERE jc.title IN (SELECT title FROM job_listings);
```

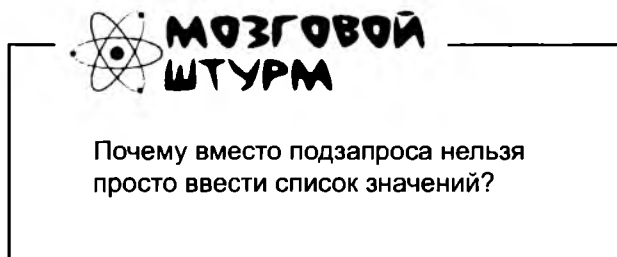
Значение jc.title каждой записи проверяется по всему набору, возвращенному подзапросом.

Ключевые слова **NOT IN** позволят Грегу отобразить должности, *отсутствующие* среди его вакансий. Эта конструкция берет полный набор значений title, возвращенных командой SELECT в подзапросе, и проверяет по нему каждую запись таблицы job_current. Запрос возвращает все значения, *отсутствующие* в наборе. Теперь Грег может направить усилия на поиск вакансий для этих должностей.

```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title
FROM job_current jc NATURAL JOIN my_contacts mc
WHERE jc.title NOT IN (SELECT title FROM job_listings);
```

NOT IN возвращает должности, отсутствующие среди предлагаемых вакансий.

Такие запросы называются **некоррелированными подзапросами**; ключевые слова IN или NOT IN проверяют соответственно присутствие или отсутствие результатов внешнего запроса среди результатов подзапроса.



Некоррелированный подзапрос при помощи IN или NOT IN проверяет факт наличия (или отсутствия) значений, возвращенных подзапросом, в наборе.



Упражнение

Напишите запросы для получения ответов на следующие вопросы (используйте соединения и некоррелированные подзапросы там, где это уместно). Используйте схему базы данных gregs_list на следующей странице.

В некоторых запросах должны использоваться агрегатные функции, которые рассматривались нами в задаче о продаже печенья.

Выведите все должности с зарплатой, равной наибольшей зарплате из таблицы job_listings.

→ Ответ на с. 436.

Выведите имена и фамилии людей с зарплатой выше средней.

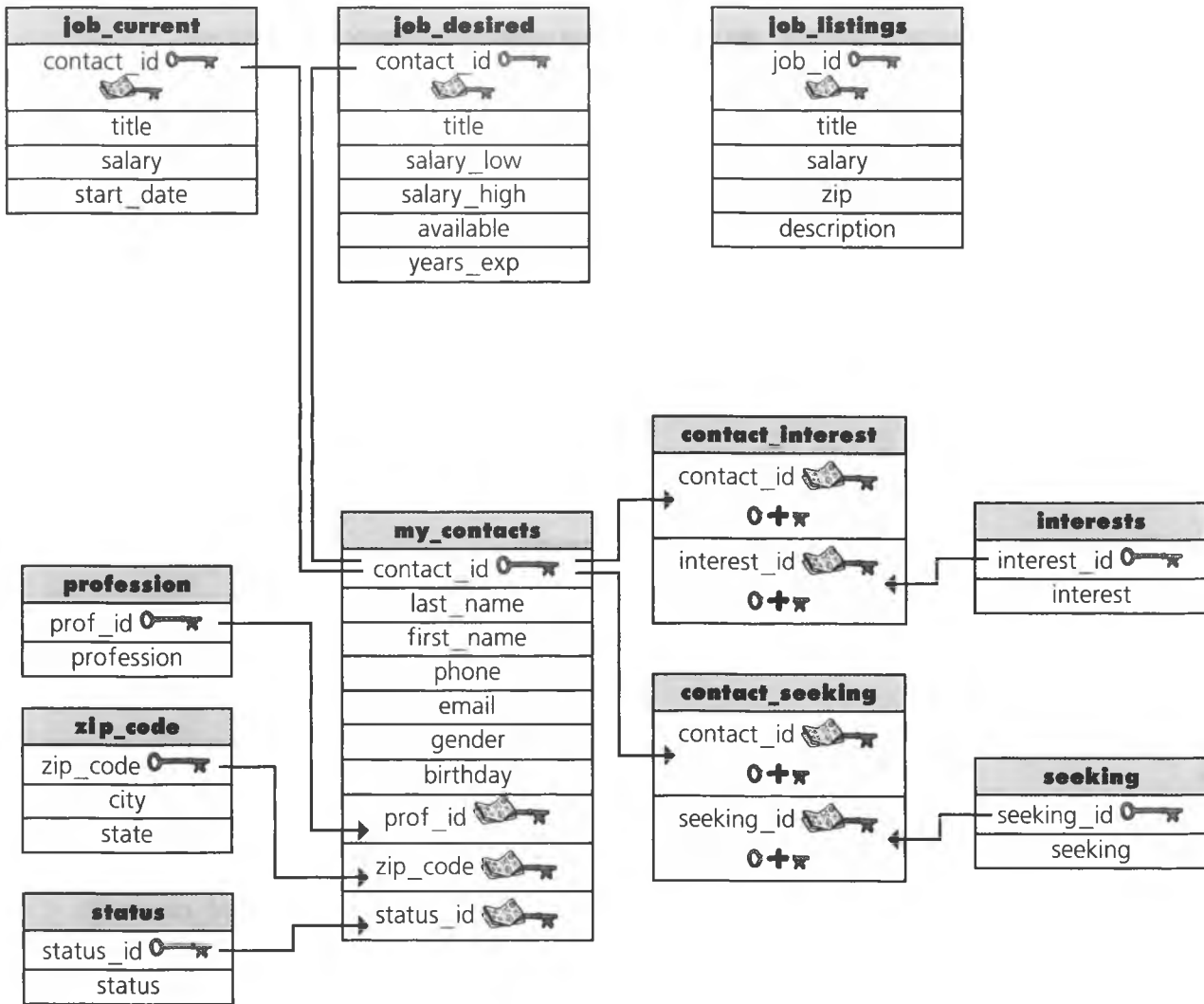
→ Ответ на с. 436.

Найдите всех веб-дизайнеров, у которых почтовый индекс (zip_code) совпадает с почтовым индексом какой-либо вакансии веб-дизайнера из таблицы job_listings.

→ Ответ на с. 437.

Выведите список всех людей, у которых почтовый индекс (zip_code) совпадает с почтовым индексом человека, получающего максимальную зарплату.

→ Ответ на с. 437.





Упражнение Решение

Напишите запросы для получения ответов на следующие вопросы (используйте соединения и некоррелированные подзапросы там, где это уместно). Используйте схему базы данных `gregs_list`.

Выведите все должности с зарплатой, равной наибольшей зарплате из таблицы `job_listings`.

Результаты внешнего запроса сравниваются со значением `MAX(salary)`.

```
SELECT title FROM job_listings
WHERE salary = (SELECT MAX(salary)
FROM job_listings);
```

Подзапрос возвращает одно значение.

`MAX` возвращает наибольшую зарплату в таблице.

Выведите имена и фамилии людей с зарплатой выше средней.

Внешний запрос получает результат подзапроса и возвращает записи, у которых «salary» больше вычисленного среднего значения.

```
SELECT mc.first_name, mc.last_name
FROM my_contacts mc
NATURAL JOIN job_current jc
WHERE jc.salary > (SELECT AVG(salary) FROM job_current);
```

Естественное соединение возвращает информацию о людях, у которых зарплата превышает значение, возвращаемое внутренним запросом.

Подзапрос возвращает среднюю зарплату.

Найдите всех веб-дизайнеров, у которых почтовый индекс (zip_code) совпадает с почтовым индексом какой-либо вакансии веб-дизайнера из таблицы job_listings.

Для получения полезной информации о найденных людях (имя, фамилия, телефон) необходимо воспользоваться естественным соединением.

```
SELECT mc.first_name, mc.last_name, mc.phone FROM my_contacts mc
NATURAL JOIN job_current jc WHERE jc.title = 'web designer' AND mc.zip_code
IN (SELECT zip FROM job_listings WHERE title = 'web designer');
```

Так как подзапрос может вернуть более одного индекса, мы рассматриваем результаты как набор, для проверки принадлежности к которому используется ключевое слово «IN».

Внутренний запрос возвращает все почтовые индексы вакансий веб-дизайнеров.

Выведите список всех людей, у которых почтовый индекс (zip_code) совпадает с почтовым индексом человека, получающего максимальную зарплату.

Вопрос с подвохом — в таблице может быть сразу несколько человек с максимальной зарплатой. Это означает, что в запросе придется использовать IN и нам понадобятся два подзапроса.

Внешний запрос получает почтовые индексы и ищет совпадения в таблице my_contacts. Так как средний подзапрос может вернуть более одного почтового индекса, мы используем IN.

```
SELECT last_name, first_name FROM my_contacts
WHERE zip_code IN (SELECT mc.zip_code FROM my_contacts mc
NATURAL JOIN job_current jc
WHERE jc.salary = (SELECT MAX(salary) FROM job_current));
```

Подзапрос с наибольшим уровнем вложенности получает максимальную зарплату из таблицы job_current. Так как результат представляет собой скалярное значение, мы можем использовать =.

Средний подзапрос находит почтовые индексы людей с максимальной зарплатой.

Коррелированные подзапросы



Если некоррелированные подзапросы существуют сами по себе, могут поспорить, что коррелированные подзапросы каким-то образом зависят от внешнего запроса.

Правильно. В модели с некоррелированным подзапросом внутренний запрос (то есть подзапрос) интерпретируется РСУБД после внешнего запроса.

Таким образом, в модели с коррелированным подзапросом выполнение внутреннего запроса возможно только при условии выполнения внешнего.

Следующий запрос подсчитывает количество увлечений в таблице `interest` для каждого человека в `my_contacts`, а затем возвращает имена и фамилии людей с тремя увлечениями.

```
SELECT mc.first_name, mc.last_name
```

```
FROM my_contacts AS mc
```

```
WHERE
```

```
3 = (
```

```
SELECT COUNT(*) FROM contact_interest
```

```
WHERE contact_id = mc.contact_id
```

```
);
```

В подзапросе используется псевдоним `mc`.

Псевдоним для `my_contacts` создается во внешнем запросе.

Чтобы значение `mc.contact_id` могло использоваться во внутреннем запросе, сначала должен быть выполнен внешний запрос.

Подзапрос зависит от внешнего запроса. Обработка внутреннего запроса станет возможной только после того, как внешний запрос предоставит значение `contact_id`.

В подзапросе используется псевдоним таблицы `my_contacts` – `mc`, который был создан во внешнем запросе.

Коррелированный подзапрос с NOT EXISTS

Очень распространенный сценарий использования коррелированного подзапроса — поиск во внешнем запросе всех записей, у которых нет соответствующих записей в связанной таблице.

Допустим, Грег хочет расширить круг клиентов своей службы поиска работы. Для этого он собирается разослать сообщения всем людям из `my_contacts`, данные которых еще не содержатся в таблице `job_current`. Для поиска записей он использует условие `NOT EXISTS`.

```
SELECT mc.first_name firstname, mc.last_name lastname, mc.email email
FROM my_contacts mc
WHERE NOT EXISTS
  (SELECT * FROM job_current jc
   WHERE mc.contact_id = jc.contact_id );
```

Запрос с NOT EXISTS находит имена, фамилии и адреса электронной почты тех людей из таблицы my_contacts, которые в настоящее время не представлены в таблице job_current.

* КТО И ЧТО ДЕЛАЕТ? *

Соедините каждую часть приведенного выше запроса с описанием того, что она делает.

<code>mc.first_name firstname</code>	Назначает псевдоним для поля <code>mc.last_name</code>
<code>WHERE NOT EXISTS</code>	Если два значения <code>contact_id</code> совпадают, условие выполнено
<code>WHERE mc.contact_id = jc.contact_id</code>	Назначает полю псевдоним <code>firstname</code>
<code>FROM my_contacts mc</code>	Выбирает все поля таблицы с псевдонимом <code>jc</code>
<code>mc.last_name lastname</code>	Назначает полю псевдоним <code>email</code>
<code>SELECT * FROM job_current jc</code>	Истинно, если записи не найдены
<code>mc.email email</code>	Назначает псевдоним для таблицы <code>my_contacts</code>

EXISTS и NOT EXISTS

По аналогии с IN и NOT IN, в подзапросах также можно использовать ключевые слова **EXISTS** и **NOT EXISTS**. Приведенный ниже подзапрос возвращает данные из `my_contacts`, у которых значение `contact_id` по крайней мере один раз встречается в таблице `contact_interest`.

```
SELECT mc.first_name firstname, mc.last_name lastname, mc.email email
FROM my_contacts mc
WHERE EXISTS (SELECT * FROM contact_interest ci WHERE mc.contact_id = ci.contact_id );
```

Запрос с EXISTS находит имена, фамилии и адреса электронной почты людей из таблицы `my_contacts`, у которых значение `contact_id` по крайней мере один раз встречается в таблице `contact_interest`.

* КТО И ЧТО ДЕЛАЕТ? *

Соедините каждую часть приведенного выше запроса с описанием того, что она делает.

<code>mc.first_name firstname</code>	Назначает псевдоним для поля <code>mc.last_name</code>
<code>WHERE NOT EXISTS</code>	Если два значения <code>contact_id</code> совпадают, условие выполнено
<code>WHERE mc.contact_id = jc.contact_id</code>	Назначает полю псевдоним <code>firstname</code>
<code>FROM my_contacts mc</code>	Выбирает все поля таблицы с псевдонимом <code>jc</code>
<code>mc.last_name lastname</code>	Назначает полю псевдоним <code>email</code>
<code>SELECT * FROM job_current jc</code>	Истинно, если записи не найдены
<code>mc.email email</code>	Назначает псевдоним для таблицы <code>my_contacts</code>

Возьми в руку карандаш



Напишите запрос для получения адресов электронной почты людей, которые имеют не менее одного увлечения, но при этом отсутствуют в таблице `job_current`.

→ Ответ на с. 444.

Служба поиска работы Грега принимает заказы

Грег вполне освоился с выборкой данных с использованием подзапросов. Он даже научился пользоваться ими в командах INSERT, UPDATE и DELETE.

Он снял небольшой офис и собирается провести вечеринку, чтобы отпраздновать начало нового дела.



Интересно, удастся ли мне найти своего первого работника в таблице job_desired...

Часто задаваемые вопросы

В: Итак, подзапрос можно вложить в другой подзапрос?

О: Безусловно. Количество уровней вложения подзапросов ограничено, но в большинстве РСУБД оно значительно превышает практический «потолок».

В: Как лучше всего строить подзапрос внутри подзапроса?

О: Попробуйте написать маленькие запросы для различных частей вопроса. Присмотритесь к ним и попробуйте скомбинировать. Если вы пытаетесь найти людей с такой же зарплатой, как у самого высокооплачиваемого веб-дизайнера, разбиение запроса может выглядеть так:

Найти самого высокооплачиваемого веб-дизайнера
Найти людей, зарабатывающих x

после чего подставить первый ответ на место x.

В: Подзапросы мне не нравятся, могу ли я использовать вместо них соединения?

О: В большинстве случаев — да, можете, но сначала необходимо еще кое-что узнать о соединениях...

По дороге на Вечеринку

Грег обнаруживает в газете статью с сенсационным заголовком.

THE WEEKLY INQUERUER



ШОКИРУЮЩАЯ ПРАВДА

О ПОДЗАПРОСАХ!

СКРЫТЫЕ СОЕДИНЕНИЯ

Соседи утверждают, что подзапросы — «не что иное», как обычные соединения, и «...люди должны узнать правду».

Трой Армстронг
Репортер INQUERUER

Дейтавилль — Источники Inqueruer подтвердили то, что что в течение многих лет считалось обычными слухами. Соединения и подзапросы могут использоваться для реализации абсолютно одинаковых запросов. К смущению местных жителей, все, что можно сделать с помощью подзапроса, также можно сделать с помощью некоторого типа соединения.

«Это ужасно, — рыдает учительница Хейди Мусгроув, — Как я скажу детям, что после всех трудов по изучению подзапросов, после всего потраченного на них времени они могли просто использовать соединения? Это катастрофа!»

Тема будет продолжена в следующей главе, где внешние запросы станут предметом пристального внимания общест-венности.



Местная жительница Хейлли Мусгроув узнает шокирующую правду о подзапросах.

**ВЫХОДИТ, МЫ ТОЛЬКО ЗРЯ ПОТРАТИЛИ ВРЕМЯ? И ПОДЗАПРОСЫ
НИЧЕМ НЕ ОТЛИЧАЮТСЯ ОТ СОЕДИНЕНИЙ?
ОТВЕТ НА ЭТОТ ВОПРОС ВЫ УЗНАЕТЕ В СЛЕДУЮЩЕЙ ГЛАВЕ.**



Новые инструменты

В главе 9 вы овладели искусством построения подзапросов. Вспомните то, что вы узнали в ней. Полный список инструментов приведен в приложении III.

Некоррелированный подзапрос

Подзапрос, который существует сам по себе и не содержит ссылок на данные внешнего запроса.

Коррелированный подзапрос

Подзапрос, который зависит от значений, возвращаемых внешним запросом.

Внешний запрос

Запрос, содержащий внутренний запрос (подзапрос).

Внутренний запрос

Запрос, находящийся внутри другого запроса. Также может называться подзапросом.

Подзапрос

Запрос, вложенный в другой запрос. Также может называться «внутренним запросом».

Возьми в руку карандаш



Решение
Со с. 441.

Напишите запрос для получения адресов электронной почты людей, которые имеют не менее одного увлечения, но при этом отсутствуют в таблице `job_current`.

```

SELECT mc.email FROM my_contacts mc WHERE
EXISTS
(SELECT * FROM contact_interest ci WHERE mc.contact_ID = ci.contact_ID)
AND ← Как обычно при реализации двух условий, которые
NOT EXISTS      должны быть истинны одновременно,
                в условии WHERE используется связка AND.
(SELECT * FROM job_current jc
WHERE mc.contact_id = jc.contact_id );

```

10 Внешние соединения, самосоединения и союзы

* Новые приемы *



Пока вы знаете только половину того, что необходимо знать о соединениях. Вы видели перекрестные соединения, которые возвращают все возможные комбинации записей, и внутренние соединения, которые возвращают записи обеих таблиц при наличии совпадения. Однако существуют еще и **внешние соединения**, которые возвращают записи, *не имеющие совпадений в другой таблице*, **самосоединения**, которые (как ни странно) *соединяют таблицу саму с собой*, и **союзы**, которые *объединяют результаты запросов*. Освоив эти приемы, вы сможете получить все данные точно в том виде, в котором они вам нужны. (И узнаете правду о подзапросах, как мы и обещали!)

Уничтожение старых данных



Надо бы почистить таблицу professions. Там хранятся некоторые значения, которые я больше не использую. Как легко найти профессии, не связанные ни с одной записью в таблице my_contacts? Внутреннее соединение для этого не подойдет.

Для получения этой информации можно воспользоваться внешним соединением.

Давайте сначала посмотрим, как работает внешнее соединение, а потом мы выясним, как же найти неиспользуемые профессии.

Внешние соединения возвращают все записи одной из таблиц вместе со всеми соответствующими данными из другой таблицы.

При внутреннем соединении *сравниваются записи двух таблиц*, причем порядок следования этих двух таблиц неважен.

Давайте в общих чертах посмотрим, что делает эквивалентное соединение. Мы получаем столбцы, связанные совпадением значений toy_id в обеих таблицах:

```
SELECT g.girl, t.toy
FROM girls g
INNER JOIN toys t
ON g.toy_id = t.toy_id;
```



Левое, правое...

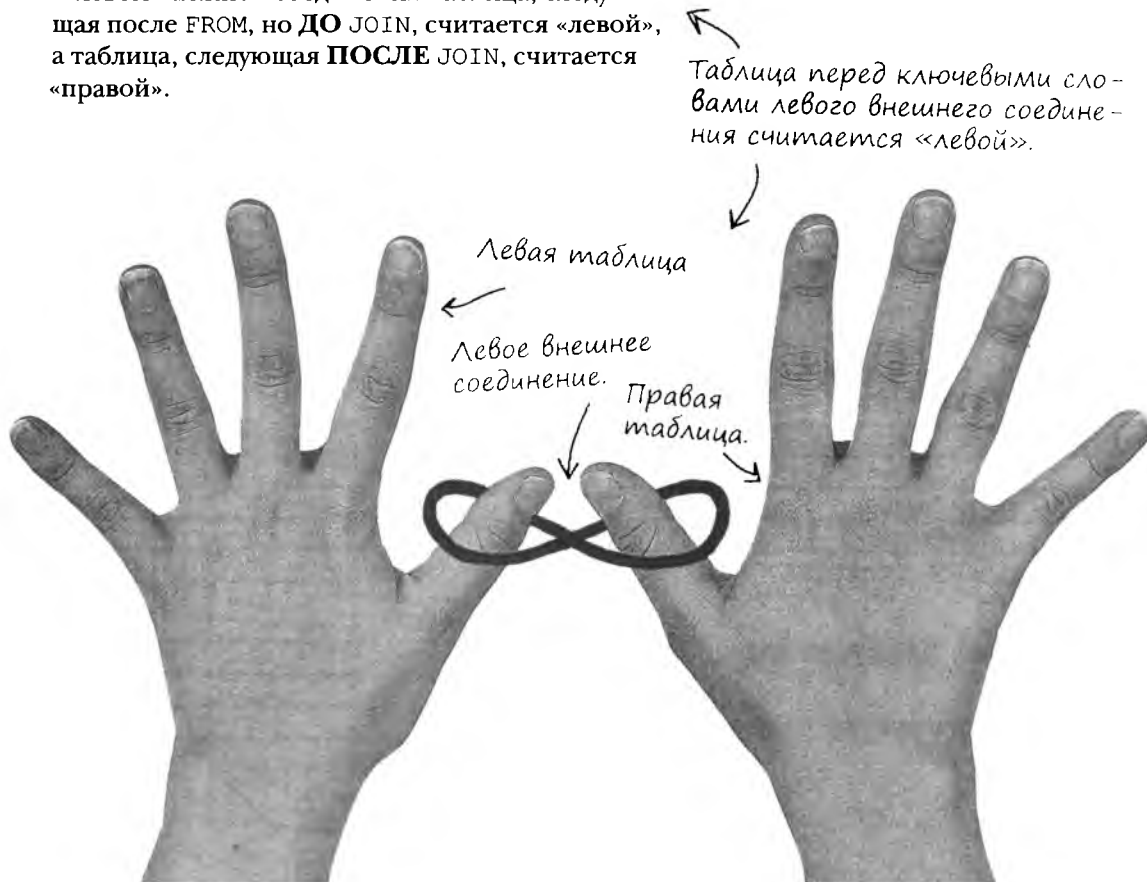
С другой стороны, внешние соединения в большей степени зависят от *отношений между двумя таблицами*, чем все рассмотренные ранее типы соединений.

Левое внешнее соединение (**LEFT OUTER JOIN**) перебирает *все записи левой таблицы* и ищет для каждой **соответствие среди записей правой таблицы**. В частности это удобно, когда между левой и правой таблицей существует связь типа «один-ко-многим».

Чтобы понять логику внешнего соединения, необходимо понять, какая таблица находится «слева», а какая — «справа».

В левом внешнем соединении таблица, следующая после FROM, но **ДО JOIN**, считается «левой», а таблица, следующая **ПОСЛЕ JOIN**, считается «правой».

В левом внешнем соединении для КАЖДОЙ ЗАПИСИ ЛЕВОЙ таблицы ищется соответствие среди записей правой таблицы.



Пример левого внешнего соединения

При помощи левого внешнего соединения мы можем узнать, какая игрушка принадлежит той или иной девочке.

Ниже приведен синтаксис левого внешнего соединения на примере уже использовавшихся таблиц. Таблица `girls` указана первой после `FROM`, поэтому она считается левой таблицей; далее следуют ключевые слова `LEFT OUTER JOIN`; и наконец, таблица `toys` считается правой таблицей.

Итак, левое внешнее соединение перебирает все записи левой таблицы (`girls`) и ищет для каждой соответствие среди записей правой таблицы (`toys`).

```
SELECT g.girl, t.toy
```

```
FROM girls g
```

```
LEFT OUTER JOIN toys t
```

```
ON g.toy_id = t.toy_id;
```

Таблица `girls` предшествует `LEFT OUTER JOIN`, поэтому она является левой...

...а таблица `toys`, следующая за `LEFT OUTER JOIN`, является правой...

Таблица `girls` предшествует `LEFT OUTER JOIN`, поэтому она является левой...

...а таблица `toys`, следующая за `LEFT OUTER JOIN`, является правой.

girls

girl_id	girl	toy_id
1	Джейн	3
2	Салли	4
3	Синди	1

toys

toy_id	toy
1	обруч
2	самолет
3	солдатики
4	губная гармошка
5	бейсбольные карточки
6	кубики
7	волшебный экран
8	пружина

Результат левого внешнего соединения

Результат тот же, что и при внутреннем соединении.

Результат

girl	toy
Синди	обруч
Джейн	солдатики
Салли	губная гармошка



o o

И все? Спрашивается, чего мы добились? Выходит, внешнее соединение ничем не отличается от внутреннего.

Отличается: внешнее соединение возвращает запись независимо от того, есть у нее совпадение в другой таблице или нет.

Отсутствие совпадений обозначается значением NULL. В нашем примере с девочками и игрушками NULL в результатах означает, что данная игрушка не принадлежит никому из девочек. Очень ценная информация!

Значение NULL в результатах левого внешнего соединения означает, что правая таблица не содержит значений, соответствующих левой таблице.

Возьми в руку карандаш



Напишите, как, по вашему мнению, будет выглядеть таблица результатов этого запроса.

```
SELECT g.girl, t.toy
FROM toys t
LEFT OUTER JOIN girls g
ON g.toy_id = t.toy_id;
```

(Подсказка. Таблица результатов будет содержать 8 записей.)



Возьми в руку карандаш

Решение

В этом запросе таблицы поменялись местами. Напишите, как, по вашему мнению, будет выглядеть таблица результатов.

```
SELECT g.girl, t.toy
FROM toys t ← Левая таблица
LEFT OUTER JOIN girls g ← Правая таблица
ON g.toy_id = t.toy_id;
```

На этот раз каждая запись в таблице toys (левая таблица) проверяется по таблице girls (правая таблица).

Левая таблица
toys

toy_id	toy
1	обруч
2	самолет
3	солдатики
4	губная гармошка
5	бейсбольные карточки
6	кубики
7	волшебный экран
8	пружина

Правая таблица
girls

girl_id	girl	toy_id
1	Джейн	3
2	Салли	4
3	Синди	1

С изменением порядка таблиц изменяется и результат.

Если совпадение будет найдено, результат включается в таблицу. Если совпадение отсутствует, запись в таблице все равно создается, но отсутствующее значение заменяется NULL.

girl	toy
Синди	обруч
NULL	самолет
Джейн	солдатики
Салли	губная гармошка
NULL	бейсбольные карточки
NULL	кубики
NULL	волшебный экран
NULL	пружина

Столбцы в таблице результатов показаны в порядке их ВЫБОРКИ. Этот порядок не имеет ничего общего с ЛЕВОЙ и ПРАВОЙ таблицей соединения.



Упражнение

Ниже приведены два результата. Напишите для каждого из них левый внешний запрос, который мог бы привести к его созданию, а также таблицы `girls` и `toys` с данными, соответствующими результатам.

Запрос

Результат левого внешнего соединения

girl	toy
Джен	водяной пистолет
Клео	дудка
Мэнди	NULL

Левая таблица

Это мы сделали за вас.
↓
girls

<i>girl_id</i>	<i>girl</i>	<i>toy_id</i>
1	Джен	1
2	Клео	2
3	Мэнди	3

Правая таблица

Запрос

Результат левого внешнего соединения ↓

Эта задача посложнее.

girl	toy
Jen	водяной пистолет
Cleo	водяной пистолет
NULL	дудка
Sally	пружина
Martha	пружина

Левая таблица

Правая таблица



упражнение

Ниже приведены два результата. Напишите для каждого из них левый внешний запрос, который мог бы привести к его созданию, а также таблицы girls и toys с данными, соответствующими результатам.

Решение

Запрос

```
SELECT g.girl, t.toy
FROM girls g
LEFT OUTER JOIN toys t
ON g.toy_id = t.toy_id;
```

Левая таблица

girls		
girl_id	girl	toy_id
1	Джен	1
2	Клео	2
3	Мэнди	3

Здесь может быть любое значение toy_id, не существующее в таблице toys, потому что в столбце toy результата хранится NULL.

Запрос

```
SELECT g.girl, t.toy
FROM toys t
LEFT OUTER JOIN girls g
ON g.toy_id = t.toy_id;
```

Левая таблица

toys	
toy_id	toy
1	водяной пистолет
2	дудка
3	пружина

Результат левого внешнего соединения

girl	toy
Джен	водяной пистолет
Клео	дудка
Мэнди	NULL

Игрушки, присутствующие в результатах.

Правая таблица

toys	
toy_id	toy
1	водяной пистолет
2	дудка

Повторяющиеся значения указывают на то, что одна игрушка есть у нескольких девочек.

Результат левого внешнего соединения

girl	toy
Jen	водяной пистолет
Cleo	водяной пистолет
NULL	дудка
Sally	пружина
Martha	пружина

NULL означает, что ни у одной из девочек нет дудки.

Правая таблица

girls		
girl_id	girl	toy_id
1	Джен	1
2	Клео	1
3	Салли	3
4	Марта	3

Внешние соединения и множественные совпадения

Как видно из этого примера, запись в результирующем наборе создается даже при отсутствии совпадений в другой таблице, а при множественных совпадениях будет создано несколько записей. Вот что происходит при левом внешнем соединении:

```
SELECT g.girl, t.toy
FROM toys t
LEFT OUTER JOIN girls g
ON g.toy_id = t.toy_id;
```

toys		girls		
toy_id	toy	girl_id	girl	toy_id
1	водяной пистолет	1	Джен	1
2	дудка	2	Клео	1
3	пружина	3	Салли	3
		4	Марта	3

Запись с водяным пистолетом (toys) сравнивается с записью Джен (girls): toys.toy_id = 1, girls.toy_id = 1

Есть совпадение.

Запись с водяным пистолетом (toys) сравнивается с записью Клео (girls): toys.toy_id = 1, girls.toy_id = 1

Есть совпадение.

Запись с водяным пистолетом (toys) сравнивается с записью Салли (girls): toys.toy_id = 1, girls.toy_id = 3

Нет совпадения.

Запись с водяным пистолетом (toys) сравнивается с записью Марты (girls): toys.toy_id = 1, girls.toy_id = 3

Нет совпадения.

Запись с дудкой (toys) сравнивается с записью Джен (girls): toys.toy_id = 2, girls.toy_id = 1

Нет совпадения.

Запись с дудкой (toys) сравнивается с записью Клео (girls): toys.toy_id = 2, girls.toy_id = 1

Нет совпадения.

Запись с дудкой (toys) сравнивается с записью Салли (girls): toys.toy_id = 2, girls.toy_id = 3

Нет совпадения.

Запись с дудкой (toys) сравнивается с записью Марты (girls): toys.toy_id = 2, girls.toy_id = 3

Нет совпадения.

Конец таблицы, создается запись с NULL.

Запись с пружиной (toys) сравнивается с записью Джен (girls): toys.toy_id = 3, girls.toy_id = 1

Нет совпадения.

Запись с пружиной (toys) сравнивается с записью Клео (girls): toys.toy_id = 3, girls.toy_id = 1

Нет совпадения.

Запись с пружиной (toys) сравнивается с записью Салли (girls): toys.toy_id = 3, girls.toy_id = 3

Есть совпадение.

Запись с пружиной (toys) сравнивается с записью Марты (girls): toys.toy_id = 3, girls.toy_id = 3

Есть совпадение.

girl	toy
Джен	водяной пистолет
Клео	водяной пистолет
NULL	дудка
Салли	пружина
Марта	пружина

Правое внешнее соединение

Правое внешнее соединение почти полностью аналогично левому внешнему соединению, кроме того, что оно сравнивает правую таблицу с левой. Следующие два запроса возвращают абсолютно одинаковые результаты.

Правое внешнее соединение ищет в левой таблице соответствия для правой таблицы.

```
SELECT g.girl, t.toy
FROM toys t ← Правая таблица.
RIGHT OUTER JOIN girls g ← Левая таблица.
ON g.toy_id = t.toy_id;
```

```
SELECT g.girl, t.toy
FROM girls g ← Левая таблица.
LEFT OUTER JOIN toys t ← Правая таблица.
ON g.toy_id = t.toy_id;
```

Этот запрос уже приводился на с. 448.

Левая таблица
(в обоих запросах).

В обоих запросах
таблица girls явля-
ется левой таблицей.

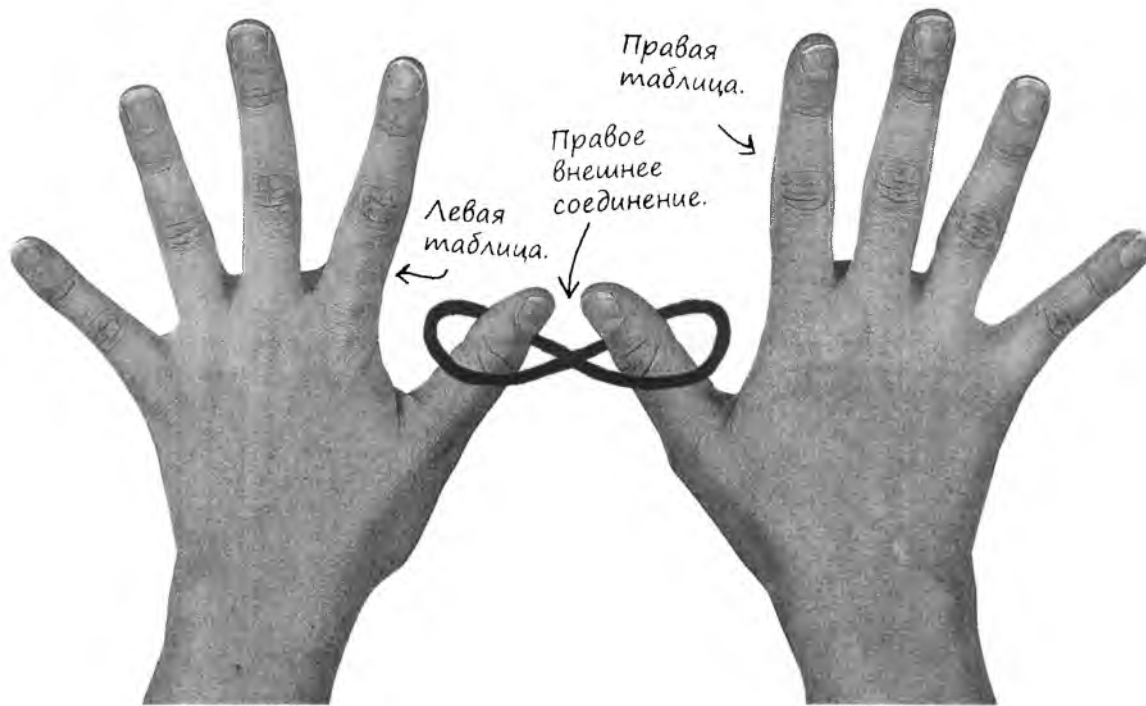
Правая таблица
(в обоих запросах).

girl_id	girl	toy_id
1	Джейн	3
2	Салли	4
3	Синди	1

toy_id	toy
1	обруч
2	самолет
3	солдатики
4	губная гармошка
5	бейсбольные карточки
6	кубики
7	волшебный экран
8	пружина

Наши
результаты.

girl	toy
Синди	обруч
Джейн	солдатики
Салли	губная гармошка



часто
Задаваемые
Вопросы

В: Есть ли причины для использования левого внешнего соединения вместо правого?

О: Заменить ключевое слово LEFT ключевым словом RIGHT проще, чем изменять порядок таблиц в запросе. Достаточно изменить всего одно слово — вам не придется переставлять имена таблиц и их псевдонимы.

С другой стороны, на практике обычно бывает проще всегда придерживаться одного типа (скажем, левых внешних соединений) и менять местами таблицы. В этом случае код получается более понятным.

В: Если существуют ЛЕВОЕ и ПРАВОЕ внешние соединения, то существует ли соединение, возвращающее результаты левого и правого соединений?

О: В некоторых (но не во всех) РСУБД существуют такие соединения, называемые ПОЛНЫМИ внешними соединениями. Они не поддерживаются в MySQL, SQL Server и Access.



Нельзя ли воспользоваться внешним соединением для того, чтобы связать таблицу с ней самой? Иногда это может быть полезно.

Одна и та же таблица может использоваться в качестве левой и правой таблицы соединения.

И хотя сама идея такого соединения на первый взгляд может показаться странной, она может оказаться полезной. Рассмотрим пример ситуации, в которой может пригодиться внешнее соединение таблицы с ней самой.

Но сначала необходимо разобраться с одной проблемой, которая возникла в Дейтавиле...

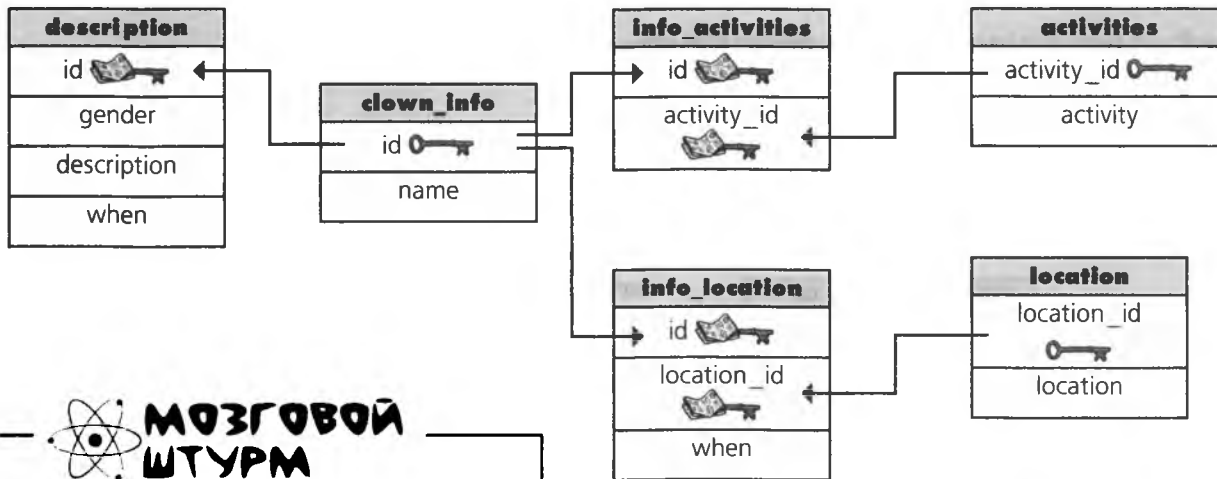
Пока мы занимались внешними соединениями...

В Дейтавиле клоуны организовали профсоюз, и у них появились свои начальники и подчиненные. Ситуация принимает угрожающий оборот, и мы должны следить за тем, кто является начальником и кому из начальников подчиняется тот или иной клоун.

Перед вами пример новой иерархии клоунов. У каждого клоуна имеется один начальник (кроме самого главного клоуна, Мистера Снифлза).



Перед вами текущая схема базы данных. Давайте подумаем, как лучше внедрить в нее новую информацию.



МОЗГОВОЙ ШТУРМ

Как изменить структуру схемы, чтобы сохранить в базе данных информацию об иерархии клоунов?

Вам смешно, надеюсь? Ведь я же клоун... Смешно, спрашиваю?

Снифлз, начальник Кларабелл и Снаглз.



Создание новой таблицы

Мы можем создать таблицу с перечислением всех клоунов и идентификаторов их начальников. Вот как выглядит иерархия с идентификаторами.



В новой таблице для каждого клоуна указан идентификатор его начальника из таблицы `clown_info`.

clown_boss

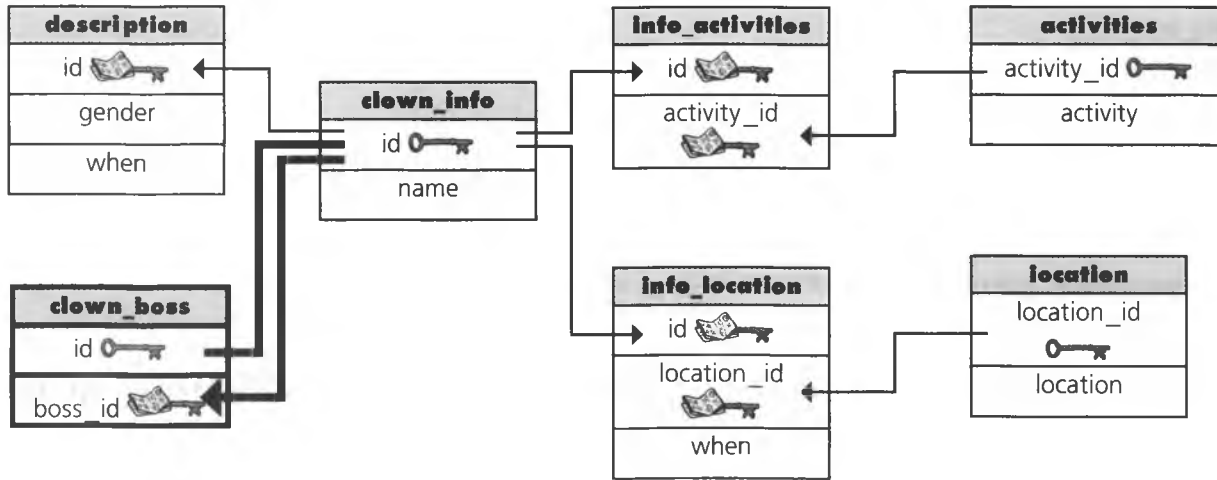
id	boss_id
1	3
2	5
3	10
4	3
5	10
6	3
7	3
8	5
9	5
10	10

Между таблицей `clown_boss` и таблицей `clown_info` существует связь типа «один-к-одному».

У Мистера Снифлза нет начальника, но идентификатор ему нужен. В столбце «`boss_id`» будет указан его собственный идентификатор, чтобы избежать значения `NULL` в этом столбце.

Место новой таблицы в схеме

Посмотрите на текущую схему базы данных. Как лучше встроить в нее новую таблицу?



Ситуация немного странная. В схеме возникает связь типа «один-к-одному» с `id` (первичный ключ) и связь типа «один-ко-многим» с `boss_id`. И первичный ключ, и внешний ключ *находятся в таблице `clown_info`*.

Похоже, мы можем воспользоваться таблицей со связью «один-к-одному». Но раз таблица не содержит закрытой информации, нельзя ли встроить эти данные в основную таблицу?



Можно ли хранить информацию об иерархии клоунов без создания отдельной таблицы?

Рефлексивный Внешний ключ

В таблицу `clown_info` следует добавить новый столбец с информацией о том, кто является начальником того или иного клоуна. В новом столбце будет храниться идентификатор начальника. Мы назовем его `boss_id`, как в таблице `clown_boss`.

В таблице `clown_boss` столбец `boss_id` был внешним ключом. При добавлении в `clown_info` этот столбец все равно остается внешним ключом, хотя и находится в другой таблице. Такие внешние ключи, ссылающиеся на другое поле той же таблицы, называются **рефлексивными**.

Мы считаем, что Мистер Снифлз является своим собственным начальником, поэтому у него значение `boss_id` совпадает с `id`.

Рефлексивным внешним ключом называется *первичный ключ* таблицы, *используемый в той же таблице для другой цели*.

РЕФЛЕКСИВНЫЙ внешний ключ — **первичный** ключ таблицы, **используемый в той же таблице для других целей**.

Новый столбец «`boss_id`», который мы просто добавили в таблицу `clown_info`. В столбце хранится рефлексивный внешний ключ.

clown_info

id	name	boss_id
1	Элси	3
2	Пиклз	5
3	Снаглз	10
4	Мистер Хобо	3
5	Кларабелл	10
6	Скутер	3
7	Зиппо	3
8	Бэйб	5
9	Бонзо	5
10	Мистер Снифлз	10

По ссылке на поле «`id`» в той же таблице можно определить, кто из клоунов является начальником Элси.

И снова в столбце «`boss_id`» Мистера Снифлза хранится его собственный идентификатор.

Соединение таблицы с ней самой

Допустим, мы хотим вывести список всех клоунов и их начальников. Список всех клоунов с идентификаторами начальников легко выводится запросом SELECT:

```
SELECT name, boss_id FROM clown_info;
```

Но нам нужны пары имен клоуна и его начальника.

name	boss
Элси	Снаглз
Пиклз	Кларабелл
Снаглз	Мистер Снифлз
Мистер Хобо	Снаглз
Кларабелл	Мистер Снифлз
Скутер	Снаглз
Зиппо	Снаглз
Бэйб	Кларабелл
Бонзо	Кларабелл
Мистер Снифлз	Мистер Снифлз

Возьми в руку карандаш



Имеются две идентичные таблицы, `clown_info1` и `clown_info2`. Напишите соединение для получения таблицы результатов с именами каждого клоуна и его начальника.

clown_info1

id	name	boss_id
1	Элси	3
2	Пиклз	5
3	Снаглз	10
4	Мистер Хобо	3
5	Кларабелл	10
6	Скутер	3
7	Зиппо	3
8	Бэйб	5
9	Бонзо	5
10	Мистер Снифлз	10

clown_info2

id	name	boss_id
1	Элси	3
2	Пиклз	5
3	Снаглз	10
4	Мистер Хобо	3
5	Кларабелл	10
6	Скутер	3
7	Зиппо	3
8	Бэйб	5
9	Бонзо	5
10	Мистер Снифлз	10



Возьми в руку карандаш

Решение

Имеются две идентичные таблицы, `clown_info1` и `clown_info2`.
Напишите соединение для получения таблицы результатов с именами каждого клоуна и его начальника.

`clown_info1`

id	name	boss_id
1	Элси	3
2	Пиклз	5
3	Снаглз	10
4	Мистер Хобо	3
5	Кларабелл	10
6	Скутер	3
7	Зиппо	3
8	Бэйб	5
9	Бонзо	5
10	Мистер Снифлз	10

`clown_info2`

id	name	boss_id
1	Элси	3
2	Пиклз	5
3	Снаглз	10
4	Мистер Хобо	3
5	Кларабелл	10
6	Скутер	3
7	Зиппо	3
8	Бэйб	5
9	Бонзо	5
10	Мистер Снифлз	10

```
SELECT c1.name, c2.name AS boss
FROM clown_info1 c1
INNER JOIN clown_info2 c2
ON c1.boss_id = c2.id;
```

Чтобы не запутаться в двух столбцах с именами «name» мы назначаем второму псевдоним «boss».

Здесь проверяется совпадение «boss_id» из таблицы `clown_info1` с «id» из таблицы `clown_info2`.

Потребуется самосоединение

В только что выполненном упражнении одна таблица использовалась дважды. Но в нормализованной базе не может быть двух копий одной таблицы. Вместо этого для *имитации соединения двух таблиц* используется **самосоединение**.

Этот запрос очень похож на предыдущее задание, но имеет существенное отличие.

```
SELECT c1.name, c2.name AS boss
FROM clown_info c1
INNER JOIN clown_info c2
ON c1.boss_id = c2.id;
```

Таблица `clown_info` используется дважды, с синонимами `c1` (откуда берется `boss_id`) и `c2` (откуда берется имя начальника).

`clown_info`

id	name	boss_id
1	Элси	3
2	Пиклз	5
3	Снаглз	10
4	Мистер Хобо	3
5	Кларабелл	10
6	Скутер	3
7	Зиппо	3
8	Бэйб	5
9	Бонзо	5
10	Мистер Снифлз	10

Вместо двух идентичных таблиц мы используем `clown_info` дважды: сначала с псевдонимом `c1`, а затем с псевдонимом `c2`. Далее столбец `boss_id` (из `c1`) связывается с именем начальника (из `c2`) посредством внутреннего соединения.

name	boss
Элси	Снаглз
Пиклз	Кларабелл
Снаглз	Мистер Снифлз
Мистер Хобо	Снаглз
Кларабелл	Мистер Снифлз
Скутер	Снаглз
Зиппо	Снаглз
Бэйб	Кларабелл
Бонзо	Кларабелл
Мистер Снифлз	Мистер Снифлз

Данные столбца образуются внутренним соединением `boss_id` первого экземпляра таблицы `clown_info` (`c1`) и именем начальника, которое берется из таблицы `clown_info` (`c2`).

При самосоединении запрос к одной таблице строится так, как если бы она была двумя таблицами, содержащими одинаковую информацию.

Другой способ получения многотабличной информации

Я пытаюсь построить полный список всех должностей, встречающихся в gregs_list. Как узнать, какие должности встречаются во всех трех таблицах?



Три таблицы, о которых говорит Грег.

Текущая работа.

job_current
contact_id
title
salary
start_date

Работа, которую ищут претенденты.

job_desired
contact_id
title
salary_low
salary_high
available
years_exp

Доступные вакансии.

job_listings
job_id
title
salary
zip
description

Пока он написал три отдельные команды SELECT:

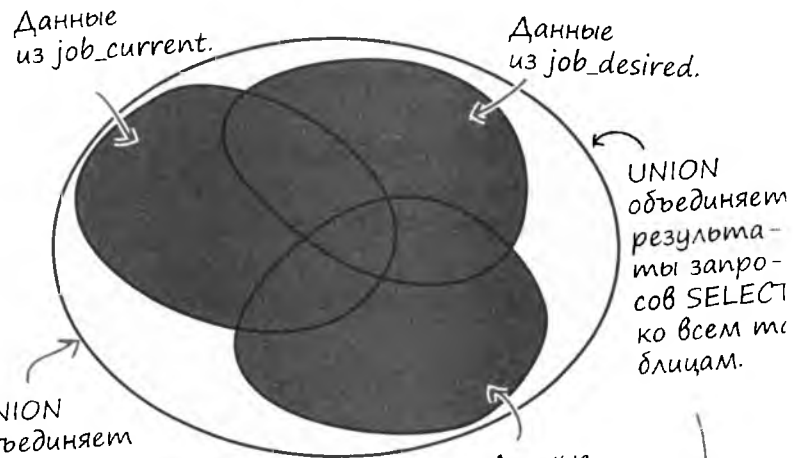
```
SELECT title FROM job_current;
SELECT title FROM job_desired;
SELECT title FROM job_listings;
```

Команды работают, но Грег хочет объединить результаты в одном запросе и получить список всех должностей, присутствующих хотя бы в одной из трех таблиц.

Союзы

Существует еще один способ получения объединенных результатов таблиц — так называемые союзы (ключевое слово UNION).

Союз объединяет в одну таблицу результаты двух и более запросов на основании того, что указано в запросе SELECT. Союзы можно трактовать как «пересекающиеся» значения всех запросов.



```
SELECT title FROM job_current
UNION
SELECT title FROM job_desired
UNION
SELECT title FROM job_listings;
```

UNION объединяет результаты этих трех разных запросов в одну общую таблицу результатов.

Данные из job_listings.

Небольшая часть сомен должностей, присутствующих в объединенных результатах из трех таблиц.

title
Бухгалтер
Адвокат
Программист
Веб-дизайнер
Менеджер
Шеф-повар
Психолог
Парикмахер
Учитель
Писатель

Грег замечает, что в результатах нет дубликатов, однако должности перечислены не по порядку, поэтому он пытается повторить запрос с добавлением условия ORDER BY в каждой команде SELECT.

```
SELECT title FROM job_current ORDER BY title
UNION
SELECT title FROM job_desired ORDER BY title
UNION
SELECT title FROM job_listings ORDER BY title;
```

Грег добавил ORDER BY в каждую команду, чтобы должности в таблице результатов были упорядочены по алфавиту.



Как вы думаете, что произошло при выполнении нового запроса?

Ограничения союзов

Запрос Грега не сработал! РСУБД выдала сообщение об ошибке, потому что она не знала, как интерпретировать многократное повторение ORDER BY.

При использовании UNION допускается **только одно** условие **ORDER BY** в **конце команды**. Это объясняется тем, что союз **объединяет и группирует результаты** нескольких команд **SELECT**.

Есть многое на свете, что вам следует знать о союзах.

Кодекс союзов в SQL

Количество столбцов в командах **SELECT** должно **быть одинаковым**. Нельзя выбрать два столбца одной командой и еще один столбец другой.

Команды **SELECT** должны содержать одинаковые выражения и агрегатные функции.

Команды **SELECT** могут следовать в любом порядке; на результаты это не влияет.

Кодекс союзов в SQL

По умолчанию SQL исключает дубликаты из результатов союзов

Типы данных в столбцах должны либо совпадать, либо быть совместимыми.

Если вы по какой-то причине **ХОТИТЕ** получить список со всеми дубликатами, используйте оператор **UNION ALL**. Он возвращает все совпадения, не только уникальные.

Правила союзов в действии

Количество столбцов в командах SELECT должно быть одинаковым. Нельзя выбрать два столбца одной командой и еще один столбец другой.

В каждой команде SELECT должно использоваться одинаковое количество столбцов.

```
SELECT title FROM job_current
UNION
SELECT title FROM job_desired
UNION
SELECT title FROM job_listings
ORDER BY title;
```

Если вы хотите упорядочить результаты, включите ORDER BY за последней из объединяемых команд SELECT. Это приведет к упорядочению всего списка результатов.

title
Адвокат
Клоун
Механик
Нейрохирург
Парикмахер
Пекарь
Психиатр
Психолог
Тренер
Ювелир

Результаты, которые нам хотелось бы получить, выглядят примерно так.

В этом примере все три столбца относятся к одному типу данных VARCHAR. В результате столбец, возвращаемый запросом, тоже относится к типу VARCHAR.



Как вы думаете, что произойдет, если объединяемые столбцы относятся к разным типам данных?

UNION ALL

UNION ALL работает точно так же, как UNION, если не считать того, что он возвращает все значения из столбцов — вместо одного экземпляра из каждой группы дубликатов.

На этот раз мы хотим получить все значения, хранящиеся в столбцах «title» всех трех таблиц.

```

SELECT title FROM job_current
UNION ALL
SELECT title FROM job_desired
UNION ALL
SELECT title FROM job_listings
ORDER BY title;

```

title
Адвокат
Адвокат
Клоун
Механик
Нейрохирург
Нейрохирург
Нейрохирург
Парикмахер
Парикмахер
Пекарь
Психиатр
Психолог
Психолог
Психолог
Психолог
Тренер
Ювелир

В этом списке одна должность может встречаться несколько раз.

До настоящего момента в наших союзах использовались столбцы с совпадающим типом данных. Однако в некоторых ситуациях может возникнуть необходимость в создании союза из разнотипных столбцов.

Когда мы говорим, что типы данных должны быть совместимы друг с другом, это означает, что при необходимости их можно привести к общему типу; а если этого сделать не удастся, выполнение запроса приведет к ошибке.

Допустим, в союзе тип INTEGER объединяется с типом VARCHAR. Так как данные VARCHAR нельзя преобразовать в целое число, в полученных результатах тип INTEGER будет преобразован в VARCHAR.

Создание таблицы на основе союза

Чтобы узнать, какой тип данных будет возвращен в составе союза, необходимо каким-то образом сохранить эти данные. Команда `CREATE TABLE AS` позволяет сохранить результаты и проанализировать их более подробно.

Команда `CREATE TABLE AS` получает результаты запроса `SELECT` и строит на их основе таблицу. В следующем примере данные союза столбцов `title` размещаются в новой таблице с именем `my_union`.

Имя новой таблицы.

```
CREATE TABLE my_union AS
SELECT title FROM job_current UNION
SELECT title FROM job_desired
UNION SELECT title FROM job_listings;
```

Уже знакомый союз. Таблица может создаваться на основе любых команд `SELECT`.

Возьми в руку карандаш



Создайте союз из столбцов `contact_id` (таблица `job_current`) и `salary` (таблица `job_listings`).

Как вы думаете, к какому типу данных будет относиться результат? Напишите команду `CREATE TABLE AS` для сохранения результатов союза.

Выведите описание таблицы командой `DESC` и проверьте правильность своего предположения.

→ Ответ на с. 479.

INTERSECT и EXCEPT

Конструкции INTERSECT и EXCEPT, в отличие от UNION, используются для поиска перекрывающихся результатов запросов.

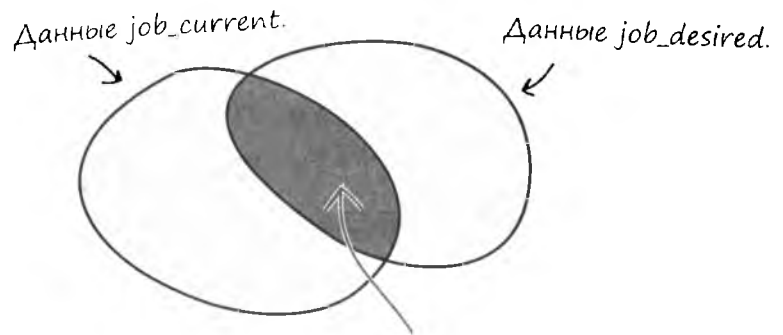
INTERSECT возвращает данные из результатов первого запроса, которые также присутствуют во втором запросе.



Будьте осторожны!

Эти две операции НЕ ПОДДЕРЖИВАЮТСЯ в MySQL.

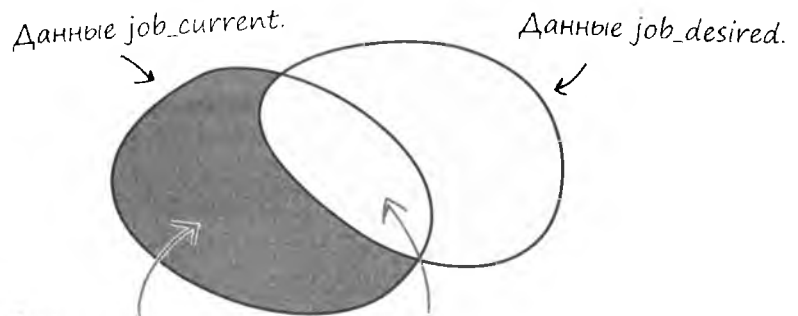
```
SELECT title FROM job_current  
INTERSECT  
SELECT title FROM job_desired;
```



Чтобы данные были включены в результате, они должны присутствовать в обеих таблицах.

EXCEPT возвращает только те значения, которые присутствуют в результатах первого запроса, но не встречаются в результатах второго.

```
SELECT title FROM job_current  
EXCEPT  
SELECT title FROM job_desired;
```



Только те данные, которые НЕ ВСТРЕЧАЮТСЯ в таблице, указанной после ключевого слова EXCEPT.

Данные, встречающиеся в обеих таблицах, исключаются из результатов.

С союзами разобрались, пора переходить к...



Погодите, так же нельзя. Вы сказали, что союзы и подзапросы делают одно и то же. Это нужно доказать.

(Эээ... То есть мы имели в виду...)

Сравнение подзапросов и соединений

Практически все, что делается при помощи подзапросов, может быть сделано с использованием соединений. Давайте вернемся к началу главы 9.

знакомство с подзапросами
подзапросы

Подзапросы

Чтобы сделать то, что делается двумя запросами, всего в одном запросе, нам понадобится вложить в него подзапрос.

Второй запрос, в котором извлекаются сведения из таблицы professions, мы назовем **ВНЕШНИЙ** запросом, потому что в него «упакован» другой, **ВНУТРЕННИЙ** запрос. Давайте посмотрим, что происходит.

ВНЕШНИЙ запрос

```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title
FROM job_current AS jc NATURAL JOIN my_contacts AS mc
WHERE
jc.title IN (SELECT title FROM job_listings);
```

Это **внешний** запрос.

Эту часть можно выделить и сделать независимым запросом, хотя роль станет **внутренним** запросом.

Два запроса преобразуются в запрос с подзапросом

Фактически мы всего лишь объединим два запроса в один. Первый запрос называется **внешним**, а второй — **внутренним**.

ВНЕШНИЙ запрос

+

ВНУТРЕННИЙ запрос

=

Запрос с подзапросом

```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title
FROM job_current AS jc NATURAL JOIN my_contacts AS mc
WHERE jc.title IN (SELECT title FROM job_listings);
```

Для объединения запросов преобразуются в запрос с подзапросом.

Нам больше не придется объединять результаты двух запросов. Вместо этого мы будем использовать **внутренний** запрос подзапросом. Давайте же посмотрим!

Результаты, полученные при выполнении этого запроса, в точности совпадают с результатами при явном перечислении **всех** записей в условии WHERE, но собирать приходится намного меньше.

То же относится и ко **всему** в одном запросе!

Внутренний запрос

```
SELECT title FROM job_listings;
```

Эта часть первого запроса преобразуется во **внутренний** запрос (или подзапрос).

Подзапрос представляет собой запрос, «упакованный» в другой запросе. Также он может называться «внутренним запросом».

mc.first_name	mc.last_name	mc.phone	jc.title
Джо	Лонниган	(555) 555-3214	Повар
Венди	Хиллерман	(555) 555-8976	Официант
Шон	Миллер	(555) 555-4443	Веб-дизайнер
Джаред	Колузи	(555) 555-5674	Веб-разработчик
Хуан	Гарда	(555) 555-0098	Веб-разработчик

Преобразование подзапроса в соединение

Первый подзапрос, созданный нами в главе 9, выглядел так.

Внешний
запрос.

```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title  
FROM job_current AS jc NATURAL JOIN my_contacts AS mc  
WHERE jc.title IN (SELECT title FROM job_listings);
```

При выполнении запроса был по-
лучен следующий результат.

Внутренний
запрос.

mc.first_name	mc.last_name	mc.phone	jc.title
Джо	Лонниган	(555) 555-3214	Повар
Венди	Хиллерман	(555) 555-8976	Официант
Шон	Миллер	(555) 555-4443	Веб-дизайнер
Джаред	Колуэй	(555) 555-5674	Веб-разработчик
Хуан	Гарза	(555) 555-0098	Веб-разработчик

Возьми в руку карандаш



Вот как выглядит условие WHERE, если переписать запрос в виде внутреннего соединения.

```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title  
FROM job_current AS jc NATURAL JOIN my_contacts AS mc  
INNER JOIN job_listings jl  
ON jc.title = jl.title;
```

Секцию WHERE с подза-
просом можно заменить
внутренним соединением.

Объясните, почему конструкция INNER JOIN этого запроса возвращает те же результаты, что и подзапрос.

Какой из двух запросов кажется вам более понятным?

Отвечь на с. 479.



Я написал уже немало подзапросов.
Нужно ли переписать их все в виде
соединений?

Нет — если все подзапросы делают именно то, что требуется, переписывать их не нужно.

И все же иногда существуют веские причины в пользу того или иного выбора...

Беседа у камина



Соединения и подзапросы › кто лучше?

Соединение

Безусловно, я лучше подхожу для большинства ситуаций. Меня проще понять и я обычно выполняюсь намного быстрее, чем какой-нибудь старый подзапрос.

И без вас прекрасно обходились. Прежде всего, меня проще понять.

Кто бы говорил — как насчет ваших **КОРРЕЛИРОВАННЫХ** и **НЕКОРРЕЛИРОВАННЫХ** разновидностей?

Подзапрос

Простите, это кто «старый»? Во многих РСУБД я не поддерживался до последнего времени. Меня пришлось **ДОБАВИТЬ**, потому что многие программисты хотели работать со мной.

Кого вы пытаетесь обмануть? Поди разберись в ваших **ВНУТРЕННИХ** и **ВНЕШНИХ** соединениях. Напустили туману...

Хорошо, у нас обоих есть свой жаргон; это правда. Но со мной можно сначала вычислить внутреннюю часть, а уже потом отдельно заняться внешней.

→ Продолжение на следующей странице.

Беседа у камина



Сегодняшняя тема: соединения и подзапросы. Кто лучше?

Соединение

Далеко не всегда, мистер КОРРЕЛИРОВАННЫЙ Подзапрос. Но пока довольно об этом. Я лучше подхожу, когда в результатах должны присутствовать столбцы из нескольких таблиц. Более того, я — единственное решение для подобных случаев.

Может, это и правда, но разобраться в том, что я делаю, не так уж сложно. Вы даже можете воспользоваться псевдонимами, чтобы вам не приходилось снова и снова вводить имена таблиц.

Ага, кое-кто слишком хорош для псевдонимов? И если уж вы намного проще меня, то как насчет коррелированных подзапросов? Они ничуть не проще того, что делаю я.

Тоже мне достижение.

Подзапрос

Да, и именно поэтому у вас нелады с агрегатными функциями — их нельзя использовать в условиях WHERE без подзапроса. Согласитесь, это отчасти компенсирует запрет на возвращение нескольких столбцов. Вы создаете слишком много трудностей.

Да, насчет псевдонимов — мне кажется, что они только усложняют понимание запроса. Просто для сведения: я ведь тоже могу ими пользоваться. Но в моем случае это делается куда более прямолинейно, причем в половине случаев псевдонимы и вовсе не нужны.

Эээ... верно. Но я знаю как минимум одно, что отличает меня от вас. Я могу использоваться в командах UPDATE, INSERT и DELETE.



Упражнение

Возьмите запросы с подзапросами из главы 9 и посмотрите, что лучше — переписать их заново с использованием соединений или оставить подзапросы.

Выведите все должности с зарплатой, равной наибольшей зарплате из таблицы `job_listings`.

```
SELECT title FROM job_listings WHERE salary = (SELECT
MAX(salary) FROM job_listings);
```

Решение с подзапросами удобнее?

Выведите имена и фамилии людей с зарплатой выше средней.

```
SELECT mc.first_name, mc.last_name FROM my_contacts mc
NATURAL JOIN job_current jc WHERE jc.salary > (SELECT
AVG(salary) FROM job_current);
```

Решение с подзапросами удобнее?



Упражнение Решение

Возьмите запросы с подзапросами из главы 9 и посмотрите, что лучше — переписать их заново с использованием соединений или оставить подзапросы.

Выведите все должности с зарплатой, равной наибольшей зарплате из таблицы `job_listings`.

```
SELECT title FROM job_listings WHERE salary = (SELECT  
MAX(salary) FROM job_listings);
```

```
.....  
SELECT title FROM job_listings  
ORDER BY salary DESC LIMIT 1;  
.....
```

↑
Чтобы запрос возвращал только одну запись — с наибольшей зарплатой.

Решение с подзапросами удобнее? Нет.

Выведите имена и фамилии людей с зарплатой выше средней.

```
SELECT mc.first_name, mc.last_name FROM my_contacts mc  
NATURAL JOIN job_current jc WHERE jc.salary > (SELECT  
AVG(salary) FROM job_current);
```

```
.....  
В отличие от предыдущей реализации, мы не можем  
использовать LIMIT и ORDER BY.  
.....
```

Решение с подзапросами удобнее?

Да.

←
В предыдущем решении мы могли использовать LIMIT для получения наибольшей зарплаты из упорядоченного списка зарплат. Зарплаты «выше средней» не упорядочиваются, поэтому мы не можем использовать LIMIT для их получения.

Самосоединение как подзапрос

Вы уже видели, как подзапрос преобразуется в соединение. Давайте посмотрим, как самосоединение преобразуется в подзапрос.

Помните столбец `boss_id`, добавленный в таблицу `clown_info`? Вот как выглядело самосоединение, в котором использовались два псевдонима `clown_info` — `c1` и `c2`.

Показывает, кто является начальником данного клоуна.

`clown_info`

id	name	boss_id
1	Элси	3
2	Пиклз	5
3	Снаглз	10
4	Мистер Хобо	3
5	Кларабелл	10
6	Скутер	3
7	Зиппо	3
8	Бэйб	5
9	Бонзо	5
10	Мистер Снифлз	10

ДО ПРЕОБРАЗОВАНИЯ

```
SELECT c1.name, c2.name AS boss
FROM clown_info c1
INNER JOIN clown_info c2
ON c1.boss_id = c2.id;
```

Первый экземпляр `clown_info`.

Второй экземпляр `clown_info`.

ПОСЛЕ ПРЕОБРАЗОВАНИЯ

Подзапрос, полученный в результате преобразования самосоединения, является коррелированным, потому что он зависит от результата внешнего запроса для получения правильного значения `boss_id` и находится в списке столбцов `SELECT`.

```
SELECT c1.name,
(SELECT name FROM clown_info
WHERE c1.boss_id = id) AS boss
FROM clown_info c1;
```

Внешний запрос.

Подзапрос в списке столбцов `SELECT`.

Подзапрос зависит от результатов внешнего запроса для получения правильного значения `boss_id`, поэтому он является коррелированным.

Компания Грега растет

Грег занят изучением соединений и подзапросов. Он нанял нескольких друзей, которые должны помочь ему с менее сложными запросами.



Жаль, что новые работники плохо понимают, что творят. Вскоре Грег узнает, что происходит при одновременной работе с базой данных нескольких людей, плохо знающих SQL.

Возьми в руку карандаш Со с. 469.



Решение

Создайте союз из столбцов `contact_id` (таблица `job_current`) и `salary` (таблица `job_listings`).

```
SELECT contact_id FROM job_current UNION
SELECT salary FROM job_listings;
```

Как вы думаете, к какому типу данных будет относиться результат? Напишите команду `CREATE TABLE AS` для сохранения результатов союза.

```
CREATE TABLE my_table SELECT contact_id
FROM job_current UNION SELECT salary FROM
job_listings;
```

Выведите описание таблицы командой `DESC` и проверьте правильность своего предположения.

```
DEC(12,2)
```

Возьми в руку карандаш Со с. 472.



Решение

А вот как выглядит условие `WHERE`, если переписать запрос в виде внутреннего соединения:

```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title
FROM job_current AS jc NATURAL JOIN my_contacts AS mc
INNER JOIN job_listings jl
ON jc.title = jl.title;
```

Секцию `WHERE` с подзапросом можно заменить внутренним соединением.

Объясните, почему конструкция `INNER JOIN` этого запроса возвращает те же результаты, что и подзапрос.

Внутреннее соединение включает результаты только при выполнении условия `jc.title = jl.title`, что эквивалентно секции `WHERE` с подзапросом:

```
WHERE jc.title IN (SELECT title FROM job_listings);
```

Какой из двух запросов кажется вам более понятным?

Единственно правильного ответа здесь быть не может! Но ваш ответ показывает, что вы уже начали думать о том, какой способ будете использовать в будущем со своими данными.



Новые инструменты

Ваши познания в SQL стремительно растут. Вы освоили внешние соединения, самосоединения и союзы и даже знаете, как преобразовать соединение в запрос, и наоборот. Полный список инструментов приведен в приложении III.

РЕФЛЕКСИВНЫЙ ВНЕШНИЙ КЛЮЧ

Внешний ключ той же таблицы, в которой он является первичным ключом, используемый для других целей.

ЛЕВОЕ ВНЕШНЕЕ СОЕДИНЕНИЕ

Левое внешнее соединение перебирает все записи ЛЕВОЙ таблицы и ищет для них соответствия среди записей ПРАВОЙ таблицы.

UNION и UNION ALL

Союз (UNION) объединяет в одну таблицу результаты двух и более запросов на основании списков столбцов в командах SELECT.

С ключевым словом UNION результаты не содержат дубликатов, а конструкция UNION ALL разрешает присутствие дубликатов.

САМОСОЕДИНЕНИЕ

Способ построения запроса к одной таблице так, как если бы она была двумя таблицами, содержащими одинаковую информацию.

ПРАВОЕ ВНЕШНЕЕ СОЕДИНЕНИЕ

Правое внешнее соединение перебирает все записи ПРАВОЙ таблицы, и ищет для них соответствия среди записей ЛЕВОЙ таблицы.

INTERSECT

Ключевое слово возвращает только те значения, которые присутствуют в первом и во втором запросе.

EXCEPT

Ключевое слово возвращает только те значения, которые присутствуют в первом, но НЕ во втором запросе.

CREATE TABLE AS

Команда, используемая для создания таблицы по результатам выполнения команды SELECT.

11 Ограничения, представления и транзакции

У семи нянек

Видите, проблема вот здесь — вы в графе «количество» написали «все сразу»...

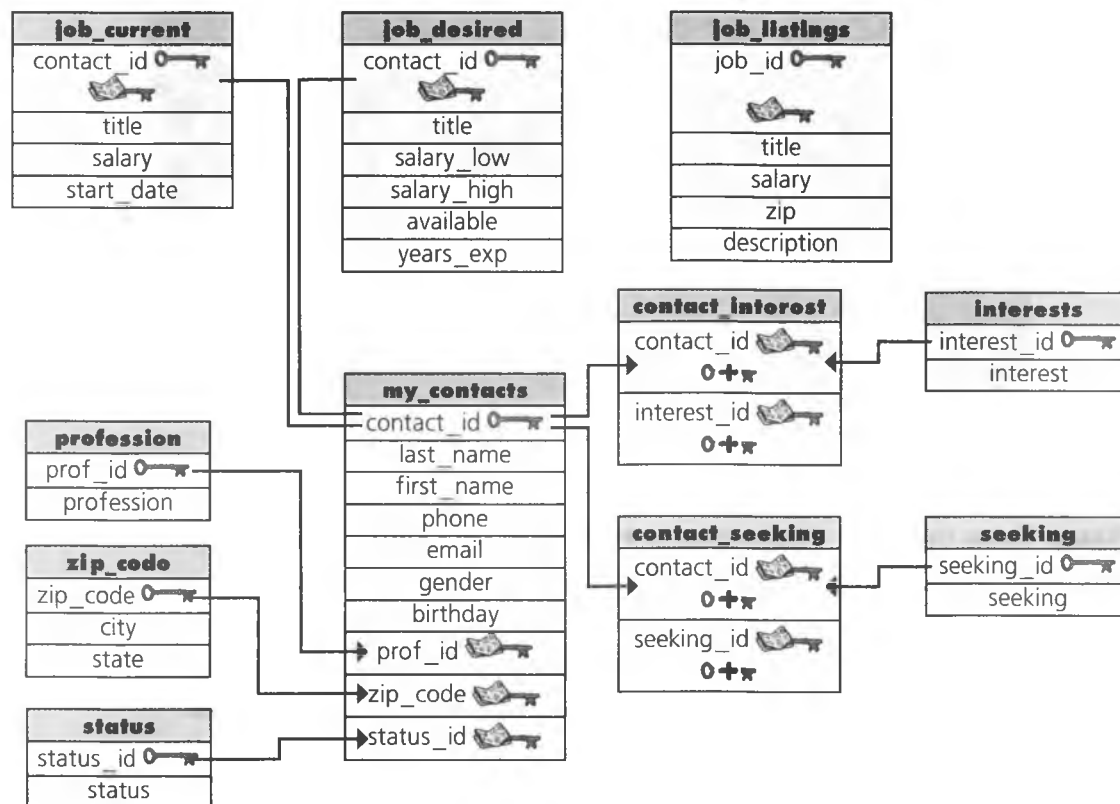


Ваша база данных выросла, и теперь с ней будут работать другие люди. К сожалению, далеко не все они так же хорошо разбираются в SQL, как вы. Вам придется позаботиться о том, чтобы **предотвратить ввод неверных данных, запретить просмотр лишних данных, а также предотвратить возможные конфликты при одновременном вводе данных.** В этой главе мы займемся защитой данных от чужих ошибок. Итак — Защита Ваз Данных, часть 1.

Грег нанимает помощников

Грег нанял двух работников, которые должны помогать ему в ведении развивающегося бизнеса. Джим будет заниматься вводом данных новых клиентов, а Фрэнк — поиском для них подходящей работы.

Грег провел предварительный инструктаж, на котором объяснил структуру базы данных и назначение каждой из таблиц.



Первый день: вставка данных нового клиента

Джим сидит в своем новом офисе и получает сообщение от Грегa.

Чат: данные для вставки

Грег: Привет, Джим, можешь добавить запись в базу данных?

Джим: Конечно, могу.

Грег: Здесь только часть информации, остальное пришлю позже.

Грег: Пэт Мэрфи, 555-1239, patmurphy@someemail.com, почтовый индекс 10087

Грег: Дата рождения — 15/4/1978.

Грег: Профессия — учитель, семейное положение — в браке не состоит. Чтобы получить правильные значения, используй запросы SELECT — описание синтаксиса найдешь в моих заметках.

Джим: Вроде все просто. Уже делаю :)

Грег: Спасибо!

Не стоит благодарности |



А вы сможете написать запросы для вставки данных нового клиента в базу данных?

Джим не хочет использовать NULL

В процессе ввода данных Джим осознает, что он даже не знает — Пэт это мужчина или женщина? Грега поблизости нет, решение приходится принимать самому. Джим решает ввести в столбце gender значение «X».

Он пишет вспомогательные запросы для получения недостающих данных, например:

Я слышал, что значения NULL в базе данных нежелательны, но я пока не знаю, какой пол указывать в этой записи.



Значение prof_id берется из таблицы profession.

```
SELECT prof_id FROM profession WHERE profession = 'Учитель';
```

prof_id
19

Идентификатор, соответствующий профессии «учитель», для последующего использования в запросе к my_contacts.

my_contacts	
contact_id	🔑
last_name	
first_name	
phone	
email	
gender	
birthday	
prof_id	🔑
zip_code	🔑
status_id	🔑

Значение status_id берется из таблицы status.

```
SELECT status_id FROM profession WHERE status = '---';
```

status_id
4

Далее Джим создает запись с буквой «X» в столбце gender.

Для столбцов AUTO_INCREMENT задавать значение не нужно. Два апострофа означают, что значение первичного ключа должно генерироваться автоматически.

```
INSERT INTO my_contacts VALUES(' ', 'Мэрфи', 'Пэт',
'5551239', 'patmurphy@someemail.com', 'X', 1978-15-04,
19, '10087', 3);
```

Идентификаторы, найденные при помощи вспомогательных запросов. Также можно было использовать подзапросы.

Фиктивное обозначение пола, которое Джим решает ввести в столбце «gender» — он не хочет выбирать наугад или вводить NULL.

Три месяца спустя

Грег занимается сбором демографических данных. Он хочет знать, сколько в таблице `my_contacts` мужчин, сколько женщин и сколько всего в ней записей. Для этого он выполняет три запроса.

```
SELECT COUNT(*) AS Females FROM my_contacts WHERE gender = 'Ж';
```

Females
5975

← Таблица `my_contacts` содержит 5975 записей с буквой «Ж» в столбце «`gender`».

```
SELECT COUNT(*) AS Males FROM my_contacts WHERE gender = 'М';
```

Males
6982

← И 6982 записи с буквой «М».

```
SELECT COUNT(*) AS Total FROM my_contacts;
```

Total
12970

← Этот запрос возвращает общее количество записей в таблице.

Грег замечает, что сумма не совпадает с общим количеством записей. В таблице содержатся 13 записей, которые не учитываются ни первым, ни вторым запросом. Он пытается ввести другой запрос:

```
SELECT gender FROM my_contacts
WHERE gender <> 'М' AND gender <> 'Ж';
```

gender
X
X
X
X
X
X
X
X
X
X
X
X
X

← В поисках пропавших данных он обнаруживает 13 записей, у которых столбец «`gender`» содержит «X».



Мог ли Джим обойтись без фиктивных значений X?

Добавление ограничения CHECK

Ограничения столбцов уже встречались нам в предшествующих главах. **Ограничение** определяет набор значений, которые могут вставляться в столбец, и устанавливается при создании таблицы. В частности мы уже рассматривали ограничения NOT NULL, PRIMARY KEY, FOREIGN KEY и UNIQUE.

Существует еще одна категория ограничений столбцов — так называемые ограничения проверки (**CHECK**). Допустим, в базе данных должна сохраняться информация о монетках, которые попадают в копилку. Копилка принимает только монетки по 1, 5, 10 и 25 центов, которые в базе данных обозначаются буквами P, N, D и Q соответственно. Следующая таблица использует ограничение CHECK для проверки значений, которые могут вставляться в столбец coin.

```
CREATE TABLE piggy_bank
(
  id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
  coin CHAR(1) CHECK (coin IN ('P', 'N', 'D', 'Q'))
)
```

Значением столбца «coin» может быть только одна из этих букв.

Если значение, которое вставляется в столбец, нарушает условие CHECK, вы получите сообщение об ошибке.

Ограничение CHECK определяет значения, которые могут вставляться в столбец базы данных. В нем используются те же условные операторы, что и в условии WHERE.



**Будьте
осторожны!**

CHECK не обеспечивает целостности данных в MySQL.

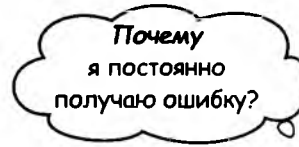
MySQL позволяет создавать таблицы с ограничениями CHECK, но эти ограничения не работают. MySQL их игнорирует.

Ограничение CHECK для столбца gender

Если бы Грег мог вернуться в прошлое, то он бы создал таблицу `my_contacts` с ограничением `CHECK` для столбца `gender`. Впрочем, он может решить проблему командой `ALTER TABLE`.

```
ALTER TABLE my_contacts  
ADD CONSTRAINT CHECK gender IN ('М', 'Ж');
```

На следующий день Джим обнаруживает, что он уже не может создать запись, в которой столбец `gender` содержит фиктивное значение «X». Он обращается с вопросом к Грегу. Тот рассказывает о новом ограничении, а поскольку вернуться в прошлое все равно не удастся — поручает Джиму связаться со всеми претендентами «X» и узнать, какого они пола.



Возьми в руку карандаш



Запишите, какие значения разрешены в каждом из следующих столбцов.

```
CREATE TABLE mystery_table  
(  
  column1 INT(4) CHECK (column1 > 200),  
  column2 CHAR(1) CHECK (column2 NOT IN ('x', 'y', 'z')),  
  column3 VARCHAR(3) CHECK ('A' = SUBSTRING(column_3, 1, 1)),  
  column4 VARCHAR(3) CHECK ('A' = SUBSTRING(column_4, 1, 1)  
  AND '9' = SUBSTRING(column_4, 2, 1))  
)
```

Столбец `column1`:

Столбец `column2`:

Столбец `column3`:

Столбец `column4`:

Возьми в руку карандаш



Решение

Запишите, какие значения разрешены в каждом из следующих столбцов.

```
CREATE TABLE mystery_table
(
  column1 INT(4) CHECK (column1 > 200),
  column2 CHAR(1) CHECK (column2 NOT IN ('x', 'y', 'z')),
  column3 VARCHAR(3) CHECK ('A' = SUBSTRING(column_3, 1, 1)),
  column4 VARCHAR(3) CHECK ('A' = SUBSTRING(column_4, 1, 1)
  AND '9' = SUBSTRING(column_4, 2, 1))
)
```

Разрешены комбинированные условия со связками AND и OR.

Столбец column1: Любое число больше 200

Столбец column2: Любым символом, кроме x, y и z

Столбец column3: Первым символом строки должна быть буква A

Столбец column4: Первым символом строки должна быть буква A, а вторым — цифра 9

Часто задаваемые вопросы

В: Значит, в CHECK можно использовать все то, что разрешено в условии WHERE?

О: Практически все. Вы можете использовать все условные конструкции: AND, OR, IN, NOT, BETWEEN и т. д., и даже объединять их, как в приведенном выше примере. Впрочем, подзапросы запрещены.

В: Если ограничения CHECK нельзя использовать в MySQL, то чем их заменить?

О: Простого ответа на этот вопрос не существует. Иногда используются триггеры — запросы, выполняемые автоматически при наступлении некоторого события. Но триггеры далеко

не так просты, как ограничения CHECK, а тонкости их использования выходят за рамки этой книги.

В: Что произойдет при попытке вставить значение, не прошедшее проверку CHECK?

О: Вы получите сообщение об ошибке, а данные вставлены не будут.

В: И какая от этого польза?

О: Гарантия того, что в таблицу попадут только осмысленные данные и она не будет забиваться фиктивными значениями.

Однообразная работа Фрэнка

Фрэнк работает над поиском вакансий. Постепенно он заметил некоторые закономерности: вакансий веб-дизайнеров много, а претендентов — мало. Технических писателей в поисках работы много, а вакансий — мало.

Фрэнк ежедневно выполняет одни и те же запросы, пытаясь подобрать для каждого подходящую работу.

Мне приходится раз за разом создавать одинаковые запросы, и так каждый день. Мне это надоело.

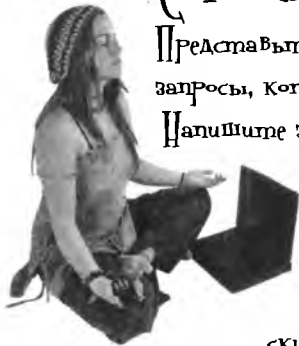


Станьте Фрэнком

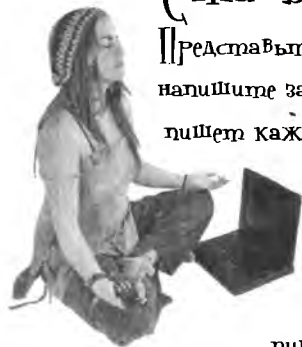
Представьте себя на месте Фрэнка и напишите запросы, которые Фрэнк пишет каждый день.

Напишите запрос для выборки всех записей веб-дизайнеров из таблицы `job_desired` вместе с их контактными данными.

Напишите другой запрос для поиска открытых вакансий для технических писателей.



Станьте Фрэнком. Ответ

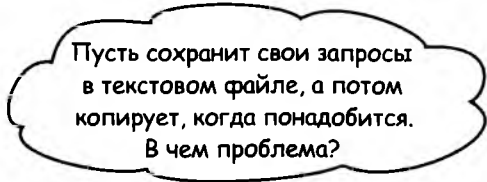


Представьте себя на месте Фрэнка и напишите запросы, которые Фрэнк пишет каждый день. Напишите запрос для выборки всех записей веб-дизайнеров из таблицы `job_desired` вместе с их контрактными данными. Напишите другой запрос для поиска открытых вакансий для технических писателей.

```
SELECT mc.first_name, mc.last_name, mc.phone, mc.email
FROM my_contacts mc
NATURAL JOIN job_desired jd
WHERE jd.title = 'Веб-дизайнер';
```

```
SELECT title, salary, description, zip
FROM job_listings
WHERE title = 'Технический писатель';
```

Запросы несложные, но если Фрэнку придется вводить их снова и снова, он почти наверняка совершит какую-нибудь ошибку. Ему нужно найти способ сохранения запросов, чтобы в один прекрасный день просмотреть результаты, не вводя запросы заново.



Пусть сохранит свои запросы в текстовом файле, а потом копирует, когда понадобится. В чем проблема?



Файлы могут быть случайно стерты или изменены.

Существует другой, куда более удобный способ сохранения запросов в самой базе данных. Для этого запросы преобразуются в **представления**.

Создание представления

Создать представление совсем несложно – достаточно добавить в запрос ключевые слова CREATE VIEW. Давайте создадим два представления для запросов Фрэнка.

```
CREATE VIEW web_designers AS
SELECT mc.first_name, mc.last_name, mc.phone, mc.email
FROM my_contacts mc
NATURAL JOIN job_desired jd
WHERE jd.title = 'Веб-дизайнер';
```

Также можно воспользоваться внутренним соединением с условием `ON mc.contact_id = jd.contact_id.`

```
CREATE VIEW tech_writer_jobs AS
SELECT title salary, description, zip
FROM job_listings
WHERE title = 'Технический писатель';
```



Действительно просто! Но как использовать представление, которое я создал?



Как по вашему мнению выглядит команда SQL, использующая представление?

Просмотр представлений

Возьмем только что созданное нами представление web_designers.

```
CREATE VIEW web_designers AS
SELECT mc.first_name, mc.last_name, mc.phone, mc.email
FROM my_contacts mc
NATURAL JOIN job_desired jd
WHERE jd.title = 'Веб-дизайнер';
```

Не забывайте: ключевое слово AS можно не указывать.

Чтобы посмотреть текущие результаты представления, мы просто выполняем с ним операции, как если бы оно было обычной таблицей. Например, можно воспользоваться командой SELECT:

```
SELECT * FROM web_designers;
```

Имя представления.

Результат:

first_name	last_name	phone	email
Джон	Мартинес	5559872	jm@someemail.com
Саманта	Хоффман	5556948	sammy@someemail.com
Тодд	Герц	5557888	tod@someemail.com
Фред	Макдугал	5557744	fm@someemail.com

И так далее — все записи, у которых в столбце «title» хранится значение «Веб-дизайнер».

Как работает представление

Представление, используемое в запросе, ведет себя так, словно оно является подзапросом. Вот что на самом деле означает только что выполненная нами команда SELECT с представлением:

```
SELECT * FROM web_designers;
```

Она означает: «Выбрать все записи из подзапроса, возвращающего имя, фамилию, телефон и адрес электронной почты всех людей из таблицы my_contacts, которые ищут вакансию веб-дизайнера».

```
SELECT * FROM
  (SELECT mc.first_name, mc.last_name, mc.phone, mc.email
   FROM my_contacts mc
  NATURAL JOIN job_desired jd
   WHERE jd.title = 'Веб-дизайнер') AS web_designers;
```

Часть, использованная в представлении.

Подзапросу назначается псевдоним, чтобы запрос интерпретировал его как таблицу.

А как насчет этой части — AS web_designers? Зачем она нужна?



В секции FROM должна быть указана таблица.

И хотя наша команда SELECT возвращает виртуальную таблицу, SQL не сможет узнать об этом без псевдонима.

Что такое представление

В сущности, представление — это таблица, существующая только во время использования представления в запросе. Представление называется **виртуальной таблицей**, потому что ведет себя как настоящая таблица и с ним можно выполнять те же операции, что и с обычными таблицами.

Но виртуальная таблица не хранится в базе данных. Она создается тогда, когда вы используете представление, а затем уничтожается. Остается только ключевое слово VIEW с именем. И это хорошо, потому что при каждой вставке новых записей в базу данных при использовании представления будет отображаться новая информация.



Почему представления удобны при работе с базами данных

1 Возможные изменения структуры базы данных не нарушат работы приложений, зависящих от таблиц.

Мы еще не говорили об этом в книге, но через какое-то время вы будете использовать свои познания SQL с другими технологиями для создания приложений. Создавая представления данных, вы сможете изменять базовую структуру таблиц — созданное представление будет имитировать прежнюю структуру таблицы, и вам не придется вносить изменения в приложение, работающее с данными.

Эти таблицы существуют только во время использования представления в запросах.

2 Приложения упрощают сложные запросы до простых команд.

Вам не придется многократно создавать сложные союзы и подзапросы — достаточно создать для них представление, скрывающее всю сложность ниже лежащего запроса. А когда код SQL будет использоваться в PHP или другом языке программирования, вам будет намного проще работать с представлением. Вы будете использовать упрощенный код представления вместо большого, сложного запроса с множеством соединений. Простота уменьшает вероятность ошибок, а код станет более удобочитаемым.

3 Представления могут скрывать информацию, которая не нужна пользователю.

Предположим, в базу данных `gregs_list` добавляются таблицы с данными кредитных карт. Вы можете создать представление, в котором будет указано, что данные карты хранятся в системе, не раскрывая ее данных. Работники будут видеть только ту информацию, которая им нужна, а конфиденциальные сведения останутся скрытыми от них.



А у меня сложный вопрос. Можно ли создать представление, которое покажет мне всех людей из таблицы `job_current`, которые также присутствуют в таблице `job_desired`, с их текущей зарплатой, минимальной желаемой зарплатой из поля `salary_low` и разностью этих двух чисел? Проще говоря, насколько они рассчитывают повысить свой заработок при смене работы? Да, и не забудьте включить их имена, адреса электронной почты и телефоны.



Упражнение

Задача не из простых, но любой запрос, который можно создать в форме `SELECT`, может быть преобразован в представление. Для начала ответьте на приведенные ниже вопросы, а затем запишите запрос Фрэнка в виде представления с именем `job_raises`.

Какие таблицы должны быть включены в запрос?

Какие столбцы этих таблиц могут использоваться для вычисления прибавки зарплаты?

Как средствами SQL создать в результатах столбец с именем `raise`?

Напишите запрос Фрэнка:

Подсказка. Попробуйте записать его с
.....
включением имени для каждой таблицы;



Упражнение Решение

Задача не из простых, но любой запрос, который можно создать в форме SELECT, может быть преобразован в представление. Для начала ответьте на приведенные ниже вопросы, а затем запишите запрос Фрэнка в виде представления с именем `job_raises`.

Какие таблицы должны быть включены в запрос?

`job_current`, `job_desired` и `my_contacts`

Какие столбцы этих таблиц могут использоваться для вычисления прибавки зарплаты?

Столбец «`salary`» таблицы `job_current` и столбец «`salary_low`» таблицы `job_desired`

Как средствами SQL создать в результатах столбец с именем `raise`?

Вычесть «`salary`» из «`salary_low`», и назначить разности псевдоним.

Напишите запрос Фрэнка:

```
CREATE VIEW job_raises AS
SELECT mc.first_name, mc.last_name, mc.email, mc.phone,
jc.contact_id, jc.salary, jd.salary_low,
jd.salary_low - jc.salary AS raise
FROM job_current jc
INNER JOIN job_desired jd
INNER JOIN my_contacts mc
WHERE jc.contact_id = jd.contact_id
AND jc.contact_id = mc.contact_id;
```

Здесь мы создаем новое представление с именем `job_raises`.

В остальном коде запроса два внутренних соединения используются для извлечения данных из трех таблиц. Простейшая математическая операция создает новый столбец «`raise`».

Текущая зарплата вычитается из ожидаемой, а результату назначается псевдоним «`raise`».

Запрос получился просто огромным, но для просмотра информации Фрэнку теперь достаточно ввести простую команду

```
SELECT * FROM job_raises;
```

Возьми в руку карандаш



Фрэнк выполняет запрос `SELECT` на с. 496 с использованием нового представления `job_raises`. Как ему упорядочить результаты в алфавитном порядке по фамилиям?

→ Ответ на с. 517.

Вставка, обновление и удаление в представлениях

Представления не ограничиваются одной лишь выборкой данных из таблиц. В некоторых ситуациях возможны также операции вставки, обновления и удаления данных.

Значит, я могу создать представление, которое позволит мне изменять содержимое таблиц?



Можете, но игра не стоит свеч.

Если в представлении используются агрегатные функции (такие, как `SUM`, `COUNT` или `AVG`), оно не может использоваться для обновления данных. Кроме того, если представление содержит условия `GROUP BY`, `DISTINCT` или `HAVING`, изменение данных также невозможно.

Как правило, операции `INSERT`, `UPDATE` и `DELETE` бывает удобнее выполнять традиционным способом, но мы рассмотрим пример обновления данных через представление.

Обновление данных через представление

Давайте создадим представление на базе новой таблицы с именем `piggy_bank`. В таблице хранятся данные о монетках, которые собираются в копилке. У каждой монетки имеется идентификатор, номинал (первая буква английского названия P, N, D или Q) и год выпуска.

```
CREATE TABLE piggy_bank
(
  id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
  coin CHAR(1) NOT NULL,
  coin_year CHAR(4)
)
```

В настоящий момент таблица `piggy_bank` содержит следующие данные:

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999
5	Q	1981
6	D	1940
7	Q	1980
8	P	2001
9	D	1926
10	P	1999

Создадим представление, которое выводит только записи 25-центовых монеток (Q):

```
CREATE VIEW AS pb_quarters
SELECT * FROM piggy_bank
WHERE coin = 'Q';
```



Как будет выглядеть таблица результатов при выполнении следующего запроса?

```
SELECT * FROM pb_quarters;
```



Упражнение

Создайте таблицу piggy_bank, а также представления pb_quarters и pb_dimes с использованием приведенных ниже запросов.

```
INSERT INTO piggy_bank VALUES ('','Q', 1950), ('','P', 1972), ('','N', 2005),
('','Q', 1999), ('','Q', 1981), ('','D', 1940), ('','Q', 1980), ('','P', 2001), ('','D',
1926), ('','P', 1999);
```

```
CREATE VIEW pb_quarters AS SELECT * FROM piggy_bank WHERE coin = 'Q';
```

```
CREATE VIEW pb_dimes AS SELECT * FROM piggy_bank WHERE coin = 'D' WITH CHECK OPTION;
```

Напишите, что произойдет при выполнении каждого из следующих запросов INSERT, DELETE и UPDATE. В конце упражнения запишите итоговое состояние таблицы piggy_bank.

↑
Попробуйте понять, что делает это условие, во время работы над упражнением.

```
INSERT INTO pb_quarters VALUES ('','Q', 1993);
```

```
.....
INSERT INTO pb_quarters VALUES ('','D', 1942);
```

```
INSERT INTO pb_dimes VALUES ('','Q', 2005);
```

```
DELETE FROM pb_quarters WHERE coin = 'N' OR coin = 'P' OR coin = 'D';
```

```
UPDATE pb_quarters SET coin = 'Q' WHERE coin = 'P';
```




Создайте таблицу piggy_bank, а также представления pb_quarters и pb_dimes с использованием приведенных ниже запросов.

```
INSERT INTO piggy_bank VALUES ('','Q', 1950), ('','P', 1972), ('','N', 2005),
('','Q', 1999), ('','Q', 1981), ('','D', 1940), ('','Q', 1980), ('','P', 2001), ('','D',
1926), ('','P', 1999);

CREATE VIEW pb_quarters AS SELECT * FROM piggy_bank WHERE coin = 'Q';

CREATE VIEW pb_dimes AS SELECT * FROM piggy_bank WHERE coin = 'D' WITH CHECK OPTION;
```

Напишите, что произойдет при выполнении каждого из следующих запросов INSERT, DELETE и UPDATE. В конце упражнения запишите итоговое состояние таблицы piggy_bank.

↑
Попробуйте понять, что делает это условие, во время работы над упражнением.

```
INSERT INTO pb_quarters VALUES ('','Q', 1993);
```

Запрос будет выполнен нормально.

```
INSERT INTO pb_quarters VALUES ('','D', 1942);
```

В таблицу вставляется новая запись, хотя из-за условия WHERE этого вроде бы быть не должно.

```
INSERT INTO pb_dimes VALUES ('','Q', 2005);
```

Ошибка из-за условия CHECK OPTION. Данные, вводимые в представления, проверяются по условию WHERE перед вставкой.

```
DELETE FROM pb_quarters WHERE coin = 'N' OR coin = 'P' OR coin = 'D';
```

Этот запрос не изменяет содержимое таблицы, потому что поиск осуществляется только в результатах с coin = 'Q'

```
UPDATE pb_quarters SET coin = 'Q' WHERE coin = 'P';
```

Ничего не делает с таблицей, потому что представление pb_quarters не возвращает значений с coin = 'P'.

Итоговое содержимое таблицы выглядит так:

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999
5	Q	1981
6	D	1940
7	Q	1980
8	P	2001
9	D	1926
10	P	1999
11	Q	1993
12	D	1942

Представление с CHECK OPTION

Условие CHECK OPTION, включенное в представление, указывает вашей РСУБД, что все команды INSERT и DELETE должны проверяться по условию WHERE в вашем представлении. Итак, как CHECK OPTION влияет на команды INSERT и UPDATE?

При использовании CHECK OPTION в предыдущем упражнении попытка выполнения INSERT отвергалась, если данные не соответствовали условию WHERE в представлении pb_dimes. Попытка выполнения UPDATE тоже приведет к ошибке:

```
UPDATE pb_dimes SET coin = 'x';
```

Значение 'x' не соответствует условию WHERE в pb_dimes, поэтому данные не обновляются.



Нельзя ли использовать представления с CHECK OPTION для создания чего-то наподобие CHECK CONSTRAINT, если вы работаете с MySQL?

Да, представления могут точно имитировать таблицы, но проверять команды INSERT на соответствие условиям WHERE.

Вспомните «проблему пола», о которой говорилось ранее в этой главе — на базе таблицы my_contacts можно создать представление, которое Джим будет использовать для обновления my_contacts. Представление будет выдавать ошибку каждый раз, когда Джим попытается вставить 'x' в столбец gender.

В MySQL можно имитировать CHECK CONSTRAINT при помощи CHECK OPTION.



Как создать для my_contacts представление, которое заставит Джима ввести в поле gender либо 'М', либо 'Ж'?

Конструкция CHECK OPTION проверяет каждую попытку выполнения вставки или обновления данных на соответствие условию WHERE представления.

Представление может быть обновляемым, если...

В примере с таблицей `riggy_bank` оба созданных нами представления были обновляемыми. **Обновляемое представление** позволяет изменять таблицы, на которых оно основано. Важно, чтобы обновляемое представление включало все столбцы `NOT NULL` из своих таблиц. В этом случае при вставке с использованием представления можно быть уверенным в том, что каждый столбец, который обязательно должен быть заполнен, получит свое значение.

По сути это означает, что с созданными нами представлениями могут выполняться все команды: `INSERT`, `UPDATE` и `DELETE`. Если в представление включаются все столбцы `NOT NULL`, оно сможет предоставить все необходимые данные для своей таблицы.

Также существуют **необновляемые представления**, в которые включаются не все столбцы `NOT NULL`. Кроме создания и удаления, с необновляемыми представлениями может выполняться только операция выборки.

Обновляемое представление включает все столбцы `NOT NULL` из своих таблиц.

Если не считать `CHECK OPTION`, я не вижу особой пользы от использования `INSERT` в представлениях.

Верно, представления довольно редко используются для выполнения команд `INSERT`, `UPDATE` и `DELETE`.

Хотя у них имеются свои полезные применения (например, обеспечение целостности данных в MySQL), обычно проще использовать `INSERT`, `UPDATE` и `DELETE` с самой таблицей. Вставка в представлении может пригодиться в том случае, если в нем выводится только один столбец, а остальные столбцы заполняются `NULL` или значениями по умолчанию. В таких случаях `INSERT` может иметь смысл. Также можно включить в представление условие `WHERE`, которое ограничивает вставляемые данные, имитируя ограничение `CHECK` в MySQL.

А если вам еще недостаточно сложностей, то учтите, что обновление возможно только для представлений, не содержащих агрегатных операторов (`SUM`, `COUNT`, `AVG` и т. д.), а также таких операторов, как `BETWEEN`, `HAVING`, `IN` и `NOT IN`.

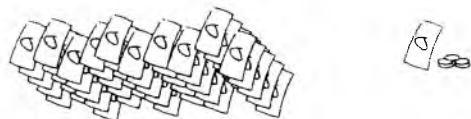


Когда хорошая база данных плохо ведет себя

Миссис Хамфрис хочет перевести 1000 долларов со своего текущего счета на сберегательный счет. Она направляется к банкомату...

Она проверяет баланс своих счетов, текущего и сберегательного.

1000 SAMOLEANS IN CHECKING 30 SAMOLEANS IN SAVINGS



Она выбирает.

TRANSFER 1000 SAMOLEANS FROM CHECKING TO SAVINGS

Она нажимает кнопку.

CHECKING → SAVINGS



Банкомат жалобно пищит, и экран гаснет.

Сбой питания.

Питание снова включилось.

Она проверяет состояние своих счетов.

0 SAMOLEANS IN CHECKING 30 SAMOLEANS IN SAVINGS



Куда же испарились деньги миссис Хамфрис?



Удаление представлений

Когда какое-либо представление становится ненужным, удалите его командой `DROP VIEW`. Это делается очень просто:

```
DROP VIEW pb_dimes;
```

Часто Задаваемые Вопросы

В: Можно ли получить список созданных представлений?

О: Представления хранятся в базе данных как таблицы. Команда `SHOW TABLES` выводит информацию обо всех представлениях и таблицах. А для просмотра структуры представления, как и для просмотра структуры таблицы, используется команда `DESC`.

В: Что происходит при удалении таблицы, использованной в представлении?

О: Возможны разные варианты. Одни РСУБД разрешают использовать представление, не возвращая данных. MySQL позволяет удалить представление только в том случае, если таблица, на которой оно основано, существует, хотя при этом вы можете удалить таблицу, задействованную в представлении. В других РСУБД реализовано другое поведение. Поэкспериментируйте со своей РСУБД и посмотрите, что получится. В общем случае представления лучше удалять до удаления таблиц, на которой они основаны.

В: Ограничения `CHECK` и представления очень полезны в тех случаях, когда с базой данных работает сразу несколько людей. Но что произойдет, если два человека попытаются одновременно изменить один столбец?

О: Чтобы ответить на этот вопрос, необходимо познакомиться с транзакциями. Но сначала мы узнаем историю миссис Хамфрис, которой понадобилось получить деньги в банкомате.

Ограничения `CHECK` и представления помогают сохранить контроль над базой данных при одновременной работе нескольких пользователей.

Что произошло с банкоматом



Здесь произошел
сбой питания.

БАНКОМАТ: 1Я-1Я-1Я...

БАНКОМАТ: ДА ЭТО ЖЕ МИССИС ЭТЕЛЬ ХАМФРИСИ ПРИВЕТ. МИС-
СИС ХАМФРИС {ACCOUNT_ID = 38221}

Миссис Хамфрис: Скажите, сколько у меня денег.

БАНКОМАТ: *Думает* (SELECT BALANCE FROM CHECKING WHERE
ACCOUNT_ID = 38221;
SELECT BALANCE FROM SAVINGS WHERE ACCOUNT_ID = 38221:)
1000 НА ТЕКУЩЕМ СЧЕТУ. 30 НА СБЕРЕГАТЕЛЬНОМ

Миссис Хамфрис: Переведите 1000 долларов с текущего счета
на сберегательный.

БАНКОМАТ: ЗАДАЧА НЕПРОСТАЯ. МИССИС ХАМФРИС. НО Я
СПРАВЛЮСЬ...
{CHECKING_BAL > 1000. ДЕНЕГ ДОСТАТОЧНО}
{СНЯТЬ 1000 С ТЕКУЩЕГО СЧЕТА}
{ПОЖАЛУЙСТА. ВСТАВЬТЕ БИИИИП...}

БАНКОМАТ:

БАНКОМАТ:

БАНКОМАТ: #####

БАНКОМАТ: ДА ЭТО ЖЕ МИССИС ЭТЕЛЬ ХАМФРИСИ ПРИВЕТ. МИС-
СИС ХАМФРИС {ACCOUNT_ID = 38221}

Миссис Хамфрис: Скажите, сколько у меня денег.

БАНКОМАТ: *Думает* (SELECT BALANCE FROM CHECKING WHERE
ACCOUNT_ID = 38221;
SELECT BALANCE FROM SAVINGS WHERE ACCOUNT_ID = 38221:)
0 НА ТЕКУЩЕМ СЧЕТУ. 30 НА СБЕРЕГАТЕЛЬНОМ

БАНКОМАТ:
АИ! РАЗВЕ МОЖНО БИТЬ КУЛАКОМ ПО ЭКРАНУ? ДО СВИДАНИЯ. МИС-
СИС ЭТЕЛЬ ХАМФРИС.



Можно ли сделать так, чтобы банкомат
не забыл о команде INSERT, выполняемой
в начале операции миссис Хамфрис?

А тем временем в другом месте...

Новые неприятности с банкоматами

Джон и Мэри имеют общий счет в банке. В пятницу они одновременно подходят к двум разным банкоматам и каждый пытается снять 300 долларов.



База данных с информацией о состоянии общего счета Мэри и Джона.



БАНКОМАТ: ПРИВЕТ, ДЖОН. С НОВА ТЫР ДУМАЕШЬ, У МЕНЯ ДЕНЬГИ ЛИШНИЕ?

Джон: Сколько денег на моем счету?

БАНКОМАТ: Думает (SELECT CHECKING_BAL FROM ACCOUNTS:) 350 ДОЛЛАРОВ

Джон: Дай мне 300 долларов.

БАНКОМАТ: THAT'S ALL YOU THINK I'M GOOD FOR. TO GIVE ME MONEY. JUST USE ME AND THEN IGNORE ME. ТОЛЬКО И ЗНАЕШЬ, ЧТО ДЕНЬГИ ИЗ МЕНЯ ТАЩИТЬ. ПРИШЕЛ, ВЗЯЛ - И ПРОПАЛ.

[CHECKING_BAL > 300. ДЕНЕГ ДОСТАТОЧНО]

[СНЯТЬ 300 СО СЧЕТА]

[SUBTRACT 300 FROM CHECKING_BAL] [УМЕНЬШИТЬ CHECKING_BAL НА 300]

Джон забирает деньги и уходит.

БАНКОМАТ: ПОКА, ДЖОН. ЗВОНИ.

БАНКОМАТ: ПРИВЕТ, МЭРИ

Мэри: Сколько денег на моем счету?

БАНКОМАТ: Думает (SELECT CHECKING_BAL FROM ACCOUNTS:) 350 ДОЛЛАРОВ

ДЗЫНЬ ДЗЫНЬ

Мэри роется в сумочке в поисках сотового телефона.

Мэри: Дай мне 300 долларов.

БАНКОМАТ: ЗАПРОСТО.

[CHECKING_BAL > 300. SHE HAS ENOUGH MONEY] [CHECKING_BAL > 300. ДЕНЕГ ДОСТАТОЧНО]

[СНЯТЬ 300 СО СЧЕТА]

[УМЕНЬШИТЬ CHECKING_BAL НА 300]

БАНКОМАТ: У ВАС БОЛЬШОЙ ПЕРЕРАСХОД.

350 долларов
350 долларов

50 долларов

-250 долларов

↑
Проблема
возникла
здесь.

А как было бы замечательно, если бы серию команд SQL можно было выполнить как единое целое, все сразу? И если что-то пойдет не так, то вся серия отменяется, словно ее и не было? Но конечно, это только мечты...



Это не мечты, а транзакции

Транзакция представляет собой набор команд SQL, выполняемых как единое целое. В случае миссис Хамфрис транзакция состоит из всех команд SQL, необходимых для перемещения денег с текущего счета на сберегательный.

Три операции образуют одну единицу работы. Это и есть транзакция.

Если баланс текущего счета ≥ 1000
Уменьшить баланс текущего счета на 1000
Увеличить баланс сберегательного счета на 1000

Джон и Мэри пытаются одновременно выполнить *одну и ту же транзакцию*.

Джон и Мэри одновременно пытаются снять по 300 долларов.

Если баланс текущего счета ≥ 300
Уменьшить баланс текущего счета на 300
Выдать 300 долларов

Если баланс текущего счета ≥ 300
Уменьшить баланс текущего счета на 300
Выдать 300 долларов

Транзакция Джона.

Транзакция Мэри.



Банкомат Мэри вообще не должен обращаться к счету (даже для проверки баланса), пока банкомат Джона не завершит обработку транзакции и не снимет блокировку с нее.

Если в ходе транзакции не удастся выполнить хотя бы одну операцию, то не выполняется ни одна операция.

Свойства транзакций

Чтобы набор команд SQL мог считаться транзакцией, он должен обладать четырьмя свойствами: атомарностью, целостностью, изолированностью и устойчивостью. В английском языке этот набор свойств часто обозначается сокращением ACID (Atomicity, Consistency, Isolation, Durability).



Атомарность

Завершаются либо все операции, входящие в транзакцию, либо не завершается ни одна. Транзакция не может быть выполнена частично. Деньги миссис Хамфрис пропали неизвестно куда именно потому, что была выполнена только часть транзакции.



Целостность

Завершенная транзакция оставляет базу данных в логически целостном состоянии. В конце обеих транзакций из наших примеров состояние счетов сбалансировано. В первом примере деньги переведены на накопительный счет, во втором – выданы наличными, но в обоих случаях ничего не пропало.



Изолированность

Каждая транзакция работает со своим целостным представлением базы данных независимо от других транзакций, выполняемых одновременно с ней. Именно это свойство было нарушено в случае Джона и Мэри: банкомат Мэри имел доступ к балансу, пока банкомат Джона завершал транзакцию. Вместо этого банкомат Мэри должен был вывести сообщение «пожалуйста, подождите – выполняется транзакция» или что-нибудь в этом роде.



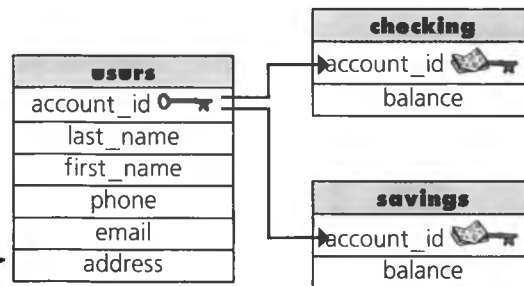
Устойчивость

После завершения транзакции база данных должна сохранить измененные данные, защитив их от сбоев питания или других опасностей. Обычно для этого описание транзакции сохраняется вне основной базы данных. Если бы запись о транзакции миссис Хамфрис хранилась отдельно, то ее 1000 долларов вернулись бы на счет.

SQL помогает работать с транзакциями

Рассмотрим очень простую базу данных для банка. Она состоит из таблицы с данными владельцев счетов, таблицы текущих счетов и таблицы сберегательных счетов.

И еще много других столбцов.



При работе с транзакциями в SQL используются три команды.

START TRANSACTION;

Команда сообщает SQL о начале транзакции.

START TRANSACTION отслеживает выполнение всех последующих команд SQL вплоть до выполнения COMMIT или ROLLBACK.

Здесь РСУБД начинает следить за выполнением вашего кода.



COMMIT;

Команда закрепляет результаты всего выполненного кода.

Если все команды выполнены успешно и все выглядит хорошо, **закрепите изменения** командой **COMMIT**.

Если результат выполнения кода вас устраивает, вы закрепляете изменения в базе данных командой **COMMIT**...



ROLLBACK;

Команда возвращает базу данных к состоянию до начала транзакции.

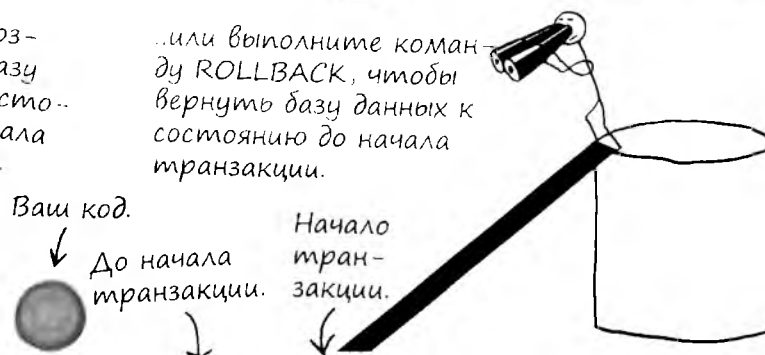
Если что-то пошло не так, команда **ROLLBACK** отменяет все изменения, и база данных возвращается к состоянию до ввода команды **START TRANSACTION**.

...или выполните команду **ROLLBACK**, чтобы вернуть базу данных к состоянию до начала транзакции.

Ваш код.

До начала транзакции.

Начало транзакции.



Изменения вносятся в базу данных только при выполнении команды COMMIT.

Как должен был работать банкомат



БАНКОМАТ: 1Я-1Я-1Я...

БАНКОМАТ: ДА ЭТО ЖЕ МИССИС ЭТЕЛЬ ХАМФРИСИ ПРИВЕТ. МИССИС ХАМФРИС {ACCOUNT_ID = 38221}

Миссис Хамфрис: Скажите, сколько у меня денег.

БАНКОМАТ: *Думает* (SELECT BALANCE FROM CHECKING WHERE ACCOUNT_ID = 38221;
SELECT BALANCE FROM SAVINGS WHERE ACCOUNT_ID = 38221;) 1000 НА ТЕКУЩЕМ СЧЕТУ. 30 НА СБЕРЕГАТЕЛЬНОМ

Миссис Хамфрис: Переведите 1000 долларов с текущего счета на сберегательный.

БАНКОМАТ: ЗАДАЧА НЕПРОСТАЯ. МИССИС ХАМФРИС. НО Я СПРАВЛЮСЬ...
[START TRANSACTION;

[SELECT BALANCE FROM CHECKING WHERE ACCOUNT_ID=38221]

БАНКОМАТ: 1000 НА ТЕКУЩЕМ СЧЕТУ. ПОЖНО ПРОДОЛЖАТЬ.

БАНКОМАТ: [UPDATE CHECKING SET BALANCE = BALANCE - 1000 WHERE ACCOUNT_ID = 38221]

[ПОЖАЛУЙСТА. ВСТАВЬТЕ БИИИИП...]

Здесь произошел сбой питания.

ВКЛЮЧАЕТСЯ РЕЗЕРВНОЕ ПИТАНИЕ: ОТМЕНА ТРАНЗАКЦИИ

БАНКОМАТ:

БАНКОМАТ:

БАНКОМАТ: #####

БАНКОМАТ: ДА ЭТО ЖЕ МИССИС ЭТЕЛЬ ХАМФРИСИ ПРИВЕТ. МИССИС ХАМФРИС {ACCOUNT_ID = 38221}

Миссис Хамфрис: Скажите, сколько у меня денег.

АТМ: *Думает* (SELECT BALANCE FROM CHECKING WHERE ACCOUNT_ID = 38221;
SELECT BALANCE FROM SAVINGS WHERE ACCOUNT_ID = 38221;) 1000 НА ТЕКУЩЕМ СЧЕТУ. 30 НА СБЕРЕГАТЕЛЬНОМ

Команда COMMIT не была выполнена, поэтому состояние базы данных не изменилось.

Как работать с транзакциями в MySQL

Чтобы использовать транзакции в MySQL, необходимо выбрать правильное ядро хранения данных. Так называется механизм, обеспечивающий хранение всей информации и структур базы данных. Одни типы ядер поддерживают транзакции, другие их не поддерживают.

Вспомните, как в главе 4 мы рассматривали результат выполнения команды

```
SHOW CREATE TABLE my_contacts;
```

А теперь выбор ядра хранения стал для нас существенным.

Не обращайте внимания на текст после закрывающей круглой скобки. Он описывает механизм хранения данных и используемую кодировку символов. Пока нас устроят значения по умолчанию.

Если скопировать и выполнить этот код, он создаст таблицу.

Команда для экономии времени

Взгляните на код, который использовался для создания таблицы на с. 217, и приведенный ниже результат выполнения команды SHOW CREATE TABLE my_contacts. Эти фрагменты не идентичны, но если вставить этот код в команду CREATE TABLE, результат будет тем же. Удалить обратные апострофы или параметры данных не нужно, но если вы это сделаете, команда получится более компактной.

Имена символьных и числовых значений в обратные апострофы. Эти символы присутствуют в результатах команды SHOW CREATE TABLE.

```
CREATE TABLE `my_contacts`
(
  `last_name` varchar(30) default NULL,
  `first_name` varchar(20) default NULL,
  `email` varchar(50) default NULL,
  `gender` char(1) default NULL,
  `birthday` date default NULL,
  `profession` varchar(50) default NULL,
  `location` varchar(50) default NULL,
  `status` varchar(20) default NULL,
  `interests` varchar(100) default NULL,
  `seeking` varchar(100) default NULL,
) ENGINE=MyISAM DEFAULT CHARSET=cp1251
```

SQL считает, что столбцы по умолчанию инициализируются значением NULL (если явно не задано другое значение).

При создании таблицы желательно указывать, может ли столбец содержать NULL.

Проследите за тем, чтобы было выбрано ядро BDB или InnoDB — только эти два ядра поддерживают транзакции.



РАССЛАБЬТЕСЬ

InnoDB и BDB — два разных механизма, используемых РСУБД для хранения информации.

Они называются ядрами хранения данных, и выбор любого из них позволит вам использовать транзакции. За дополнительной информацией о различиях между ядрами хранения данных MySQL обращайтесь к документации.

Для наших целей неважно, какое из двух ядер вы выберете. Для смены ядра используется команда следующего вида:

```
ALTER TABLE имя_таблицы TYPE = InnoDB;
```

Теперь попробуйте сами

Предположим, мы решили превратить все 1-центовые монетки в копилке (P) в 25-центовые (Q).

Попробуйте выполнить следующий код для таблицы `piggy_bank`, созданной ранее в этой главе. В первой транзакции будет использована команда `ROLLBACK`, отменяющая все изменения.

```
START TRANSACTION;
SELECT * FROM piggy_bank;
UPDATE piggy_bank set coin = 'Q' where coin= 'P';
SELECT * FROM piggy_bank; ← Здесь изменения еще видны...
ROLLBACK; ← Передумали.
SELECT * FROM piggy_bank; ← ...а здесь их уже нет.
```

Второй раз используем команду `COMMIT`, потому что изменения нас устраивают.

```
START TRANSACTION;
SELECT * FROM piggy_bank;
UPDATE piggy_bank set coin = 'Q' where coin= 'P';
SELECT * FROM piggy_bank; ← Здесь изменения еще видны...
COMMIT; ← Закрепление транзакции.
SELECT * FROM piggy_bank; ← ...и здесь тоже.
```

Возьми в руку карандаш



Запишите содержимое базы данных piggy_bank после выполнения транзакций. Сейчас база данных содержит следующие данные.

piggy_bank

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'Q' where coin = 'P'
AND coin_year < 1970;
COMMIT;
```

id	coin	coin_year
1		
2		
3		
4		

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'N' where coin = 'Q';
ROLLBACK;
```

id	coin	coin_year
1		
2		
3		
4		

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'Q' where coin = 'N'
AND coin_year > 1950;
ROLLBACK;
```

id	coin	coin_year
1		
2		
3		
4		

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'D' where coin = 'Q'
AND coin_year > 1980;
COMMIT;
```

id	coin	coin_year
1		
2		
3		
4		

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'P' where coin = 'N'
AND coin_year > 1970;
COMMIT;
```

id	coin	coin_year
1		
2		
3		
4		

→ Ответ на с. 518.

Часто
**Задаваемые
 Вопросы**

В: Необходима ли команда **START TRANSACTION**, или **COMMIT** и **ROLLBACK** будут работать без нее?

О: Вы должны сообщить своей РСУБД о начале транзакции командой **START TRANSACTION**. Это необходимо для того, чтобы РСУБД знала, до какой точки следует отменить выполненные операции.

В: Можно ли использовать **START TRANSACTION** просто для того, чтобы проверить работу некоторых запросов?

О: Можно и нужно. Это отличный способ поэкспериментировать с запросами, изменяющими данные таблиц, без

вреда для самих таблиц в случае ошибки. Только не забудьте выполнить **COMMIT** или **ROLLBACK** после завершения экспериментов.

В: Нельзя ли обойтись без **COMMIT** и **ROLLBACK**?

О: Ваша РСУБД регистрирует все операции, выполняемые внутри транзакций, в журнале транзакций; чем больше вы выполняете операций, тем больше места занимает журнал. Постарайтесь применять транзакции только тогда, когда вам действительно необходимо иметь возможность отмены выполненных действий, чтобы не расходовать напрасно дисковое пространство, а вашей РСУБД не приходилось выполнять лишнюю работу по отслеживанию выполняемых действий.



Мне все еще нужно придумать, как полностью **закрыть** доступ к некоторым таблицам. Скажем, мой новый бухгалтер должен работать только с таблицами зарплат, и только. А еще нужно сделать так, чтобы некоторые пользователи могли использовать **SELECT**, но операции **INSERT**, **UPDATE** и **DELETE** были им **ЗАПРЕЩЕНЫ**.

Как организовать контроль за доступом пользователей к базе данных?

Об этом вы узнаете в следующей главе.



Новые инструменты

Глава 11 осталась позади, а ваш инструментарий практически полон. В этой главе вы узнали, как создавать представления для своих данных и как выполнять транзакции. Полный список инструментов приведен в приложении III.

Транзакция

Группа команд, выполняемых как единое целое. Если выполнение хотя бы одной команды будет прервано, то отменяются сразу все команды.

START TRANSACTION

Команда, сообщающая РСУБД о начале транзакции. Все дальнейшие изменения существуют лишь временно, пока не будет выполнена команда **COMMIT**. Транзакция продолжает выполняться, пока она либо не будет закреплена командой **COMMIT**, либо отменена командой **ROLLBACK**. В случае отмены база данных возвращается в состояние, в котором она находилась до выполнения **START TRANSACTION**.

Представление

Результат запроса, рассматриваемый как таблица. Представления особенно удобны для сокращения сложности запросов.

Обновляемое представление

Представление, которое позволяет изменять данные в базовых таблицах. Обновляемые представления должны содержать все столбцы **NOT NULL** своих базовых таблиц.

Необновляемое представление

Представление, которое не может использоваться для вставки или обновления данных базовой таблицы.

Ограничения проверки

Ограничения, разрешающие вставку или обновление в таблице только конкретных значений.

CHECK OPTION

Ключевые слова, используемые при создании обновляемого представления; дальнейшие операции вставки и обновления проверяются на соответствие условию **WHERE** представления.

Возьми в руку карандаш



Решение
Со с. 497.

Фрэнк выполняет запрос `SELECT` на с. 496 с использованием нового представления `job_raises`. Как ему упорядочить результаты в алфавитном порядке по фамилиям?

Нужно добавить `ORDER BY last_name` либо в команду создания представления, либо в команду `SELECT` при его использовании.



Возьми в руку карандаш Решение

Со с. 514.

Запишите содержимое базы данных piggy_bank после выполнения транзакций. Сейчас база данных содержит следующие данные.

piggy_bank

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

START TRANSACTION;

UPDATE piggy_bank set coin = 'Q' where coin = 'P'
AND coin_year < 1970;

COMMIT;

← Совпадений нет — а значит, нет и изменений.

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

START TRANSACTION;

UPDATE piggy_bank set coin = 'N' where coin = 'Q';

ROLLBACK;

← Отмена, изменений нет.

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

START TRANSACTION;

UPDATE piggy_bank set coin = 'Q' where coin = 'N'
AND coin_year > 1950;

ROLLBACK;

← Отмена, изменений нет.

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

START TRANSACTION;

UPDATE piggy_bank set coin = 'D' where coin = 'Q'
AND coin_year > 1980;

COMMIT;

← Измененная запись.

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	D	1999

START TRANSACTION;

UPDATE piggy_bank set coin = 'P' where coin = 'N'
AND coin_year > 1970;

COMMIT;

← Измененная запись.

id	coin	coin_year
1	Q	1950
2	P	1972
3	P	2005
4	Q	1999

Защита данных



Вы потратили массу времени и сил на создание базы данных. И если теперь с ней что-нибудь случится, это будет полной катастрофой. Кроме того, вам приходится предоставлять другим пользователям **доступ к данным** и вы опасаетесь, что они могут ошибиться с вставкой или обновлением — или и того хуже, **удалить нужные данные**. В этой главе вы узнаете, как **защитить** базу данных и хранящиеся в ней объекты и как установить контроль над тем, **какие операции с данными разрешены тем или иным пользователям**.

Проблемы с пользователями

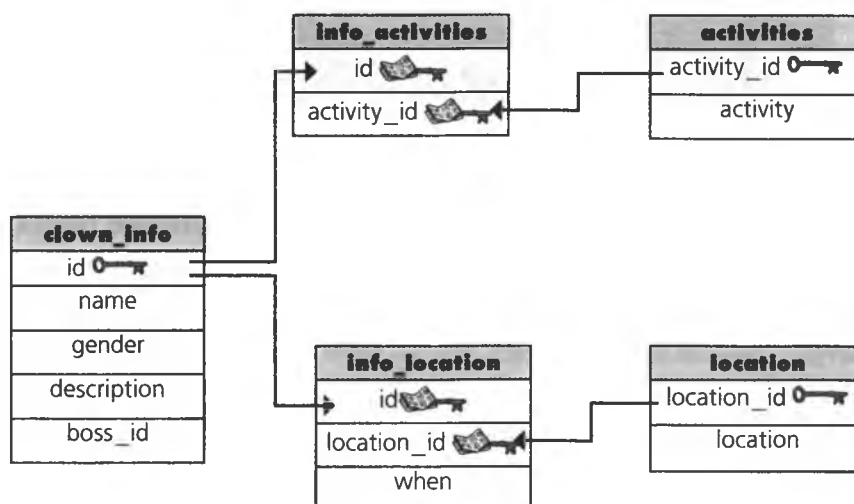
Сбор данных о клоунах в Дейтавиле развернулся настолько широко, что городской совет нанял целую группу работников для наблюдения за клоунами и занесения информации в базу данных `clown_tracking`.

К сожалению, в группу проник переодетый клоун, известный под кличкой «Джордж». Он создал целый ряд проблем в базе данных, включая потерю и нежелательное изменение данных, а также почти совпадающие записи, существующие только из-за преднамеренных опечаток. Вот лишь некоторые из возникших проблем:



В таблице `clown_info` имеются записи клоунов с именами Снаглз, Снаглз и Снуглз. Мы уверены, что это один и тот же клоун, потому что столбцы `gender` и `description` во всех случаях совпадают (различаются только имена).

Лишние записи в таблице `clown_info` создают массу проблем с просмотром информации. Так, в таблице `info_location` используются идентификаторы разных записей Снаглз, Снаглз и Снуглз из таблицы `clown_info`.



Предотвращение ошибок в базе данных

Джордж уволился до того, как его саботаж был замечен, и нам пришлось исправлять причиненный ущерб. Теперь, когда на работу приходят новые люди, им необходимо предоставить возможность выборки, чтобы они могли опознать клоунов. Но вставка или обновление должны быть запрещены — как и другие операции, пока мы не присмотримся к ним повнимательнее.

Также при исправлении ошибок необходимо действовать осторожно — если мы разрешим новому работнику удаление данных, он может случайно удалить хорошие данные вместе с плохими.

Настало время защитить базу данных, пока другие клоуны не уничтожили ее полностью.

Возьми в руку карандаш



Защитите базу данных от возможного саботажа со стороны клоунов. С каждой стороны запишите некоторые запросы, которые должны быть разрешены (или запрещены) новым работникам. Там, где это возможно, укажите имена таблиц.

Операции, которые следует разрешить новым работникам.

пример: SELECT from activities

Операции, которые следует **запретить** новым работникам.

пример: DROP TABLE on clown_info

Возьми в руку карандаш Решение



Защитите базу данных от возможного саботажа со стороны клоунов. С каждой стороны запишите некоторые запросы, которые должны быть разрешены (или запрещены) новым работникам. Там, где это возможно, укажите имена таблиц.

Операции, которые следует разрешить новым работникам.

пример: `SELECT from activities`

`SELECT from clown_info,
info_activities, activities,
info_location, location`

Операции, которые следует **запретить** новым работникам.

пример: `DROP TABLE on clown_info`

`DROP TABLE on clown_info, info_
activities, activities, info_location, location`

`INSERT on clown_info, info_activities,
activities, info_location, location`

`UPDATE on clown_info, info_activities,
activities, info_location, location`

`ALTER on clown_info, info_activities,
activities, info_location, location`

`DELETE on clown_info, info_activities,
activities, info_location, location`

Хорошие новости — мы можем помешать клоунам уничтожать наши данные!

SQL позволяет управлять тем, какие операции разрешены или запрещены тому или иному пользователю базы данных. Но сначала необходимо создать для таких пользователей — и вообще всех, кто будет работать с нашей базой данных — **учетные записи**.



Защита учетной записи root

До настоящего момента у нашей базы данных был только один пользователь, а доступ к ней не был защищен паролем. Каждый, кто мог получить доступ к базе через консоль или графический интерфейс, мог делать с данными все, что угодно.

По умолчанию первый пользователь — `root` — может выполнять с базой данных любые операции. Это важно, потому что пользователь `root` должен иметь возможность создавать учетные записи других пользователей. Ограничивать права `root` не нужно, но учетной записи `root` необходимо назначить пароль. В MySQL это делается следующей командой:

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('b4dc10wnZ');
```

Имя привилегированного пользователя — 'root'.

«localhost» указывает, где установлено и выполняется программное обеспечение SQL.

Пароль, назначенный пользователю root.

В других РСУБД используются другие команды. Например, в Oracle команда выглядит так:

```
alter user root identified by новый-пароль;
```

Если вы работаете с базой данных через графический интерфейс, вероятно, для смены пароля будет намного удобнее использовать диалоговые окна. *Здесь важно не то, как это делается, а то, что это обязательно нужно сделать.*

За информацией о защите учетной записи `root` обращайтесь к документации своей РСУБД.

Часть Задаваемые Вопросы

В: А что означает это «localhost»? Можете объяснить подробнее?

О: localhost означает, что для выполнения запросов используется тот же компьютер, на котором установлена РСУБД. Значение localhost используется по умолчанию, поэтому включать его в команду не обязательно.

В: А если я работаю с SQL-клиентом на другом компьютере?

О: Это называется удаленным доступом. Вы должны указать в запросе местонахождение этого компьютера; в этом случае localhost заменяется IP-адресом или именем хоста. Например, если РСУБД установлена на компьютере с именем kumquats в сети издательства O'Reilly, то имя будет иметь вид root@kumquats.oreilly.com.

Создание нового пользователя

Вопрос, ответ на который вам, вероятно, очевиден:

Как SQL хранит информацию о пользователях?

В таблице, конечно! РСУБД поддерживает базу данных с информацией о себе. В таблице хранятся идентификаторы пользователей, имена, пароли и сведения об операциях, которые пользователю разрешено выполнять с каждой базой данных.

Создание нового пользователя можно начать с имени и пароля. В SQL команды для создания пользователей не существует, но в большинстве РСУБД используются команды следующего вида:

```
CREATE USER elsie  
IDENTIFIED BY 'cl3v3rp4s5w0rd';
```

Имя пользователя
для нашего нового
работника — Элси.

Пароль

Разве нельзя запретить
Элси доступ к некоторым
таблицам при создании
учетной записи?



**Будьте
осторожны!**

SQL не определяет синтаксис управления пользователями.

Синтаксис команд создания пользователей зависит от конкретной РСУБД. За информацией о том, как следует создавать пользователей в вашей РСУБД, обращайтесь к документации.

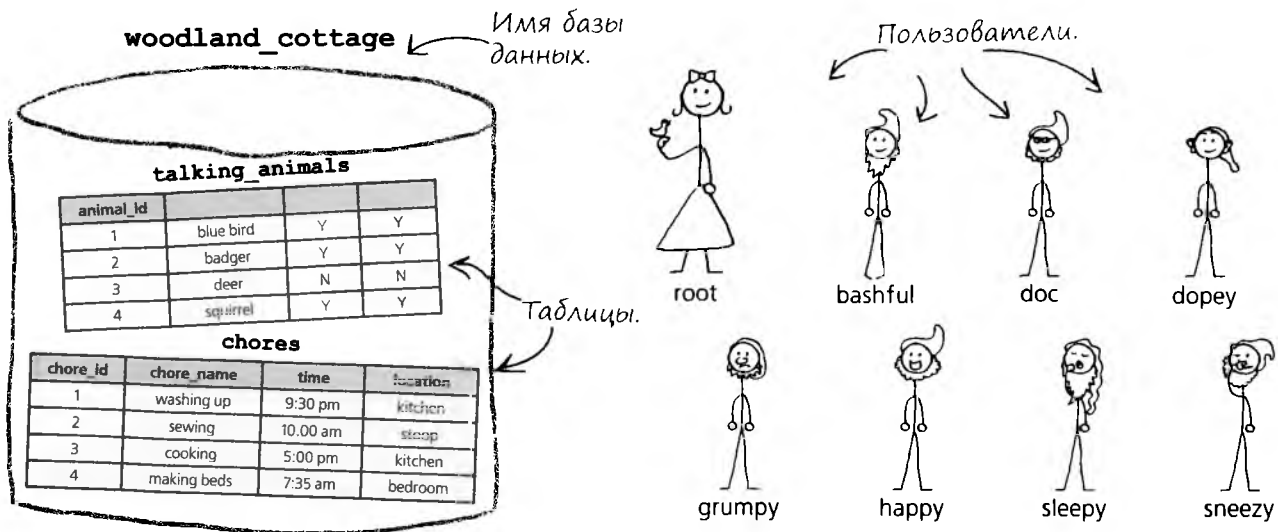
Можно, но иногда мы еще не знаем, какой уровень доступа следует предоставить пользователю.

Однако мы все равно должны решить, **какая информация должна быть доступна для созданного пользователя.** Все по порядку: сначала вы создаете пользователя, а затем предоставляете ему необходимый уровень доступа. Если вы будете уметь предоставлять доступ отдельно от создания пользователя, то сможете внести изменения позднее при изменении базы.

Решите, что необходимо каждому пользователю

Мы создали учетную запись Элси. Пока она не имеет разрешений для выполнения каких-либо операций. Чтобы разрешить ей выполнение любых операций с `slow_info` (даже простой выборки), необходимо использовать команду **GRANT**.

В отличие от учетной записи `root`, которой разрешено выполнение любой команды SQL с любым объектом базы данных, учетным записям новых пользователей запрещены любые действия. Чтобы разрешить пользователям какие-либо операции с базой данных, необходимо выполнить команду **GRANT**.



Изменение таблицы разрешается только некоторым пользователям.

Новые записи в таблице `chores` могут добавляться только администратором базы данных. Только пользователь `root` может выполнять команды `INSERT`, `UPDATE` и `DELETE` для этой таблицы. Пользователь `happy` отвечает за таблицу `talking_animals` и может изменять ее структуру командой `ALTER`, а также выполнять другие операции.

Выборка из таблицы разрешается только некоторым пользователям.

Выборка из таблицы `talking_animals` разрешается всем пользователям, кроме `grumpy`.

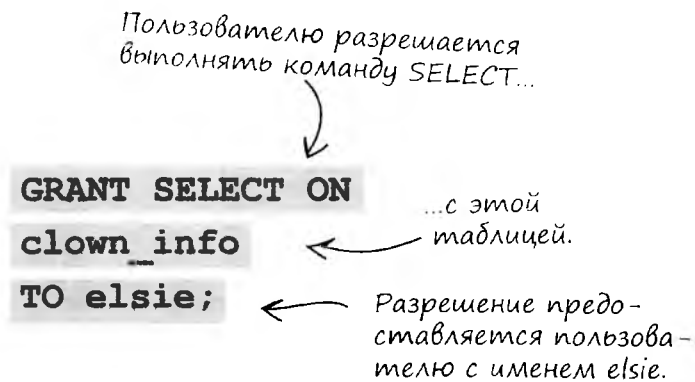
Даже внутри таблиц могут действовать свои ограничения: некоторые пользователи могут видеть лишь часть столбцов.

Все, кроме `dopey`, могут просматривать содержимое столбца `instructions` таблицы `chores`.

Команда GRANT точно определяет, какие операции с таблицами и столбцами могут выполняться пользователями.

Простая команда GRANT

Итак, на данный момент Элси запрещено выполнение каких-либо операций. Она может подключиться к РСУБД со своим именем пользователя и паролем, но этим все и ограничивается. Чтобы Элси могла выполнить команду SELECT с таблицей `clown_info`, ей необходимо предоставить соответствующее **разрешение**. Это делается следующей командой.



Элси также необходимо разрешить выполнение SELECT с другими таблицами, чтобы она могла использовать соединения и подзапросы в своих командах SELECT. Для каждой таблицы разрешение предоставляется отдельной командой GRANT:

```
GRANT SELECT ON activities TO elsie;  
GRANT SELECT ON location TO elsie;  
GRANT SELECT ON info_activities TO elsie;  
GRANT SELECT ON info_location TO elsie;
```



Упражнение

Попробуйте предположить, что делают следующие команды GRANT для базы данных woodland_cottage (с. 525).

	Команда	Что она делает?
1.	GRANT INSERT ON magic_animals TO doc;	
2.	GRANT DELETE ON chores TO happy, sleepy;	
3.	GRANT DELETE ON chores TO happy, sleepy WITH GRANT OPTION;
4.	GRANT SELECT(chore_name) ON chores TO dopey;	<p>Подсказка: это имя столбца.</p> <p>.....</p>
5.	GRANT SELECT, INSERT ON talking_animals TO sneezy;	
6.	GRANT ALL ON talking_animals TO bashful;	

Теперь попробуйте самостоятельно написать несколько команд GRANT.

7.	Разрешить пользователю doc выборку из таблицы chores.
8.	Разрешить пользователю sleepy удаление из таблицы talking_animals, а также разрешить sleepy предоставлять разрешение на удаление из talking_animals любому другому пользователю.
9.	Разрешить всем (ALL) пользователям выполнение любых операций с chores.
10.	Разрешить пользователю doc выборку из всех таблиц базы данных woodland_cottage.



Упражнение

Решение

Попробуйте предположить, что делают следующие команды GRANT для базы данных woodland_cottage (с. 525).

Команда	Что она делает?
1. GRANT INSERT ON magic_animals TO doc;	Разрешает пользователю doc вставку в таблицу magic_animals.
2. GRANT DELETE ON chores TO happy, sleepy;	Разрешает happy и sleepy удаление из таблицы chores.
3. GRANT DELETE ON chores TO happy, sleepy WITH GRANT OPTION;	Разрешает happy и sleepy удаление из таблицы chores, а также предоставление другим того же разрешения.
4. GRANT SELECT(chore_name) ON chores TO dopey;	Разрешает dopey выборку только из столбца chore_name таблицы chores.
5. GRANT SELECT, INSERT ON talking_animals TO sneezy;	Разрешает sneezy выборку и вставку в таблицу talking_animals.
6. GRANT ALL ON talking_animals TO bashful;	Разрешает bashful выборку, обновление, вставку и удаление с таблицей talking_animals.

Теперь попробуйте самостоятельно написать несколько команд GRANT.

7. GRANT SELECT ON chores TO doc;	Разрешить пользователю doc выборку из таблицы chores.
8. GRANT DELETE ON talking_animals TO sleepy WITH GRANT OPTION;	Разрешить пользователю sleepy удаление из таблицы talking_animals, а также разрешить sleepy предоставлять разрешение на удаление из talking_animals любому другому пользователю.
9. GRANT ALL ON chores TO bashful, doc, dopey, grumpy, happy, sleepy, sneezy;	Разрешить всем (ALL) пользователям выполнение любых операций с chores.
10. GRANT SELECT ON woodland_cottage.* TO doc	Разрешить пользователю doc выборку из всех таблиц базы данных woodland_cottage.

Разновидности GRANT

В только что выполненном упражнении были представлены все основные разновидности команды GRANT. Вот они:

1

В одной команде GRANT можно перечислить несколько имен пользователей.

Всем перечисленным пользователям предоставляются одинаковые разрешения.

2

Конструкция WITH GRANT OPTION разрешает пользователям предоставлять другим разрешение, указанное в команде.

Описание выглядит крайне запутанно, но по сути означает, что если пользователь получает разрешение выполнять выборку из chores, то с WITH GRANT OPTION он может также разрешать выборку из chores другим пользователям.

3

Вместо всей таблицы можно указать конкретный столбец или набор столбцов.

Пользователю можно разрешить выборку только из одного столбца. В этом случае содержимое всех остальных столбцов останется скрытым от него.

4

Для таблицы можно указать более одного разрешения.

Перечислите предоставляемые разрешения, разделяя их запятыми.

5

GRANT ALL разрешает выполнение операций SELECT, UPDATE, INSERT и DELETE с заданной таблицей.

Фактически это сокращенная запись для перечисления всех операций с таблицей.

6

Конструкция database_name.* обозначает все таблицы в базе данных.

По аналогии с тем, как * обозначает «все записи» в команде SELECT, эта конструкция обозначает все таблицы в базе данных.

Команда REVOKE

А если потребуется лишить Элси предоставленной привилегии SELECT? Для этого используется команда **REVOKE**.

Помните простейшую форму команды GRANT? Синтаксис REVOKE выглядит почти так же, только GRANT заменяется на REVOKE, а TO — на FROM.

Привилегия, которая отнимается у пользователя (SELECT).

```
REVOKE SELECT ON  
clown_info  
FROM elsie;
```

Пользователь, который лишается привилегии.

Также можно отозвать WITH GRANT OPTION, оставив саму привилегию неизменной. В следующем примере *happy* и *sleepy* смогут выполнять команду DELETE с таблицей chores, но не смогут предоставить эту привилегию другим пользователям.

Пользователь лишается только привилегии GRANT OPTION.

```
REVOKE GRANT OPTION ON  
DELETE ON chores  
FROM happy, sleepy;
```

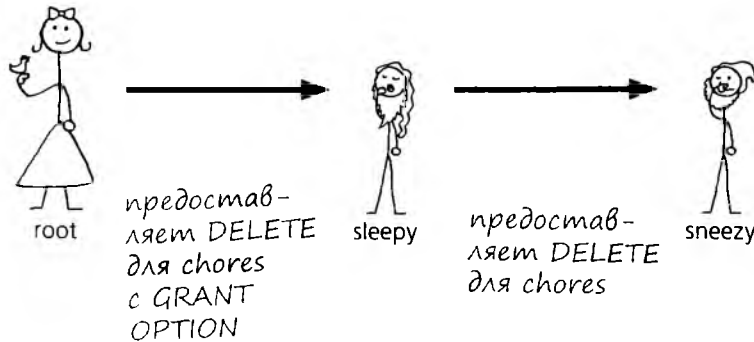
Пользователи happy и sleepy по-прежнему используют DELETE, но не смогут предоставить эту привилегию другим пользователям.

Только попробуй, Джим, и я лишу тебя всех привилегий на целый месяц.

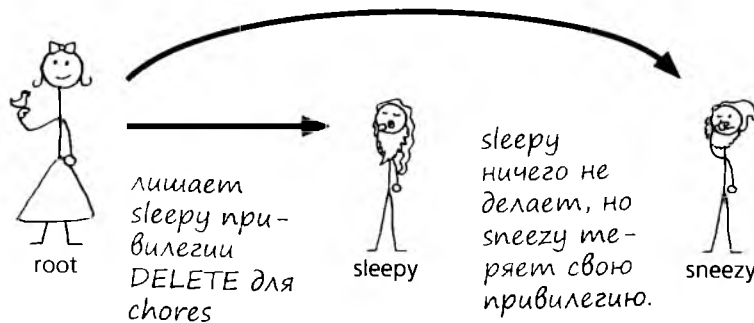


Отзыв использованной привилегии GRANT OPTION

Допустим, пользователь *root* предоставил *sleepy* привилегию DELETE с GRANT OPTION для таблицы *chores*. Пользователь *sleepy* предоставил *sneezy* привилегию DELETE для таблицы *chores*.



Допустим, пользователь *root* меняет свое решение и отнимает привилегию у *sleepy*. Пользователь *sneezy* также лишится этой привилегии, хотя непосредственно отозвана она была только у *sleepy*.



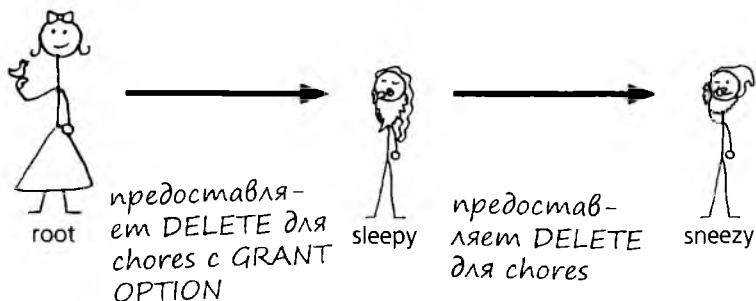
У команды REVOKE существует побочный эффект: *sneezy* также лишается своей привилегии. Чтобы вы могли управлять тем, что происходит при отзыве привилегий, в синтаксисе REVOKE предусмотрены два специальных ключевых слова.



Вскоре вы познакомитесь с ключевыми словами RESTRICT и CASCADE. Как вы думаете, что они делают?

CASCADE и RESTRICT

Существует два способа отозвать привилегию так, чтобы отзыв не отразился на других пользователях. Ключевые слова **CASCADE** и **RESTRICT** позволяют более точно указать, кто должен сохранить свои привилегии или лишиться их.



С ключевым словом **CASCADE** привилегии лишается заданный пользователь (в данном случае *sleepy*), а также все, кому он предоставил соответствующие разрешения.

REVOKE DELETE ON chores FROM sleepy CASCADE;



С ключевым словом **RESTRICT** привилегии лишается заданный пользователь, а если он предоставил привилегию кому-то еще — выдается сообщение об ошибке.

REVOKE DELETE ON chores FROM sleepy RESTRICT;



Оба пользователя сохраняют свои привилегии, а *root* получает сообщение об ошибке. Изменение не сохраняется, потому что оно распространяется на пользователя *sneezy*, не указанного в команде.

Возьми в руку карандаш



Кто-то продолжает предоставлять Эли лишние привилегии. Напишите команды REVOKE для возвращения к безопасному уровню, на котором разрешена только выборка командой SELECT.


```
GRANT SELECT, INSERT, DELETE ON locations TO elsie;
```

```
GRANT ALL ON clown_info TO elsie;
```

```
GRANT SELECT, INSERT ON activities TO elsie;
```

```
GRANT DELETE, SELECT on info_location TO elsie  
WITH GRANT OPTION;
```

```
GRANT INSERT(location), DELETE ON locations TO elsie;
```

 Возьми в руку карандаш
Решение

Кто-то продолжает предоставлять Элси лишние привилегии. Напишите команды REVOKE для возвращения к безопасному уровню, на котором разрешена только выборка командой SELECT.

```
GRANT SELECT, INSERT, DELETE ON locations TO elsie;
```

```
.....  
REVOKE INSERT, UPDATE, DELETE ON locations FROM elsie;
```

```
GRANT ALL ON clown_info TO elsie;
```

```
.....  
REVOKE INSERT, UPDATE, DELETE ON clown_info FROM elsie;
```

```
GRANT SELECT, INSERT ON activities TO elsie;
```

```
.....  
REVOKE INSERT ON activities FROM elsie;
```

```
GRANT DELETE, SELECT on info_location TO elsie  
WITH GRANT OPTION;
```

```
.....  
REVOKE DELETE on info_location FROM elsie CASCADE;
```

```
GRANT INSERT(location), DELETE ON locations TO elsie;
```

```
.....  
REVOKE GRANT INSERT(location), DELETE ON locations FROM elsie;
```

Вероятно, здесь также можно использовать GRANT, чтобы Элси могла выполнять выборку из locations.

И еще стоит убедиться в том, что Элси не предоставила другим те привилегии, которые были предоставлены ей.

Привилегия SELECT должна остаться, поэтому отзывает не все.

Другой способ — сначала отозвать все привилегии, а потом предоставить необходимые.

Часто
Задаваемые
Вопросы

В: Я все еще думаю о командах `GRANT` с именами столбцов. Что произойдет, если разрешить вставку для одного столбца таблицы?

О: Хороший вопрос. Вообще-то такая привилегия будет практически бесполезной: если пользователь может сохранить данные только в одном столбце, он не сможет вставить запись в таблицу. Попытка завершится успешно только в том случае, если таблица состоит из единственного столбца, указанного в команде `GRANT`.

В: Существуют ли другие, столь же бесполезные команды `GRANT`?

О: Почти все привилегии для столбцов бесполезны, если они не сочетаются с привилегией `SELECT` в команде `GRANT`.

В: Предположим, я хочу добавить пользователя, которому разрешена выборка из всех таблиц во всех моих базах данных. Существует ли простой способ сделать это?

О: Как уже не раз было в этой главе, все зависит от вашей разновидности РСУБД. В MySQL глобальные привилегии такого рода предоставляются командой вида:

```
GRANT SELECT ON *.*
TO elsie;
```

Первая звездочка обозначает все базы данных, а вторая — все таблицы.

В: Значит, режим `CASCADE` используется по умолчанию для команды `REVOKE`?

О: Обычно `CASCADE` используется по умолчанию, но за подробностями вам снова следует обращаться к документации вашей РСУБД.

В: Что произойдет при отзыве привилегии, которой пользователь не обладает?

О: Вы получите сообщение об ошибке, в котором говорится об отсутствии `GRANT`.

В: Что произойдет, если два разных пользователя предоставят `sleezy` одинаковые привилегии, которые отзываются `root` в предыдущем примере?

О: Здесь начинаются сложности. Одни системы не обращают внимания на то, какая команда `GRANT` выполнялась в режиме `CASCADE`, другие игнорируют привилегии, предоставленные другими пользователями. Это еще одна ситуация, в которой ответ следует искать в документации РСУБД.

В: Существуют ли другие объекты, кроме таблиц и столбцов, которые могут использоваться в `GRANT` и `REVOKE`?

О: Представления могут использоваться точно так же, как таблицы — если только представление не является необновляемым. В противном случае вам не удастся использовать `INSERT` даже при наличии разрешения. И по аналогии с таблицами, доступ может разрешаться на уровне конкретных столбцов представления.

Значит, если я хочу предоставить одинаковые разрешения пяти пользователям, мне нужно перечислить их через запятую в конце команды `GRANT`?



Такое решение определенно сработает. И при небольшом количестве пользователей действовать нужно именно так.

Но по мере роста вашей организации в ней будут появляться классы пользователей. Допустим, 10 человек занимаются вводом данных и им достаточно предоставить разрешения на вставку и выборку из некоторых таблиц. Также в системе могут быть три администратора, которым разрешены любые операции, и множество рядовых пользователей с доступом на уровне `SELECT`. Возможно даже существование программ и веб-приложений, которые подключаются к базе данных и работают с конкретными представлениями.

Так почему бы не создать по одной учетной записи для каждого класса, чтобы несколько пользователей использовали одно и то же имя с паролем?



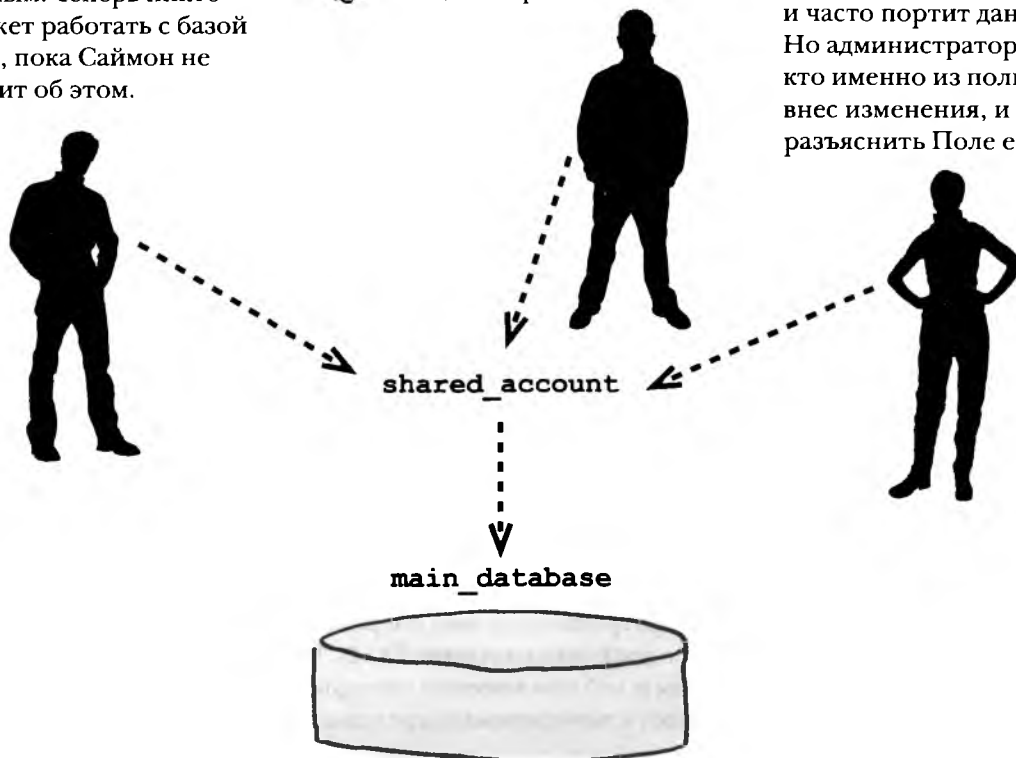
Проблема общих учетных записей

Некоторые компании неплохо управляются с общими учетными записями, которые могут работать с базой данных, но это довольно рискованный путь. Несколько примеров ситуаций, в которых могут возникнуть проблемы:

Саймон изменяет пароль и забывает сказать об этом остальным. Теперь никто не сможет работать с базой данных, пока Саймон не вспомнит об этом.

Рэнди по своей работе должен иметь полный доступ ко всем объектам базы данных. Из-за этого база данных становится уязвимой для других пользователей, которые не так хорошо знают SQL и чаще совершают ошибки.

Пола плохо знает, как пишутся команды обновления, и часто портит данные в базе. Но администратор не знает, кто именно из пользователей внес изменения, и не сможет разъяснить Поле ее ошибки.



Отдельные учетные записи — не лучшее решение для групп пользователей, совместное использование одной учетной записи тоже не годится... Что же делать?



Нужно каким-то образом назначить группам их привилегии, одновременно сохраняя за каждым пользователем индивидуальную учетную запись.

Для этого нам понадобятся **роли**. Роль позволяет определить набор привилегий и назначить их всем участникам некоторой группы. Роли становятся объектами базы данных, которые можно изменять в случае изменения базы — при этом вам не придется явно изменять привилегии каждого пользователя в соответствии с изменениями в базе.

Команда создания роли очень проста:

```
CREATE ROLE data_entry;
```

↑
Имя создаваемой роли.



Будьте осторожны!

В MySQL роли не поддерживаются.

Вероятно, поддержка ролей появится в будущих версиях MySQL, но пока привилегии приходится назначать на уровне отдельных пользователей.

Привилегии предоставляются ролям точно так же, как и именам пользователей.

```
GRANT SELECT, INSERT ON some_table TO data_entry;
```

↑
Вместо имени пользователя при назначении привилегий указывается имя роли.

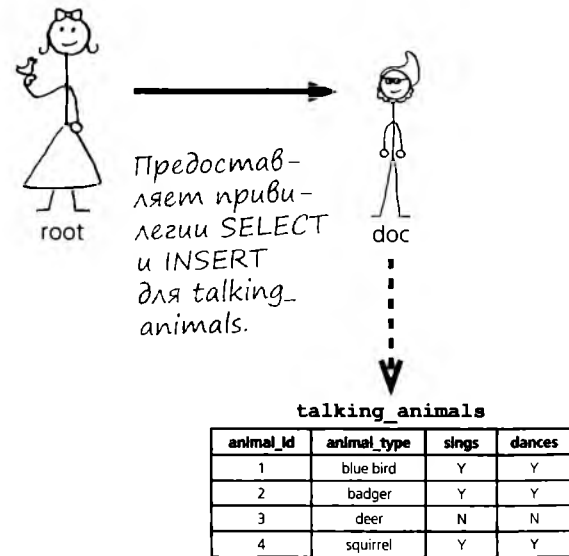
Мы создали роль и определили ее привилегии. Теперь необходимо назначить ее пользователю...

Использование роли

До создания ролей привилегии пользователей назначались командой GRANT.

```
GRANT SELECT, INSERT
ON talking_animals
TO doc;
```

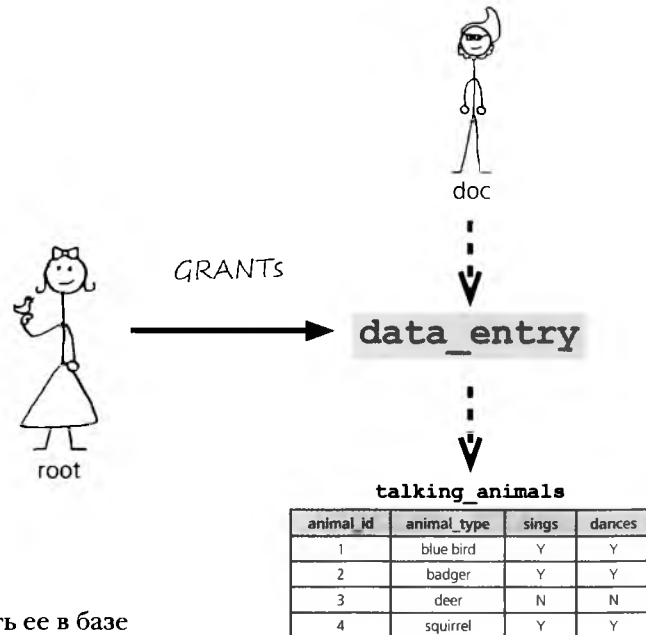
↖ ↗
Старый способ.



Остается лишь заменить операцию GRANT, указать в ней новую роль и применить к doc. Перечислять привилегии или таблицы не нужно, потому что вся информация хранится в роли data_entry:.

```
GRANT data_entry TO doc;
```

↖
Имя роли занимает место имени таблицы и привилегий.



Удаление ролей

Когда надобность в роли отпадает, незачем хранить ее в базе данных. Удаление ролей осуществляется командой DROP.

```
DROP ROLE data_entry;
```

Часть Задаваемые Вопросы

В: А если я хочу предоставить привилегии для всех таблиц в базе данных? Перечислять все имена?

О: Нет, используйте следующий синтаксис:

```
GRANT SELECT, INSERT, DELETE
ON gregs_list.*
TO jim;
```

Достаточно указать имя базы данных, а знак * обеспечит назначение привилегий для всех таблиц этой базы.

В: Можно ли удалить роль, которая в данный момент назначена пользователю?

О: Вы можете удалять роли, используемые в настоящее время. Будьте очень внимательны при удалении ролей, чтобы не лишить пользователей необходимых разрешений.

В: Если пользователь обладает ролью, которая была удалена, он лишается этих разрешений?

О: Вот именно! Все выглядит так, словно вы явно предоставили ему эти разрешения, а потом отозвали их. Только вместо одного пользователя разрешений лишаются все пользователи, которым была назначена эта роль.

В: Может ли пользователь иметь сразу несколько ролей?

О: Да, может. Только проследите за тем, чтобы роли не конфликтовали друг с другом, иначе вы создадите себе немало проблем.

Возьми в руку карандаш



Как отозвать роль

Роли отнимаются у пользователей почти так же, как и привилегии. Удастся ли вам написать команду для отзыва роли `data_entry` у пользователя `doc`, не заглядывая на следующую страницу?



Возьми в руку карандаш
Решение

Роли отнимаются у пользователей почти так же, как и привилегии. Удастся ли вам написать команду для отзыва роли `data_entry` у пользователя `doc`, не заглядывая на следующую страницу?

```
REVOKE data_entry FROM doc;
```

Конструкция WITH ADMIN OPTION

Как говорилось ранее, у команды `GRANT` имеется режим `WITH GRANT OPTION`. У ролей существует аналогичная конструкция `WITH ADMIN OPTION`. Она позволяет каждому обладателю роли назначать ее всем остальным. Например, при использовании следующей команды:

```
GRANT data_entry TO doc WITH ADMIN OPTION;
```

Теперь `doc` обладает привилегиями администратора и может назначить `happy` роль `data_entry` точно так же, как она была назначена ему самому:

```
GRANT data_entry TO happy;
```

При использовании с ролями команда `REVOKE` также поддерживает ключевые слова `CASCADE` и `RESTRICT`. Давайте посмотрим, как они работают.

Отзыв роли с CASCADE

С ключевым словом `CASCADE` команда `REVOKE` действует не только на пользователя, но и на всех остальных пользователей в цепочке.

```
REVOKE data_entry FROM doc CASCADE;
```



Отзыв роли с RESTRICT

Если пользователь успел назначить роль кому-то другому, то при попытке лишить его этой роли командой REVOKE с ключевым словом RESTRICT произойдет ошибка.

REVOKE data_entry FROM doc RESTRICT;



пытается
лишить doc
роли data_
entry...



...но ничего
не выходит,
потому что
изменение
также затро-
нет happy.



Если лишение роли
затронет других
пользователей,
команда REVOKE
с RESTRICT выдает
ошибку.

Оба пользователя сохраняют свои роли, а *root* получает сообщение об ошибке. Изменение не сохраняется, потому что оно распространяется на пользователя *happy*.

Роли — удобная штука, но давайте вернемся к реальности? У меня всего два работника, скоро появится третий. Роли мне не нужны, но я не хочу, чтобы мой персонал использовал учетную запись *root*. Как мне назначить своим работникам правильные уровни доступа без использования ролей?



Пришло время заняться настройкой безопасности доступа к базе данных *gregs_list*.

Грег должен выполнить действия, описанные в этой главе, и защитить учетную запись *root*, разобраться в том, какой уровень доступа нужен его работникам, и предоставить им соответствующие привилегии.

А теперь плохая новость: на месте Грега окажется ВВ!

Стань Грегом



В последний раз представьте себя на месте Грега и настройте разрешения пользователей, чтобы неопытные работники ничего случайно не испортили.

Прочитайте описания работы каждого пользователя. Напишите команды GRANT, которые предоставят работникам доступ к необходимым данным и не позволят им сделать ничего лишнего.



Фрэнк: Я отвечаю за поиск работников для открывшихся вакансий. Я никогда не ввожу данные в базу, хотя и удаляю вакансии при их закрытии или обнаружении претендента. Иногда мне также приходится искать контактные данные в таблице `my_contacts`.

Джим: Я ввожу всю новую информацию во всей базе данных. Теперь, когда я не могу случайно ввести X в столбце `gender`, у меня все отлично получается. Я также обновляю данные и понемногу учусь удалять, хотя Грег мне это запрещает. Конечно, ему я об этом не говорю...

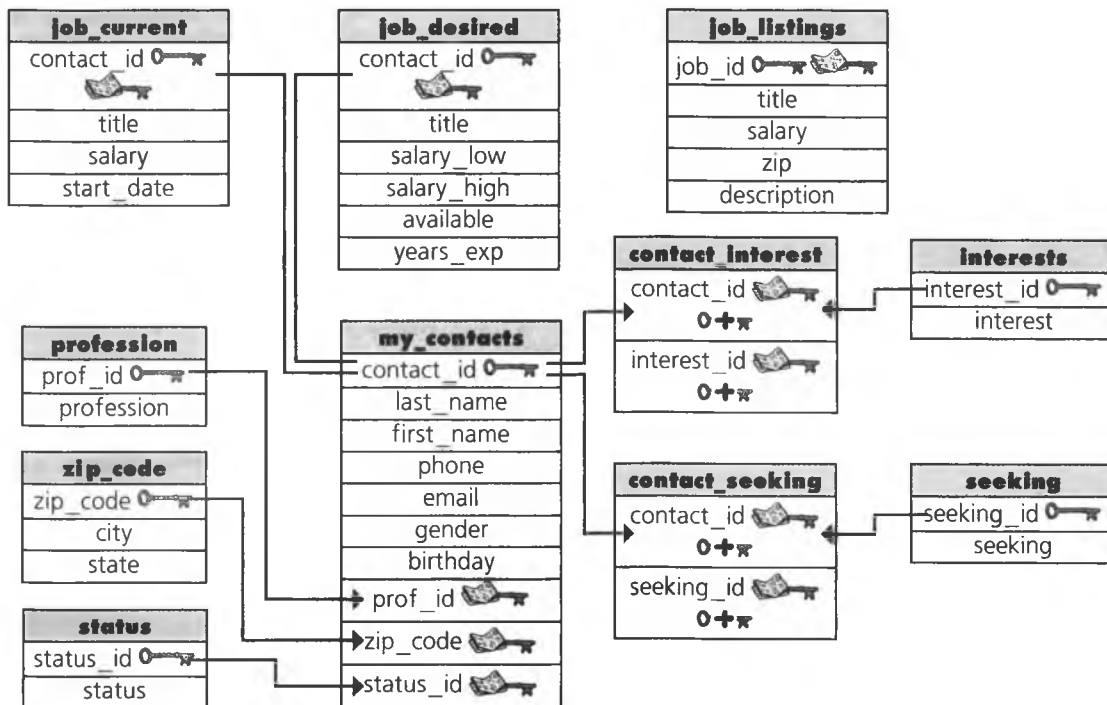
Джо: Грег только что нанял меня для работы над сайтом своей фирмы. Он хочет, чтобы контактные данные интегрировались на сайте. Вообще-то я веб-программист, а не специалист по SQL, но я умею выполнять простые команды SELECT. А всякие вставки-удаления — это не для меня.

Взгляните на структуру базы данных `gregs_list` и напишите команды GRANT для этих парней, пока они не повредили данные.

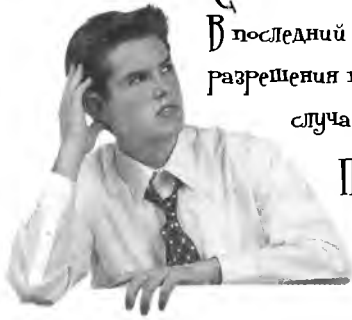
Напишите команду для назначения пароля пользователю с именем «root».

Напишите три команды, которые создают три учетные записи — по одной для каждого из трех работников.

Напишите команды GRANT, которые будут назначать правильные разрешения каждому из трех работников.



Стань Грегом. Ответ



В последний раз представьте себя на месте Грега и настройте разрешения пользователей, чтобы неопытные работники ничего случайно не испортили.

Прочитайте описание работы каждого пользователя. Напишите команды GRANT, которые предоставят работникам доступ к необходимым данным и не позволят им сделать ничего лишнего.

Напишите команду для назначения пароля пользователю с именем «root».

```
SET PASSWORD FOR root@localhost = PASSWORD('gr3GRu1z');
```

Напишите три команды, которые создают три учетные записи — по одной для каждого из трех работников.

```
CREATE USER frank IDENTIFIED BY 'jobM4tch';
```

```
CREATE USER jim IDENTIFIED BY 'NOm0r3Xs';
```

```
CREATE USER joe IDENTIFIED BY 's3LeCTd00d';
```

Если ваши пароли выглядят иначе — не обращайтесь в службу поддержки. Главное, чтобы все фрагменты команды были на месте и следовали в правильном порядке!

Напишите команды GRANT, которые будут назначать правильные разрешения каждому из трех работников.

```
GRANT DELETE ON job_listings TO frank;
```

```
GRANT SELECT ON my_contacts * TO frank;
```

Фрэнк должен иметь возможность удалять вакансии и выполнять выборку из my_contacts.

```
GRANT SELECT, INSERT ON gregs_list * TO jim;
```

Джиму необходим доступ SELECT и INSERT ко всем объектам gregs_list. А вот удалять данные ему пока рановато.

```
GRANT SELECT ON my_contacts, profession, zip_code, status,  
contact_interest, interests, contact_seeking, seeking TO joe;
```

Наконец, Джо нужно разрешить выборку из всех исходных таблиц — кроме тех, которые относятся к поискам работы.

Объединение CREATE USER с GRANT



Пока мы еще не закончили — нельзя ли объединить CREATE USER с GRANT в одну команду?

Да, можно. Для этого достаточно связать воедино две части, которые мы уже видели.

Вот как выглядели команды CREATE USER и GRANT, использованные нами для Элси:

```
CREATE USER elsie
IDENTIFIED BY 'cl3v3rp4s5w0rd';

GRANT SELECT ON
clown_info
TO elsie;
```

Давайте объединим их, опустив часть CREATE USER. Так как пользователь *elsie* должен существовать для предоставления ему привилегий, РСУБД сначала проверяет, существует ли пользователь с заданным именем, и если нет — автоматически создает учетную запись.

```
GRANT SELECT ON
clown_info
TO elsie
IDENTIFIED BY 'cl3v3rp4s5w0rd';
```

список грега выходит на мировой уровень

Оглушительный успех!

Благодаря вашей помощи Грег так хорошо освоил SQL (а также научил Джима, Фрэнка и Джо), что в его базу данных была включена поддержка рекламных объявлений и форумов.

А знаете, что самое невероятное? В Дейтавиле его ждал такой успех, что в более 500 городах по всему миру открылись филиалы фирмы Грега, а сам Грег не сходит с первых страниц газет!



Спасибо, без вас я бы не справился! Кстати, у меня открывается филиал в вашем городе... Может, это стоит обсудить?

THE WEEKLY INQUERUER

Триумф списка Грега

Вакансии и форумы

Друзья и родственники говорят, что слава ничуть не изменила Грега.

Трой Армстронг
КОРРЕСПОНДЕНТ INQUERUER

ДЕЙТАВИЛЬ — Местный предприниматель Грег добился успеха и известности. Его сетевая база данных превратилась из набора карточек сначала в простую таблицу, а потом и в многотабличную базу данных со службой знакомств, поиском работы и другими полезными функциями.



База данных Грега уже добралась до вашего города? Если нет — это вопрос времени, говорят аналитики.



Новые инструменты

Поздравляем, вы закончили главу 12!
Ниже перечислены основные концепции безопасности SQL, представленные в этой главе. Полный список инструментов приведен в приложении III.

CREATE USER

Команда используется некоторыми СУБД для создания учетных записей пользователей с назначением пароля.

GRANT

Команда точно определяет, какие операции могут выполняться пользователем с таблицами и столбцами на основании предоставленных ему привилегий.

WITH GRANT OPTION

Разрешает пользователю давать другим пользователям предоставленные ему привилегии.

REVOKE

Команда лишает пользователя привилегий.

WITH ADMIN OPTION

Разрешает пользователю назначать другим пользователям собственную ему роль.

Роль

Роль представляет собой набор привилегий. Роли используются для назначения групп привилегий нескольким пользователям.

спасибо, что посетили Дейтавилль!

Присоединяйтесь!



**Используйте SQL в своих проектах...
и возможно, вас тоже ожидает успех!**

Мы были рады встретиться с вами в Дейтавилле.

Жаль, конечно, что нам приходится расставаться, но пришло время **применить полученные знания на практике** — наверняка где-нибудь вблизи от вас есть клоуны, за которыми нужно наблюдать, пончики, которые нужно попробовать, или Список [подставьте ваше имя], который нужно создать. В конце книги вы найдете еще кое-какую полезную информацию и алфавитный список основных инструментов SQL — а потом **беритесь за дело!**

Приложение I: Прочее

Десять важнейших тем (о которых мы не рассказали)



Но даже после всего сказанного беседа еще не закончена! Есть еще кое-что, о чем вы должны знать. Мы решили, что будет неправильно просто проигнорировать эти темы — они заслуживают хотя бы краткого упоминания. Итак, прежде чем откладывать книгу, ознакомьтесь с этими **короткими, но важными разделами**. А когда вы прочтаете и эту главу, останется еще пара приложений... и может быть, немного рекламы... и ничего больше. Честное слово!

1. Используйте графический интерфейс к своей РСУБД

Безусловно, очень важно уметь напрямую выполнять команды SQL из консоли, но к этому моменту вы уже достаточно хорошо понимаете, как это делается. И конечно, вам хотелось бы иметь более простой способ создания таблиц и просмотра их содержимого.

У каждой РСУБД имеется свой графический интерфейс. Далее приводится краткая сводка графических инструментов MySQL.

Графические инструменты MySQL

Вместе с MySQL также можно загрузить графический инструментарий MySQL, и что еще важнее – программу MySQL Administrator. Весь пакет доступен по адресу:

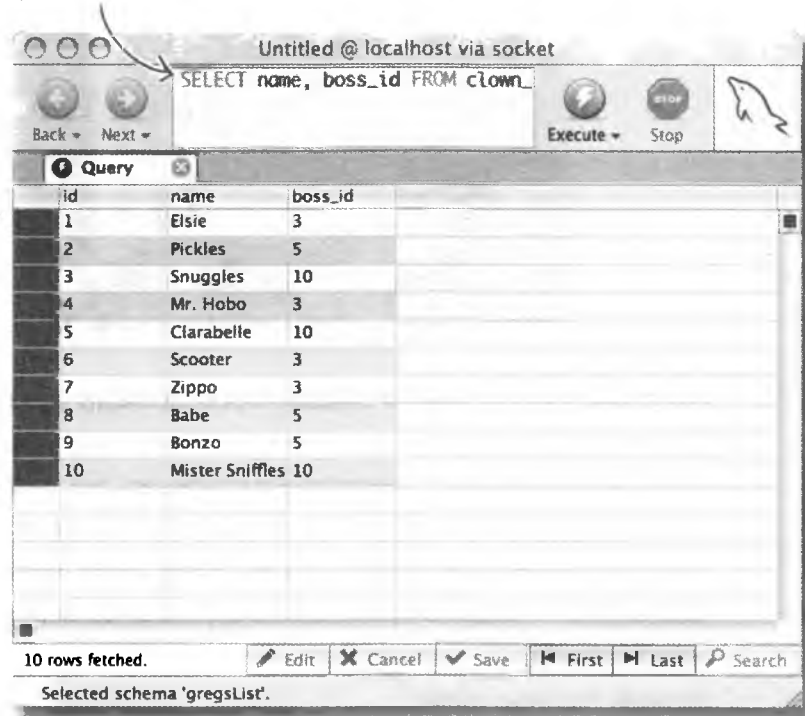
<http://dev.mysql.com/downloads/gui-tools/5.0.html>

Имеются версии для Windows, Mac и Linux. Программа MySQL Administrator обеспечивает возможность простого и удобного просмотра, создания и редактирования баз данных и таблиц.

Также вам может пригодиться программа MySQL Query Browser. Она позволяет ввести запрос и просмотреть результаты в интерфейсе программы (вместо консоли).

Здесь вводятся запросы.

Здесь выводятся результаты.

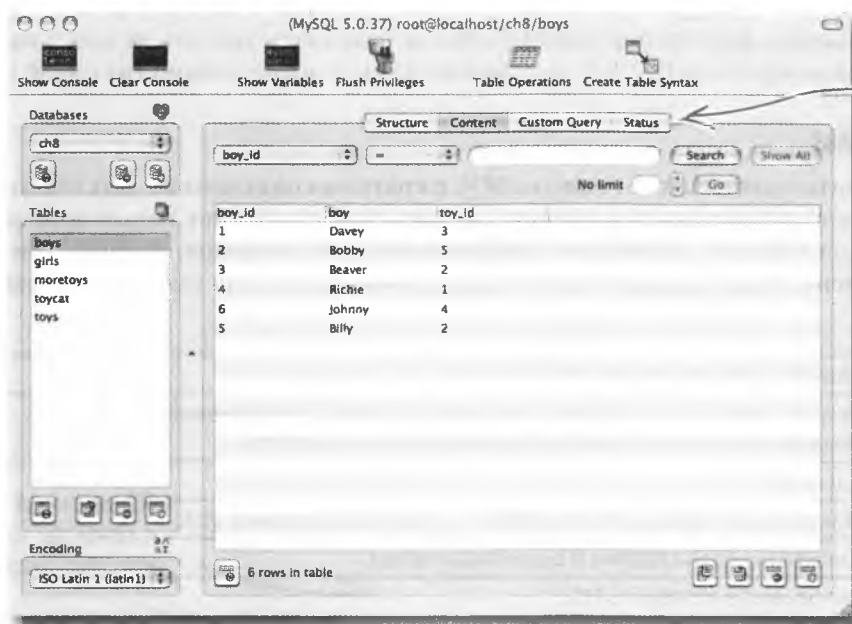


Другие графические инструменты

Также существует немало других графических программ для работы с MySQL. Выберите из них ту, которая лучше всего подходит лично вам. Ниже приведены лишь некоторые примеры, а остальное можно легко найти поиском в Интернете.

Пользователям Mac стоит обратить внимание на CocoaMySQL:

<http://cocoamysql.sourceforge.net/>



Кнопки для просмотра структуры запроса, его выполнения и изменения таблицы.

Если вам требуется сетевое решение, попробуйте использовать phpMyAdmin. Эта программа хорошо подходит для удаленного администрирования MySQL по Интернету. Для работы с MySQL на локальном компьютере она менее удобна. Дополнительную информацию можно найти на странице:

<http://www.phpmyadmin.net/>

Далее перечислены другие популярные графические инструменты. Некоторые из них работают исключительно на платформе PC; чтобы узнать, подойдут ли они вам, лучше всего зайти на сайт и прочитать информацию о последней версии.

Navicat предоставляет 30-дневный бесплатный пробный период:

<http://www.navicat.com/>

SQLyog предлагает бесплатную версию Community Edition:

<http://www.webyog.com/en/>

2. Зарезервированные слова и специальные символы

В языке SQL довольно много зарезервированных ключевых слов. Эти слова не рекомендуется использовать в качестве имен баз данных, таблиц и столбцов. Даже если вам очень хочется назвать новую таблицу «select», попробуйте найти более содержательное имя, в котором «select» не встречается. А если без зарезервированного ключевого слова никак не обойтись, постарайтесь использовать его с другими словами и символами подчеркивания, чтобы не путать РСУБД. На следующей странице приведен список ключевых слов, которые не рекомендуется использовать в именах.

Ситуация усложняется тем, что в SQL имеются незарезервированные слова, которые могут оказаться зарезервированными в будущих версиях SQL. Мы не будем перечислять их здесь; найдите их в справочнике по вашей конкретной РСУБД, который вы купите после знакомства с этой книгой.

Специальные символы

Ниже перечислены многие специальные символы SQL с краткими описаниями. Как и ключевые слова, их не рекомендуется использовать в именах — исключение составляет только символ подчеркивания (_). В общем случае имя не должно содержать никаких символов, кроме букв и символов подчеркивания. Имена из одних цифр тоже нежелательны, хотя и они иногда оказываются содержательными.

*	Возвращает все столбцы таблицы в команде SELECT.	
()	Используются для группировки выражений, определения порядка выполнения математических операций и вызова функций, а также для ограничения подзапросов.	
;	Завершает команды SQL.	
,	Разделяет элементы списков (например, в команде INSERT или условии IN).	<i>Являются специальными символами только с LIKE.</i>
.	Используется в ссылках на имена таблиц и в дробных числах.	
_	Представляет один символ в условии LIKE.	
%	Представляет несколько символов в условии LIKE.	
!	Обозначает отрицание (NOT); используется при сравнении в условиях WHERE.	
'	Строковые значения в SQL заключаются между двумя апострофами.	
"	Также для этой цели можно использовать кавычки, хотя лучше ограничиться апострофами.	
\	Используется для включения апострофов в текстовые столбцы таблиц.	
+	Кроме операции сложения, также используется для конкатенации (сцепления) строк.	

Краткая сводка математических операторов.

+	Сложение	-	Вычитание	*	Между двумя значениями обозначает операцию умножения.	/	Деление
---	----------	---	-----------	---	---	---	---------

И сравнительных операторов.

>	Больше	!>	Не больше	>=	Больше либо равно
<	Меньше	!<	Не меньше	<=	Меньше либо равно
=	Равно	<>	Не равно	!=	Не равно

В книге не рассматриваются. За информацией обращайтесь к документации своей РСУБД.

Зарезервированные слова

Этот список рекомендуется просматривать каждый раз, когда в программе какому-либо объекту присваивается имя из одного слова — убедитесь в том, что это слово не зарезервировано.

A	ABSOLUTE ACTION ADD ADMIN AFTER AGGREGATE ALIAS ALL ALLOCATE ALTER AND ANY ARE ARRAY AS ASC ASSERTION AT AUTHORIZATION
B	BEFORE BEGIN BINARY BIT BLOB BOOLEAN BOTH BREADTH BY
C	CALL CASCADE CASCADED CASE CAST CATALOG CHAR CHARACTER CHECK CLASS CLOB CLOSE COLLATE COLLATION COLUMN COMMIT COMPLETION CONNECT CONNECTION CONSTRAINT CONSTRAINTS CONSTRUCTOR CONTINUE CORRESPONDING CREATE CROSS CUBE CURRENT CURRENT DATE CURRENT PATH CURRENT ROLE CURRENT TIME CURRENT TIMESTAMP CURRENT USER CURSOR CYCLE
D	DATA DATE DAY DEALLOCATE DEC DECIMAL DECLARE DEFAULT DEFERRABLE DEFERRED DELETE DEPTH Deref DESC DESCRIBE DESCRIPTOR DESTROY DESTRUCTOR DETERMINISTIC DICTIONARY DIAGNOSTICS DISCONNECT DISTINCT DOMAIN DOUBLE DROP DYNAMIC
E	EACH ELSE END END EXEC EQUALS ESCAPE EVERY EXCEPT EXCEPTION EXEC EXECUTE EXTERNAL
F	FALSE FETCH FIRST FLOAT FOR FOREIGN FOUND FROM FREE FULL FUNCTION
G	GENERAL GET GLOBAL GO GOTO GRANT GROUP GROUPING
H	HAVING HOST HOUR
I	IDENTITY IGNORE IMMEDIATE IN INDICATOR INITIALIZE INITIALLY INNER INOUT INPUT INSERT INT INTEGER INTERSECT INTERVAL INTO IS ISOLATION ITERATE
J	JOIN
K	KEY
L	LANGUAGE LARGE LAST LATERAL LEADING LEFT LESS LEVEL LIKE LIMIT LOCAL LOCALTIME LOCALTIMESTAMP LOCATOR
M	MAP MATCH MINUTE MODIFIES MODIFY MODULE MONTH
N	NAMES NATIONAL NATURAL NCHAR NCLOB NEW NEXT NO NONE NOT NULL NUMERIC
O	OBJECT OF OFF OLD ON ONLY OPEN OPERATION OPTION OR ORDER ORDINALITY OUT OUTER OUTPUT
P	PAD PARAMETER PARAMETERS PARTIAL PATH POSTFIX PRECISION PREFIX PREORDER PREPARE PRESERVE PRIMARY PRIOR PRIVILEGES PROCEDURE PUBLIC
Q	
R	READ READS REAL RECURSIVE REF REFERENCES REFERENCING RELATIVE RESTRICT RESULT RETURN RETURNS REVOKE RIGHT ROLE ROLLBACK ROLLUP ROUTINE ROW ROWS
S	SAVEPOINT SCHEMA SCROLL SCOPE SEARCH SECOND SECTION SELECT SEQUENCE SESSION SESSION_USER SET SETS SIZE SMALLINT SOME SPACE SPECIFIC SPECIFICTYPE SQL SQLEXCEPTION SQLSTATE SQLWARNING START STATE STATEMENT STATIC STRUCTURE SYSTEM USER
T	TABLE TEMPORARY TERMINATE THAN THEN TIME TIMESTAMP TIMEZONE_HOUR TIMEZONE_MINUTE TO TRAILING TRANSACTION TRANSLATION TREAT TRIGGER TRUE
U	UNDER UNION UNIQUE UNKNOWN UNNEST UPDATE USAGE USER USING
V	VALUE VALUES VARCHAR VARIABLE VARYING VIEW
W	WHEN WHENEVER WHERE WITH WITHOUT WORK WRITE
X	
Y	YEAR
Z	ZONE

3. ALL, ANY и SOME

Три ключевых слова ALL, ANY и SOME очень полезны при работе с подзапросами. Они работают с операторами сравнения и наборами результатов. Прежде чем переходить к их рассмотрению, вспомните оператор IN из главы 9.

```
SELECT name, rating FROM restaurant_ratings
WHERE rating IN
(SELECT rating FROM restaurant_ratings
WHERE rating > 3 AND rating < 9);
```

Подзапрос возвращает все оценки в диапазоне от 3 до 9 — в данном случае 7 и 5.

name	rating
Pizza House	3
The Shack	7
Arthur's	9
Ribs 'n' More	5

Этот запрос возвращает все названия ресторанов с такими же оценками, как у результата подзапроса в круглых скобках. Результат состоит из двух записей, *The Shack* и *Ribs 'n' More*.

Ключевое слово ALL

А теперь рассмотрим следующий запрос:

```
SELECT name, rating FROM restaurant_ratings
WHERE rating > ALL
(SELECT rating FROM restaurant_ratings
WHERE rating > 3 AND rating < 9);
```

На этот раз мы хотим получить все рестораны с оценкой, превышающей *все* оценки в наборе (это ресторан *Arthur's*).

Пример запроса с оператором <:

```
SELECT name, rating FROM restaurant_ratings
WHERE rating < ALL
(SELECT rating FROM restaurant_ratings WHERE
rating > 3 AND rating < 9);
```

С ALL также можно использовать операторы >= и <=. Следующий запрос вернет записи *The Shack* и *Arthur's*. В результат включаются записи с оценкой, большей либо равной наивысшей оценке в наборе, которая равна 7:

```
SELECT name, rating FROM restaurant_ratings
WHERE rating >= ALL
(SELECT rating FROM restaurant_ratings
WHERE rating > 3 AND rating < 9);
```

В результат включаются записи с оценкой, большей либо равной наивысшей оценке в наборе.

> ALL находит все значения, большие максимального значения в наборе.

< ALL находит все значения, меньшие минимального значения в наборе.

Ключевое слово ANY

Условие с ANY истинно, если хотя бы одна запись в наборе удовлетворяет условию. Рассмотрим следующий пример.

```
SELECT name, rating FROM restaurant_ratings
WHERE rating > ANY
(SELECT rating FROM restaurant_ratings WHERE
rating > 3 AND rating < 9);
```

Он означает следующее: «Выбрать все записи, у которых оценка больше, чем хотя бы одно из значений набора (5, 7)». Так как оценка *The Shack* равна 7, что больше 5, запись включается в результат. Также возвращается запись *Arthur's* с оценкой 9.

Ключевое слово SOME

SOME означает то же, что ANY в стандартном синтаксисе SQL. Чтобы узнать, работает ли это ключевое слово в вашей РСУБД, обращайтесь к документации.

> ANY находит все значения, большие минимального значения в наборе.

< ANY находит все значения, меньшие максимального значения в наборе.

4. Подробнее о типах данных

Основные типы данных вам уже известны, но некоторые подробности помогут еще точнее подобрать нужный тип для ваших столбцов. Давайте рассмотрим несколько новых типов, а также поближе познакомимся с теми типами, которые уже использовались ранее.

BOOLEAN

В столбце этого типа может храниться значение «true», «false», или же он может остаться равным NULL. Тип очень удобен для хранения логических признаков «истина/ложь». Во внутреннем представлении РСУБД сохраняет 1 для истинных значений или 0 для ложных. При вставке также можно указывать 1 вместо «true» или 0 вместо «false».

INT

Тип INT неоднократно использовался в книге. В столбцах этого типа могут храниться значения в диапазоне от 0 до 4 294 967 295 в том случае, если столбец должен принимать только положительные значения.

Если значения столбца могут быть как положительными, так и отрицательными, следует определить столбец типа INT *со знаком*. Значения таких столбцов лежат в диапазоне от -2 147 483 648 до 2 147 483 647. Чтобы сообщить РСУБД, что в столбце должны храниться числа INT со знаком, используйте следующий синтаксис при создании столбца:

INT (SIGNED)

Другие разновидности INT

Тип INT вам уже известен, но два типа SMALLINT и BIGINT помогают немного оптимизировать хранение информации. Они определяют наибольшее число, которое может храниться в столбце.

Диапазоны значений этих типов зависят от конкретной СУБД. В MySQL они выглядят так:

	со знаком	без знака
SMALLINT	от -32768 до 32767	от 0 до 65535
BIGINT	от -9223372036854775808 до 9223372036854775807	от 0 до 18446744073709551615

Кроме того, в MySQL дополнительно определены следующие целочисленные типы:

	со знаком	без знака
TINYINT	от -128 до 127	от 0 до 255
MEDIUMINT	от -8388608 до 8388607	от 0 до 16777215

Типы DATE и TIME

Ниже приведена сводка стандартных форматов типов даты и времени в MySQL.

DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYYMMDDHHMMSS
TIME	HH:MM:SS

some_dates

a_date
2007-08-25 22:10:00
1925-01-01 02:05:00

При выборке из столбца, содержащего дату и/или время, можно изменить формат возвращаемых данных. Функции форматирования даты/времени зависят от конкретной РСУБД. Рассмотрим пример использования функции MySQL `DATE_FORMAT()`.

Допустим, у вас есть столбец `a_date`:

Форматные строки должны быть заключены в апострофы.

```
SELECT DATE_FORMAT(a_date, '%M %Y') FROM some_dates;
```

Обозначения `%M` и `%Y` описывают нужный формат даты. Результат должен выглядеть примерно так:

a_date
August 2007
January 1925

Количество параметров форматирования огромно; рассмотреть их здесь нам не удастся. Просто запомните, что параметры форматирования позволяют получить значения столбцов даты/времени в нужном формате, одновременно отсекая все лишнее.

5. Временные таблицы

В этой книге мы создали много разных таблиц. Каждый раз при создании таблицы РСУБД сохраняет описание ее структуры. При вставке данных РСУБД сохраняет эти данные. Если завершить сеанс SQL в окне терминала или в графической программе, таблица и хранящиеся в ней данные никуда не пропадут. И таблицы, и данные остаются в базе до тех пор, пока вы не удалите их.

SQL также поддерживает другую разновидность таблиц — так называемые *временные таблицы*. Временная таблица существует от ее создания до момента удаления или до завершения пользовательского сеанса. Под **сеансом** понимается промежуток времени от подключения к РСУБД с данными учетной записи до выхода или завершения графической программы. Временные таблицы также могут удаляться явно командой DROP.

Для чего может понадобиться временная таблица:

- Во временной таблице можно сохранить промежуточные результаты — например, результат выполнения некоторой математической операции со столбцом, который вам понадобится только во время текущего сеанса.
- Вам потребуется сохранить содержимое таблицы на заданный момент времени.
- Помните, как мы преобразовали базу данных Грега из одной таблицы в группу связанных таблиц? Временные таблицы упрощают реструктуризацию данных, и вы можете быть уверены в том, что они исчезнут после завершения сеанса.
- При использовании SQL в приложениях, написанных на языке программирования, вы можете создать временные таблицы на время сбора данных, а затем сохранить результаты в долгосрочной таблице.

Создание временной таблицы

Синтаксис создания временных таблиц в MySQL прост — достаточно добавить ключевое слово TEMPORARY:

```
CREATE TEMPORARY TABLE my_temp_table
(
  some_id INT,
  some_data VARCHAR(50)
)
```

← TEMPORARY — единственное изменение в команде.

Сокращенный синтаксис

Временные таблицы также можно создавать по результатам запросов:

```
CREATE TEMPORARY TABLE my_temp_table AS
SELECT * FROM my_permanent_table;
```

← После AS может следовать произвольный запрос.



**Будьте
осторожны!**

**Синтаксис
создания
временных
таблиц
зависит
от РСУБД.**

За информацией об этой
возможности обращайтесь
к документации РСУБД.

6. Преобразование типа

Иногда в столбце хранятся данные одного типа, а вам необходимо интерпретировать их как данные другого типа. В SQL преобразование типа осуществляется функцией `CAST()`.

Синтаксис:

```
CAST (столбец, ТИП)
```

Допустимые значения параметра ТИП:

CHAR()

DATE

DATETIME

DECIMAL

SIGNED [INTEGER]

TIME

UNSIGNED [INTEGER]

Когда может потребоваться функция `CAST()`?

Преобразование строки к типу `DATE`:

```
SELECT CAST('2005-01-01' AS DATE);
```

Строка '2005-01-01'
преобразуется к типу
`DATE`.

Преобразование целого числа в дробное:

```
SELECT CAST(2 AS DECIMAL);
```

Целое число 2 превра-
щается в дробное 2.00.

Функция `CAST()` применяется и в других ситуациях: в списках значений `INSERT`, в списках столбцов `SELECT` и т. д.

Функция `CAST()` не может использоваться

для преобразования:

- ♦ Дробных чисел в целые;
- ♦ `TIME`, `DATE`, `DATETIME`, `CHAR` в `DECIMAL` и `INTEGER`.

7. Имя пользователя и текущее время

Иногда для одного пользователя в РСУБД создаются несколько учетных записей с разными разрешениями и ролями. Чтобы узнать, какая учетная запись используется в настоящий момент, введите следующую команду:

```
SELECT CURRENT_USER;
```

В выходных данных команды также указывается имя хоста. Если РСУБД работает на одном компьютере с пользователем, а пользователь работает с учетной записью *root*, команда выдаст следующий результат:

```
root@localhost
```

Вы можете узнать текущую дату и время с помощью следующих команд:

```
File Edit Window Help
> SELECT CURRENT_DATE;
+-----+
| CURRENT_DATE |
+-----+
| 2011-06-07   |
+-----+
1 row in set (0.00 sec)
```

```
File Edit Window Help
> SELECT CURRENT_TIME;
+-----+
| CURRENT_TIME |
+-----+
| 11:26:48     |
+-----+
1 row in set (0.00 sec)
```

```
File Edit Window Help
SELECT CURRENT_USER;
+-----+
| CURRENT_USER |
+-----+
| root@localhost |
+-----+
1 row in set (0.00 sec)
```

8. Полезные числовые функции

Ниже перечислены функции для работы с числовыми типами данных. Некоторые из них мы уже видели.

Числовая функция	Назначение	
ABS (x)	Возвращает модуль (абсолютную величину) x	
	запрос	результат
	SELECT ABS (-23) ;	23
ACOS (x)	Возвращает арккосинус x	
	SELECT ACOS (0) ;	1.5707963267949
ASIN ()	Возвращает арксинус x	
	SELECT ASIN (0.1) ;	0.10016742116156
ATAN (x, y)	Возвращает арктангенс отношения переменных x и y	
	SELECT ATAN (-2, 2) ;	-0.78539816339745
CEIL (x)	Возвращает наименьшее целое число, большее либо равное x. Возвращаемое значение имеет тип BIGINT	
	SELECT CEIL (1.32) ;	2
COS (x)	Возвращает косинус x в радианах	
	SELECT COS (1) ;	0.54030230586814
COT (x)	Возвращает котангенс x	
	SELECT COT (12) ;	-1.5726734063977
EXP (x)	Возвращает результат возведения числа e в степень x	
	SELECT EXP (-2) ;	0.13533528323661
FLOOR (x)	Возвращает наибольшее целое, меньшее либо равное x	
	SELECT FLOOR (1.32) ;	1
FORMAT (x, y)	Преобразует x в отформатированную текстовую строку с округлением до y цифр в дробной части	
	SELECT FORMAT (3452100.50, 2) ;	3,452,100.50
LN (x)	Возвращает натуральный логарифм x	
	SELECT LN (2) ;	0.69314718055995
LOG (x) and LOG (x, y)	Возвращает натуральный логарифм x, или, при использовании двух параметров, возвращает логарифм x по основанию y	
	SELECT LOG (2) ;	0.69314718055995
	SELECT LOG (2, 65536) ;	16

→ Продолжение на следующей странице.

8. Полезные числовые функции (продолжение)

Числовая функция	Описание	
MOD (x, y)	Возвращает остаток от деления x на y	
	запрос	результат
	SELECT MOD (249, 10) ;	9
PI ()	Возвращает число "пи"	
	SELECT PI () ;	3.141593
POWER (x, y)	Возвращает результат возведения x в степень y	
	SELECT POW (3, 2) ;	9
RADIANS (x)	Возвращает результат преобразования x из градусов в радианы	
	SELECT RADIANS (45) ;	0.78539816339745
RAND ()	Возвращает случайное вещественное число	
	SELECT RAND () ;	0.84655920681223
ROUND (x)	Возвращает значение x, округленное до ближайшего целого	
	SELECT ROUND (1.34) ;	1
	SELECT ROUND (-1.34) ;	-1
ROUND (x, y)	Возвращает значение x, округленное до y цифр в дробной части	
	SELECT ROUND (1.465, 1) ;	1.5
	SELECT ROUND (1.465, 0) ;	1
	SELECT ROUND (28.367, -1) ;	30
SIGN (x)	Возвращает 1, если число x положительно, 0 – если оно равно 0, или -1 для отрицательного x	
	SELECT SIGN (-23) ;	-1
SIN (x)	Возвращает синус x	
	SELECT SIN (PI ()) ;	1.2246063538224e-16
SQRT (x)	Возвращает квадратный корень из x	
	SELECT SQRT (100) ;	10
TAN (x)	Возвращает тангенс x	
	SELECT TAN (PI ()) ;	-1.2246063538224e-16
TRUNCATE (x, y)	Возвращает число x, усеченное до y цифр в дробной части	
	SELECT TRUNCATE (8.923, 1) ;	8.9

9. Индексирование для ускорения операций

Вы уже знаете, что такое индексирование по первичному и внешнему ключу. Такие индексы отлично подходят для логического связывания таблиц и обеспечения целостности данных. Однако индексы также можно строить для отдельных столбцов, чтобы ускорить обработку запросов.

Если условие `WHERE` применяется к неиндексированному столбцу, РСУБД начинает с первого значения этого столбца и последовательно проверяет все записи, одну за одной. Для большой таблицы (скажем, 4 миллиона записей) перебор займет достаточно продолжительное время.

При построении индекса для столбца РСУБД сохраняет дополнительную информацию о столбце, которая значительно ускоряет поиск. Дополнительная информация, организованная особым образом, хранится в специальной служебной таблице. За экономию времени приходится расплачиваться дополнительными затратами пространства. Следовательно, индексировать следует только те столбцы, по которым часто проводится поиск.

Общая схема выглядит так.

Определите, для каких столбцов вашей базы индексирование будет наиболее эффективным. Например, представьте, что в огромной таблице `all_contacts` часто выполняется поиск по столбцам `last_name` и `first_name`. Выполнение запросов стало занимать слишком много времени, и вы решаете, что операции необходимо ускорить.

В MySQL для построения индекса с именем `all_contacts_names` используется следующая команда:

```
ALTER TABLE my_contacts  
ADD INDEX all_contacts_names (last_name, first_name);
```

Индекс также можно построить командами следующего вида:

```
CREATE INDEX all_contacts_names  
ON all_contacts (last_name, first_name);
```

Построение индекса `all_contacts_names` имеет одно интересное последствие: при выполнении запроса к исходной таблице (например, `SELECT * FROM all_contacts`) записи будут отсортированы по значению `last_name` со вторичной сортировкой по `first_name` без указания порядка в команде.

10. PHP/MySQL за 2 минуты

Напоследок давайте очень кратко рассмотрим возможности взаимодействия с MySQL из PHP для выборки данных по Интернету. Этот крошечный раздел дает лишь начальное представление об этой возможности, и вам определенно стоит подробнее изучить эту тему.

Предполагается, что читатель уже немного разбирается в PHP, а также достаточно уверенно пишет запросы SQL. Следующий фрагмент кода подключается к базе данных с именем `gregs_list` и выполняет выборку имен и фамилий из таблицы `my_contacts`. Код PHP сохраняет все данные, загруженные из базы, в массиве. Последняя часть кода выводит список имен и фамилий на веб-странице.

```
<?php
$conn = mysql_connect("localhost", "greg", "gr3gzpAs");
if (!$conn)
{
    die('Did not connect: ' . mysql_error());
}

mysql_select_db("my_db", $conn);

$result = mysql_query("SELECT first_name, last_name FROM my_contacts");

while($row = mysql_fetch_array($result))
{
    echo $row['first_name'] . " " . $row['last_name'];
    echo "<br />";
}

mysql_close($conn);
?>
```

Файл сохраняется под именем `gregsnames.php` на веб-сервере.

Подробнее о каждой строке

```
<?php
```

Первая строка сообщает веб-серверу о том, что дальше следует код PHP.

```
$conn = mysql_connect("localhost", "greg", "gr3gzpAs");
```

Чтобы подключиться к базе данных `gregs_list`, необходимо сообщить веб-серверу ее местонахождение, имя пользователя и пароль. По этим данным создается строка подключения, которой присваивается имя `$conn`. Функция PHP `mysql_connect()` получает эту информацию и пытается использовать ее для подключения к РСУБД.

```
if (!$conn)
{
    die('Did not connect: ' . mysql_error());
}
```

Если подключение не состоялось, PHP отправляет сообщение с описанием причины, а обработка кода PHP на этом прекращается.

```
mysql_select_db("my_db", $conn);
```

Подключение к РСУБД прошло успешно. Теперь мы должны указать PHP, какая база данных нас интересует. Для выбора базы данных `gregs_list` используется команда `USE`:

```
$result = mysql_query("SELECT first_name, last_name FROM my_contacts");
```

База данных выбрана, подключение создано, но еще нет запроса для выполнения. Мы создаем такой запрос и отправляем его РСУБД функцией `mysql_query()`. Все возвращаемые записи сохраняются в массиве `$result`.

```
while($row = mysql_fetch_array($result))
{
```

Теперь синтаксис PHP используется для вывода содержимого `$result` на веб-странице. Цикл `$while` перебирает записи, пока не будет достигнут конец данных.

```
    echo $row['first_name'] . " " . $row['last_name'];
    echo "<br />";
}
```

Две команды PHP `echo` выводят на веб-странице поля `first_name` и `last_name` каждой записи. Видимые строки разделяются тегами HTML `
`.

```
    close($conn);
```

После того как все данные будут выведены, подключение к РСУБД закрывается (по аналогии с завершением сеанса работы за терминалом).

```
?>
```

Сценарий PHP на этом завершен.

Приложение II: Установка MySQL

*★
★
★*
Попробуйте сами ★



Кто же знал, что там внизу
целая РСУБД? Может,
я вообще не вернусь.

Ваши новые знания принесут пользу только в том случае, если вы сможете применить их на практике.

В этом приложении содержатся инструкции по установке РСУБД MySQL.

За дело!

Было бы странно прочитать книгу о языке SQL, не имея возможности поэкспериментировать с ним. В этом разделе приведено краткое описание процесса установки MySQL для систем Windows и Mac OS X.

ПРИМЕЧАНИЕ. Материал раздела относится к Windows 2000, XP и Windows Server 2003, а также другим 32-разрядным операционным системам семейства Windows. На платформе Mac он относится к Mac OS X 10.3.x и более поздних версий.

Итак, в этом разделе приводится пошаговое описание процесса загрузки и установки MySQL. Бесплатная версия сервера РСУБД MySQL называется **MySQL Community Server**.

Инструкции и устранение проблем

Далее приводится список основных действий по установке MySQL для Windows и Mac OS X. Этот список *не* заменяет подробные инструкции на сайте MySQL, и мы **настоятельно рекомендуем посетить сайт и ознакомиться с инструкциями!** Подробные инструкции, а также руководство по устранению проблем в ходе установки приводятся по адресу:

Вам нужна версия 5.0 и выше.

<http://dev.mysql.com/doc/refman/5.0/en/windows-installation.html>

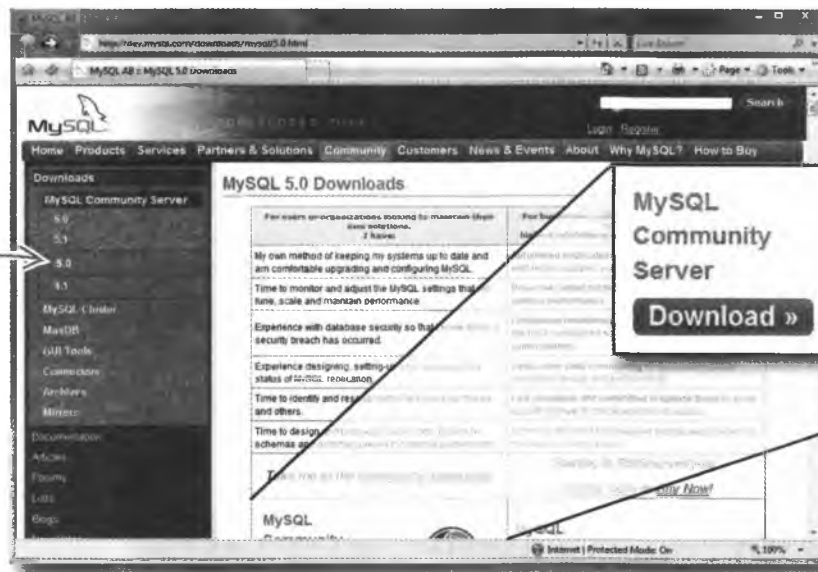
Вам также пригодится программа MySQL Query Browser, упоминавшаяся на с. 550–551. Она позволяет вводить запросы и просматривать результаты в интерфейсе приложения (вместо окна консоли).

Установка MySQL в системе Windows

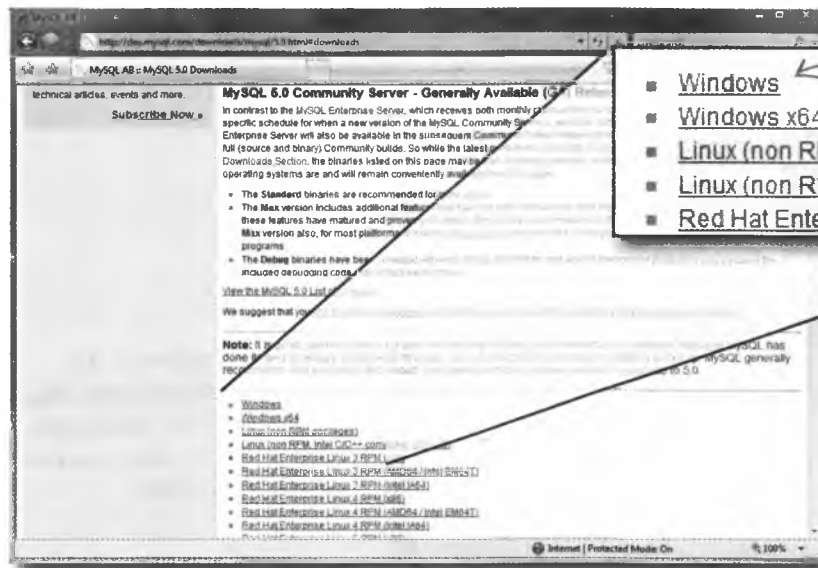
1 Откройте страницу:

<http://dev.mysql.com/downloads/mysql/5.0.html>

и щелкните на кнопке **Download** в разделе MySQL Community Server.

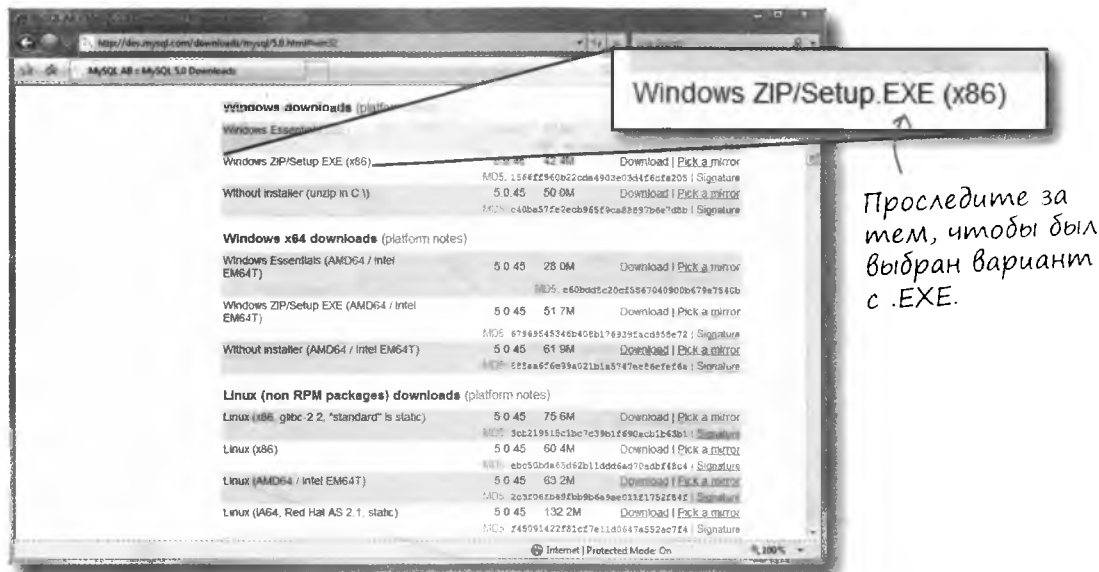


2 Выберите ссылку **Windows**.

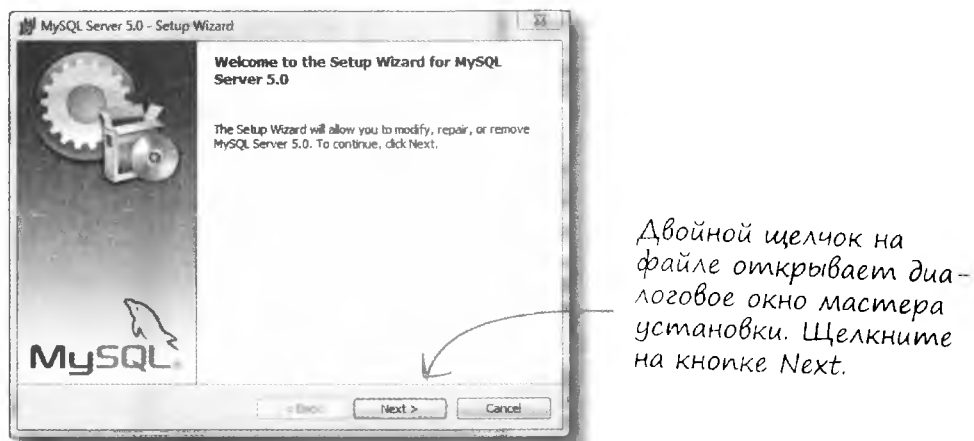


Загрузка установочного пакета

- 3 В разделе **Windows downloads** мы рекомендуем выбрать пункт **Windows ZIP/Setup.EXE**, потому что в него включена программа, сильно упрощающая процесс установки. Щелкните на ссылке **Pick a Mirror**.



- 4 Открывается список серверов, с которых можно загрузить копию установочного пакета; выберите сервер, расположенный ближе к вам.
- 5 Когда загрузка файла будет завершена, откройте его двойным щелчком. Запускается мастер установки (**Setup Wizard**), который и будет руководить процессом установки. Щелкните на кнопке **Next**.



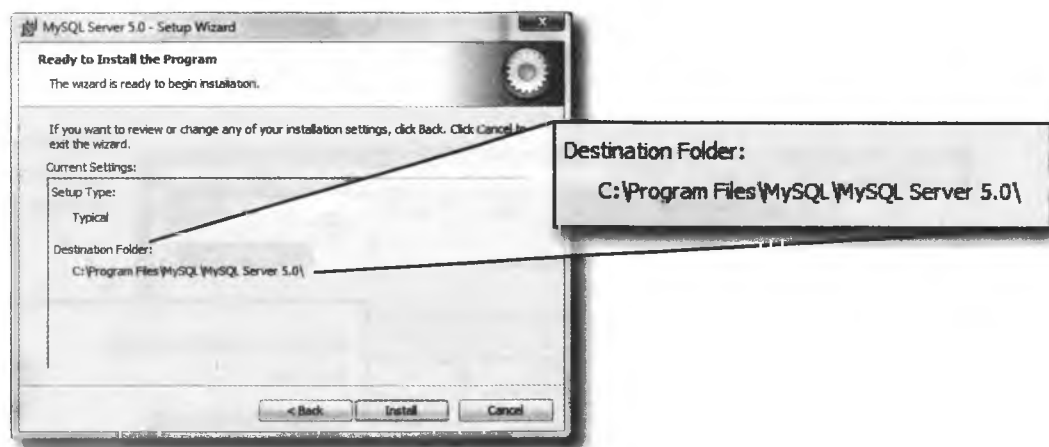
Выберите папку для установки

- 6 Вам будет предложено выбрать один из трех вариантов установки: стандартную (Typical), полную (Complete) или пользовательскую (Custom). В контексте нашей книги следует выбрать вариант **Typical**.

Вы можете выбрать папку, в которую будет установлена РСУБД MySQL на вашем компьютере, но мы рекомендуем оставить предложенный по умолчанию вариант:

C:\Program Files\MySQL\MySQL Server 5.0

Щелкните на кнопке **Next**.



Щелкните на кнопке «Install» — и все!

- 7 Открывается диалоговое окно «Ready to Install», в котором указана выбранная для установки папка. Если папка выбрана правильно, щелкните на кнопке **Install**. В противном случае щелкните на кнопке **Back**, выберите другую папку и вернитесь к этому окну.

Нажмите **Install**.

Установка MySQL в Mac OS X

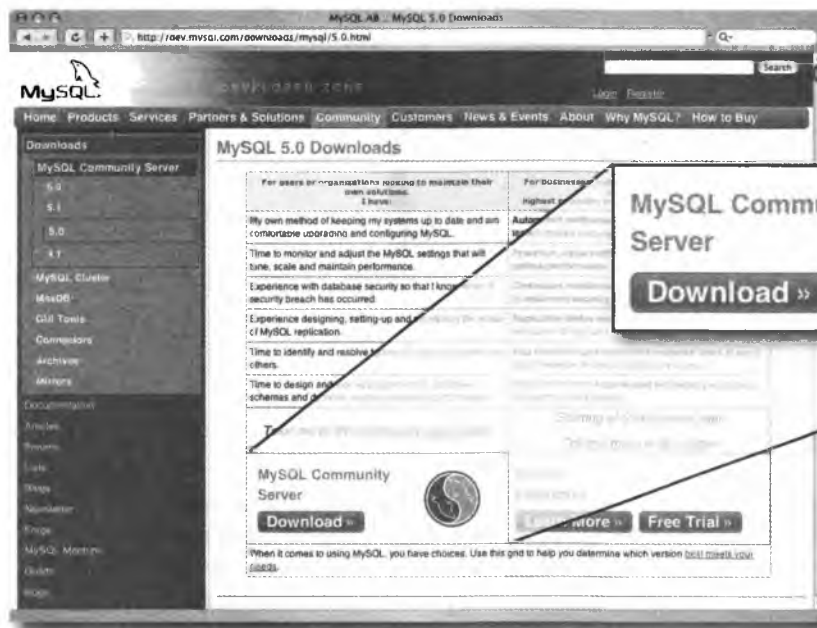
Если на вашем компьютере установлена система Mac OS X Server, то MySQL в ней уже присутствует.

Прежде чем переходить к установке, проверьте, не установлена ли версия MySQL в вашей системе (**Applications/Server/MySQL Manager**).

1 Откройте страницу:

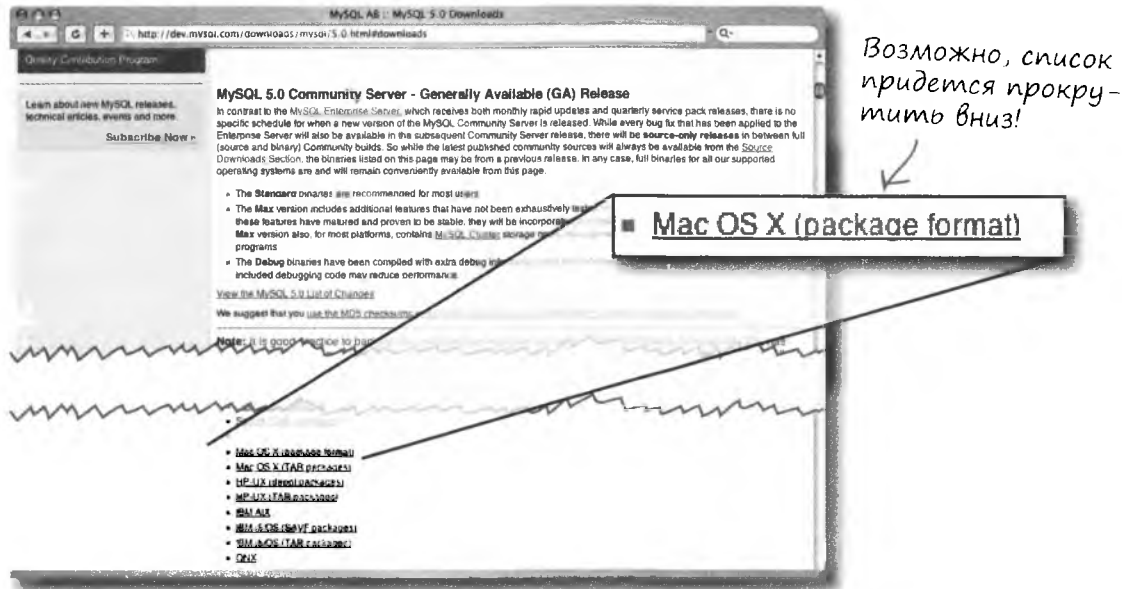
<http://dev.mysql.com/downloads/mysql/5.0.html>

и щелкните на кнопке **Download** в разделе **MySQL Community Server**.



Возможно, список придется прокрутить вниз.

2 Выберите в списке строку **Mac OS X (package format)**.



3 Выберите пакет, соответствующий вашей версии Mac OS X. Щелкните на ссылке **Pick a Mirror**.

4 Открывается список серверов, с которых можно загрузить копию установочного пакета; выберите сервер, расположенный ближе к вам.

5 Когда загрузка файла будет завершена, откройте его двойным щелчком. После установки MySQL откройте электронную документацию и посмотрите, как работать с данными в программе Query Browser, упоминавшейся на с. 550–551.

Но если вы торопитесь, ниже описан быстрый способ работы с программой в терминальном режиме.

Откройте окно терминала на Mac и введите следующие команды:

```
shell> cd /usr/local/mysql
```

```
shell> sudo ./bin/mysqld_safe
```

(Введите пароль, если потребуется.)

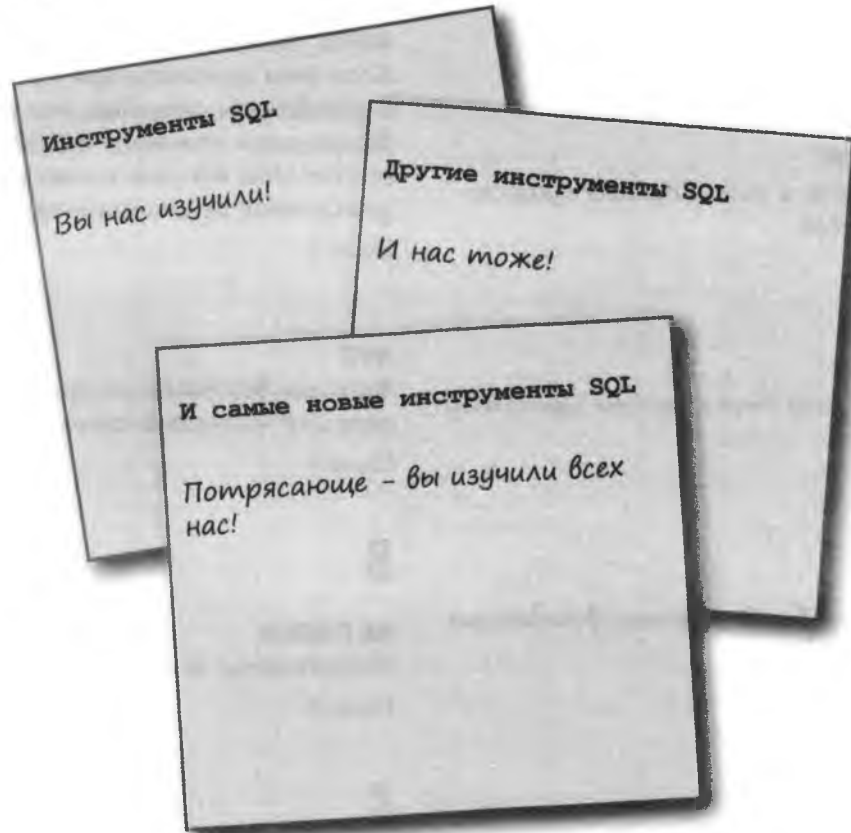
(Нажмите Control-Z.)

```
shell> bg
```

(Нажмите Control-D или введите **exit** для выхода из терминала.)

Приложение III: Список инструментов

Полный инструментарий SQL



В этом приложении перечислены все инструменты SQL, упоминавшиеся в книге. Не жалейте времени, *просмотрите весь список и возрадуйтесь* — **ведь вы изучили их все!**

Знаки

= <> < > <= >=

В вашем распоряжении целый набор операторов сравнения.

Глава 2

A

ALTER с CHANGE

Изменение имени и типа данных существующего столбца.

Глава 5

ALTER с MODIFY

Изменение только типа данных существующего столбца.

Глава 5

ALTER с ADD

Добавление столбца в таблицу в заданном порядке.

Глава 5

ALTER с DROP

Удаление столбца из таблицы.

Глава 5

ALTER TABLE

Изменение имени таблицы и всей ее структуры без потери данных.

Глава 5

AND и OR

AND и OR объединяют условия в конструкциях WHERE для повышения точности отбора.

Глава 2

AUTO_INCREMENT

Если эта конструкция присутствует в объявлении столбца, то при каждом выполнении команды INSERT для этого столбца автоматически генерируется уникальное целочисленное значение.

Глава 4

AVG

Функция возвращает среднее арифметическое для числового столбца.

Глава 6

B

BETWEEN

Определение диапазонов значений.

Глава 2

C

CHECK CONSTRAINTS

Операции вставки и обновления с таблицей выполняются только в том случае, если значения удовлетворяют заданному условию.

Глава 11

CHECK OPTION

Используется при создании обновляемых представлений; все операции вставки и обновления должны удовлетворять условию *WHERE* в определении представления.

Глава 11

COUNT

Подсчет количества записей в результатах запроса *SELECT* без их непосредственного получения. *COUNT* возвращает одно целое значение.

Глава 6

CREATE TABLE

Создание таблицы. Для выполнения команды необходимо знать *ИМЕНА* и *ТИПЫ ДАННЫХ* столбцов. Для получения этой информации следует проанализировать данные, которые вы собираетесь хранить в таблице.

Глава 1

CREATE TABLE AS

Создание таблицы по результатам произвольной команды *SELECT*.

Глава 10

CREATE USER

Команда используется в некоторых РСУБД для создания учетной записи пользователя и назначения ему пароля.

Глава 12

CROSS JOIN

Набор всех комбинаций записей одной таблицы с записями другой таблицы. Также встречаются другие названия — «декартово соединение», «декартово произведение» и др.

Глава 8

D

DELETE

удаляет записи данных из таблицы. Используйте условие *WHERE* для точного определения удаляемых записей.

Глава 3

DISTINCT

Выборка только уникальных значений, без дубликатов.

Глава 6

DROP TABLE

Команда удаляет таблицу, при создании которой была допущена ошибка — но это следует делать до выполнения команд *INSERT*, заполняющих таблицу данными.

Глава 1

E

EXCEPT

В результат включаются записи, входящие в результаты первого запроса, но **НЕ ВХОДЯЩИЕ** в результаты второго запроса.

Глава 10

G**GRANT**

Команда точно определяет, какие операции могут выполняться пользователем с таблицами и столбцами, на основании предоставленных ему привилегий.

Глава 12

GROUP BY

Группировка записей по одинаковым значениям столбца.

Глава 6

I**INTERSECT**

В результат включаются данные, присутствующие в результатах первого И второго запроса.

Глава 10

IS NULL

Условие для проверки неопределенных значений *NULL*.

Глава 2

L**LIMIT**

Условие определяет, сколько именно записей должен вернуть запрос и с какой записи следует начинать.

Глава 6

M**MAX и MIN**

MAX возвращает наибольшее значение столбца, а *MIN* — наименьшее.

Глава 6

N**NOT**

Логическое отрицание результата.

Глава 2

NULL и NOT NULL

При создании базы данных следует знать, какие столбцы не должны принимать значение *NULL* — это упростит сортировку и поиск данных. Условие *NOT NULL* задается для столбцов при создании таблицы.

Глава 1

O**ORDER BY**

Результат запроса упорядочивается по заданному столбцу.

Глава 6

S**SELECT ***

Выборка всех столбцов таблицы.

Глава 2

В

Внешний запрос

Запрос, содержащий внутренний запрос (подзапрос).

Глава 9

Внешний ключ

Столбец таблицы, значения которого ссылаются на первичный ключ другой таблицы.

Глава 7

Внутреннее соединение (INNER JOIN)

Любое соединение, комбинирующее записи двух таблиц по некоторому условию.

Глава 8

Внутренний запрос

Запрос, находящийся внутри другого запроса. Также может называться подзапросом.

Глава 9

Вторая нормальная форма (2НФ)

Таблица находится в 1НФ и не содержит частичных функциональных зависимостей.

Глава 7

Е

Естественное соединение

Внутреннее соединение без «ON». Работает только при соединении двух таблиц, содержащих одноименные столбцы.

Глава 8

Л

Левое внешнее соединение (LEFT OUTER JOIN)

Перебор всех записей ЛЕВОЙ таблицы и поиск для них соответствия среди записей ПРАВОЙ таблицы.

Глава 10

М

«Многие-ко-многим»

Две таблицы связываются через соединительную таблицу, благодаря чему многие записи первой таблицы могут быть связаны со многими записями второй — и наоборот.

Глава 7

Н

Некоррелированный подзапрос

Подзапрос, который существует сам по себе и не содержит ссылок на данные внешнего запроса.

Глава 9

Необновляемое представление

Представление, которое не может использоваться для вставки или обновления данных базовой таблицы.

Глава 11

О

Обновляемое представление

Представление, которое позволяет изменять данные в базовых таблицах. Обновляемые представления должны содержать все столбцы NOT NULL своих базовых таблиц.

Глава 11

«Один-к-одному»

Ровно одна запись родительской таблицы связывается с одной записью дочерней таблицы.

Глава 7

«Один-ко-многим»

Запись одной таблицы может быть связана со многими записями другой таблицы, но каждая запись последней может быть связана только с одной записью в первой.

Глава 7

П

Первая нормальная форма (1НФ)

Каждая запись должна содержать атомарные значения, и каждая запись должна обладать уникальным идентификатором.

Глава 4

Первичный ключ (PRIMARY KEY)

Столбец или набор столбцов, значение которого однозначно идентифицирует запись в таблице.

Глава 4

Подзапрос

Запрос, вложенный в другой запрос. Также может называться «внутренним запросом».

Глава 9

Правое внешнее соединение

Правое внешнее соединение перебирает все записи ПРАВОЙ таблицы и ищет для них соответствия среди записей ЛЕВОЙ таблицы.

Глава 10

Представление

Результат запроса, рассматриваемый как таблица. Представления особенно удобны для сокращения сложности запросов.

Глава 11

Р

Рефлексивный внешний ключ

Внешний ключ той же таблицы, в которой он является первичным ключом, используемый для других целей.

Глава 10

С

Самосоединение

Способ построения запроса к одной таблице так, как если бы она была двумя таблицами, содержащими одинаковую информацию.

Глава 10

Составной ключ

Первичный ключ, состоящий из нескольких столбцов, комбинация которых образует уникальное значение ключа.

Глава 7

Строковые функции

Функции, изменяющие копии содержимого текстовых столбцов, возвращаемые запросом. Исходные данные остаются неизменными.

Глава 5

Схема

Описание данных, хранимых в базе данных, включающее все объекты и связи между ними.

Глава 7

Т**Транзитивная функциональная зависимость**

Не-ключевой столбец связан с другим не-ключевым столбцом (-ами).

Глава 7

Третья нормальная форма (3НФ)

Таблица находится в 2НФ и не имеет транзитивных зависимостей.

Глава 7

Э**Эквивалентное и неэквивалентное соединение**

Две разновидности внутренних соединений. Эквивалентное соединение возвращает комбинации с равными значениями, а неэквивалентные — с неравными значениями столбцов.

Глава 8

Экранирование

Апострофы в текстовых данных необходимо экранировать — удвоением апострофа или обратной косой чертой.

Глава 2

Линн Бейли
Изучаем SQL

Перевел с английского Е. Матвеев

Заведующий редакцией
Руководитель проекта
Ведущий редактор
Художественный редактор
Корректор
Верстка

А. Кривцов
А. Юрченко
Ю. Сергиенко
Л. Адуевская
В. Листова
И. Смаришева

ООО «Мир книг», 198206, Санкт-Петербург, Петергофское шоссе, 73, лит. А29.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93,
том 2; 95 3005 — литература учебная.
Подписано в печать 26.08.11. Формат 84х100/16. Усл. п. л. 62,160. Тираж 2000. Заказ 26231.
Отпечатано по технологии СІР в ОАО «Первая Образцовая типография»,
обособленное подразделение «Печатный двор»,
197110, Санкт-Петербург, Чкаловский пр., 15.