

7 Многотабличные базы данных

Когда в одной таблице тесно

Мой малыш уже подросток.
Наверное, он скоро переедет.



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

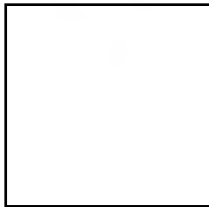
Написать запрос для Найджела: запрос должен выполнять поиск по столбцу `interests`. Что-то сложно, вроде нужно использовать **LIKE**, но на один раз сойдет...

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.

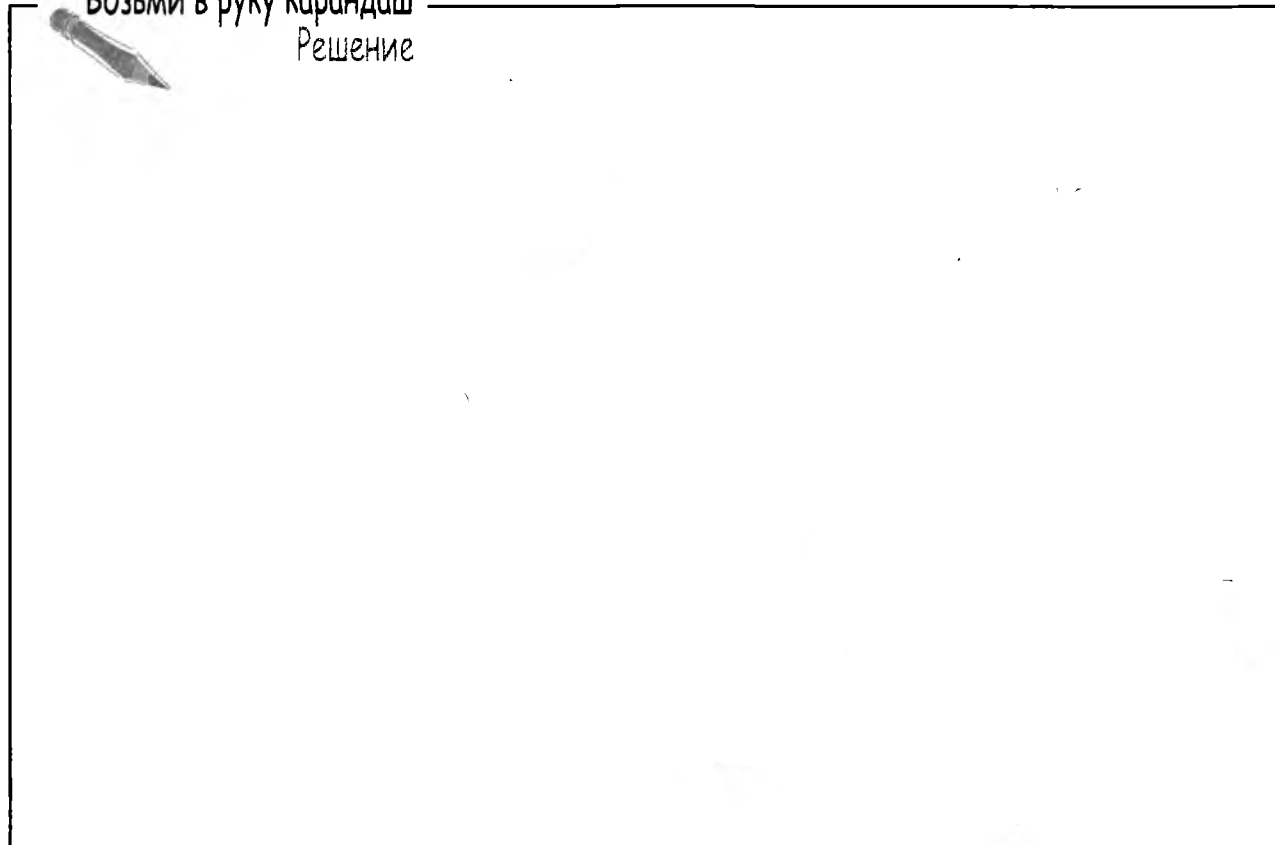
Возьми в руку карандаш



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



Запрос прекрасно сработал...

Грег находит идеальную пару для Найджела:

contact_id: 1854
last_name: Фиоре
first_name: Карла
phone: 5557894855
email: cfio@fioreanimalclinic.com
gender: Ж
birthday: 1974-01-07 ← подходит
profession: Ветеринар ← хорошая профессия
city: Раунд-Рок
state: TX ← и даже живет неподалеку
status: Не замужем
interests: лошади, кино, животные, детективы, туризм
seeking: single М ← интересы совпадают!

Карла
и Триггер



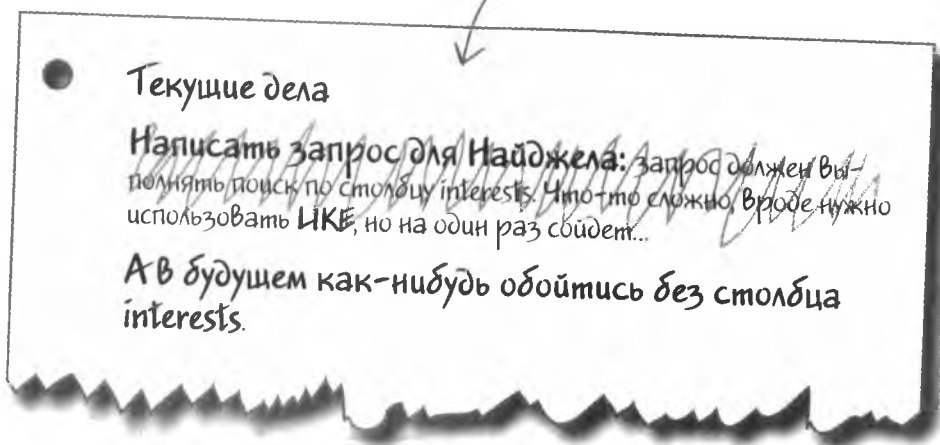
...даже слишком хорошо!

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@kathieleeisafake.com
gender: М
birthday: 1955-20-03
profession: Комик
city: Кембридж
state: МА
status: Не женат
interests: животные, коллекционные карточки, геополиск
seeking: Женщина
```

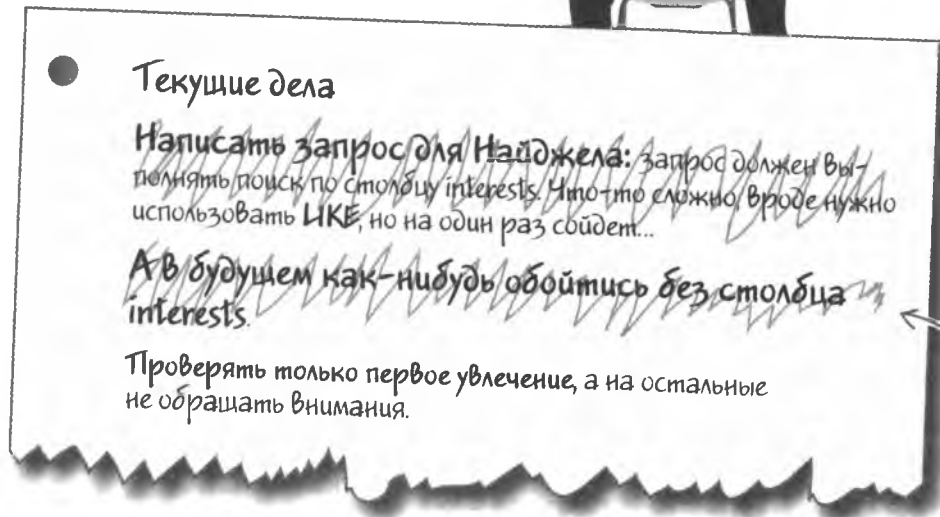
→ Ответ на с. 372.

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...



увлечения ВАЖНЫ! Их нельзя игнорировать, это ценная информация.

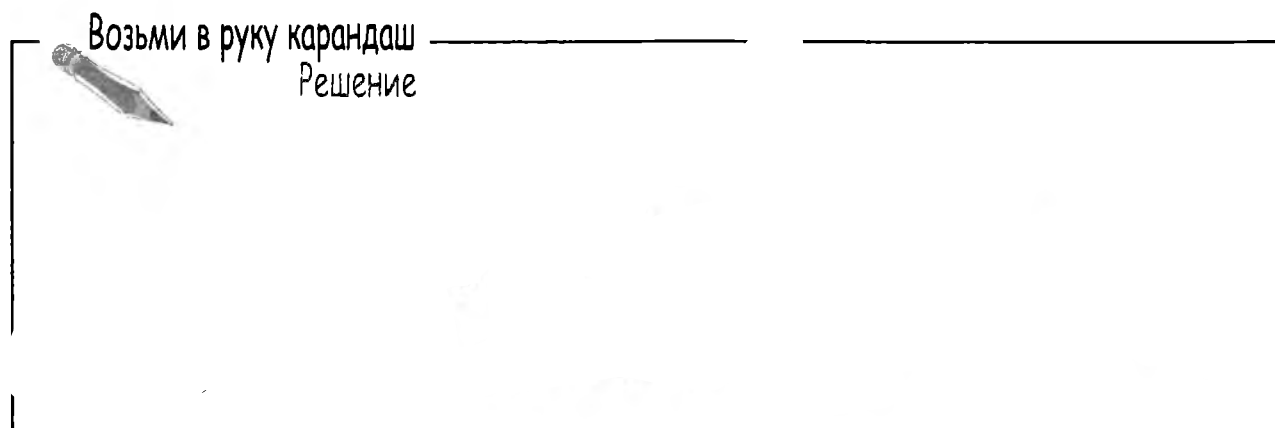
Использовать только первое увлечение

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.



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:

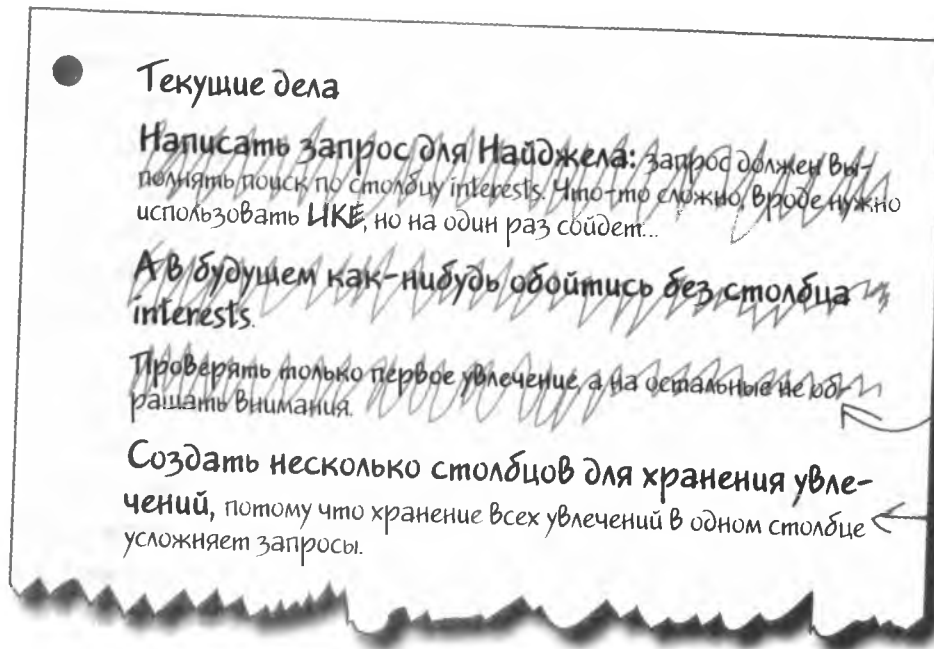
contact_id: 459
last_name: Фергюсон
first_name: Алексис
phone: 5550983476
email: alexangel@yahoo.com
gender: Ж
birthday: 1956-19-09 ← подходит
profession: Художник
city: Флиггервилль
state: MA ← живет близко
status: Не замужем
interests: животные ← подходящие увлечения
seeking: Мужчина

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!»



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: МА
status: Не женат
interest1: животные
interest2: коллекционные карточки
interest3: геополитика
interest4: NULL
seeking: Женщина
```

} В измененной таблице информация об увлечениях хранится в четырех столбцах.



Упражнение



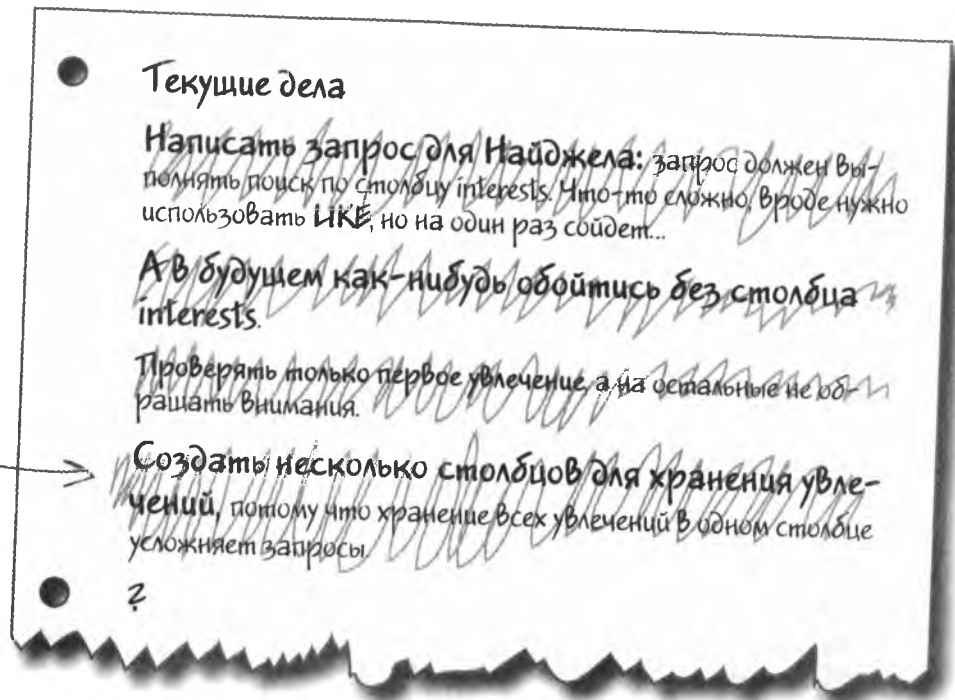
Грег пишет запрос, который должен вернуть Реджи подходящую пару. Он начинает с простых столбцов — gender, status, state, seeking и birthday — и только потом берется за столбцы interest.

Запишите его запрос.

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.

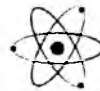


...Один момент!



о о

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`

<code>clown_info</code>	<code>name</code>	<code>last seen</code>	<code>activities</code>
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
Снагглз	Болмарт	желтая рубашка, красные штаны	рожок, зонтик

То, что раньше было главной таблицей, теперь выглядит так.

<code>clown_info</code>
<code>id</code>
<code>name</code>
<code>gender</code>
<code>description</code>

<code>info_activities</code>
<code>id</code>
<code>activity_id</code>

<code>activities</code>
<code>activity_id</code>
<code>activity</code>

<code>info_location</code>
<code>id</code>
<code>location_id</code>
<code>when</code>

<code>location</code>
<code>location_id</code>
<code>location</code>

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.

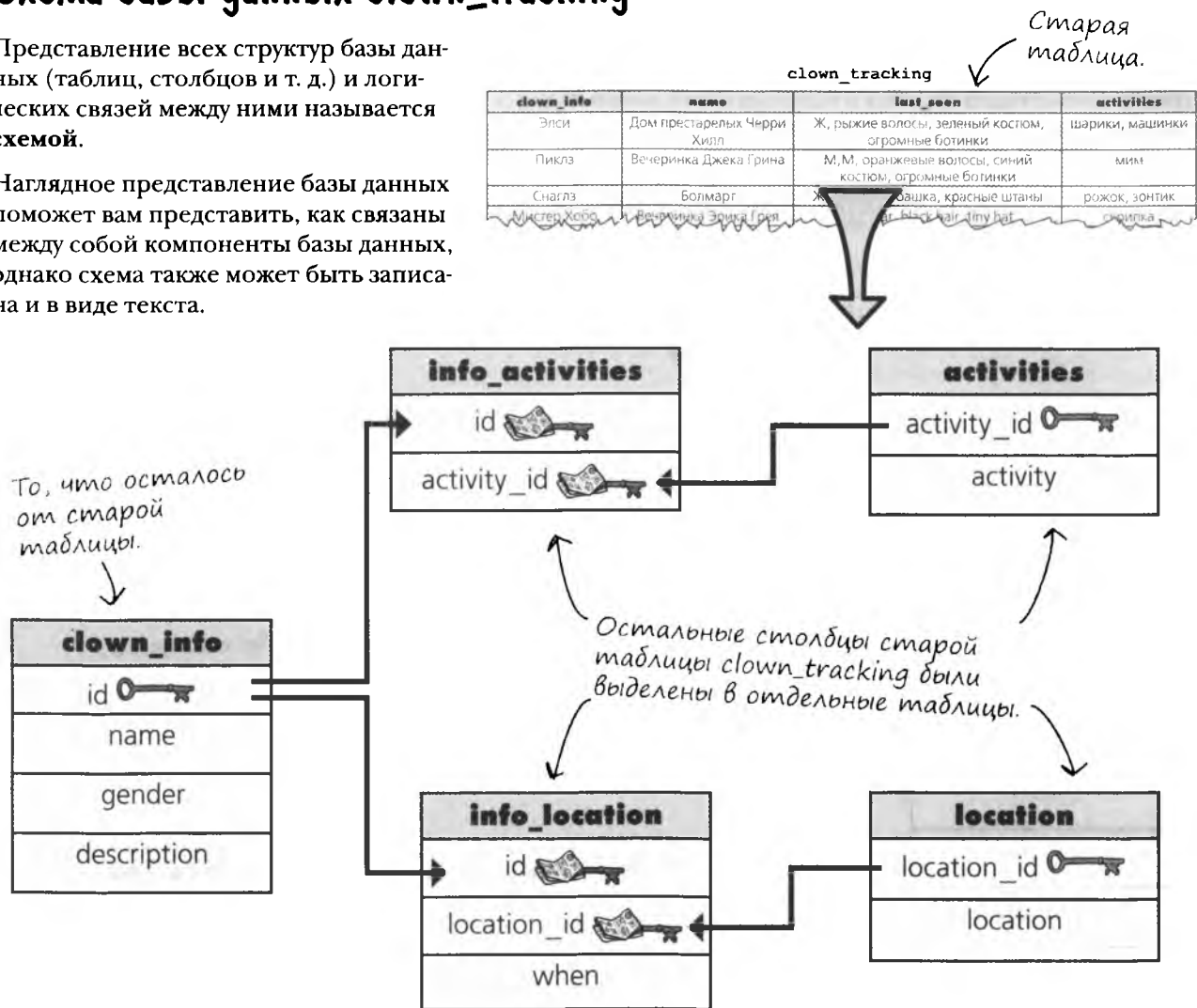


Как вы думаете, что означают линии со стрелками? А изображения ключей?

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.



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`.

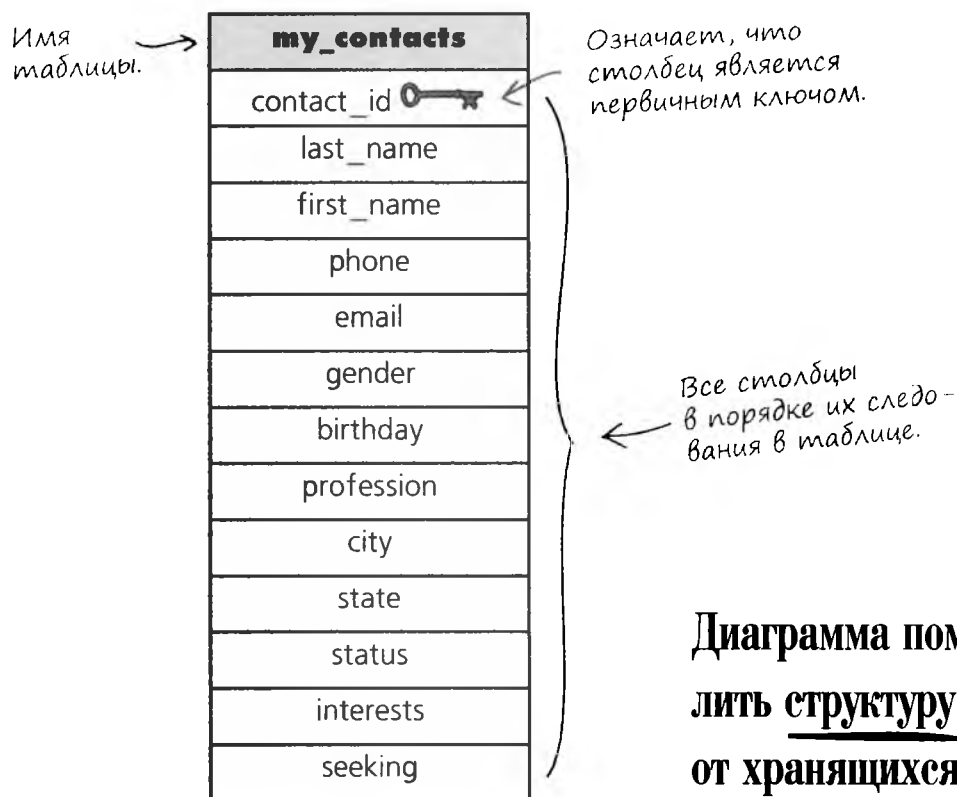


Диаграмма помогает отделить структуру таблицы от хранящихся в ней данных.

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.

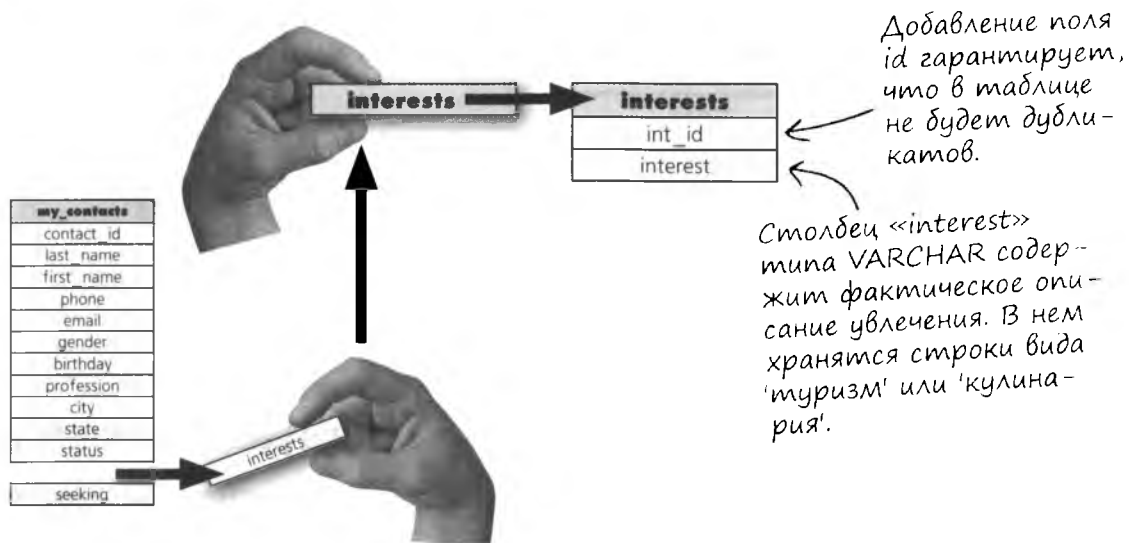
my_contacts
contact_id
last_name
first_name
phone
email
gender
birthday
profession
city
state
status
interests
seeking

Таблица `my_contacts` еще не атомарна

1

Удаляем столбец `interests` и размещаем его в отдельной таблице.

Столбец `interests` перемещается в новую таблицу.



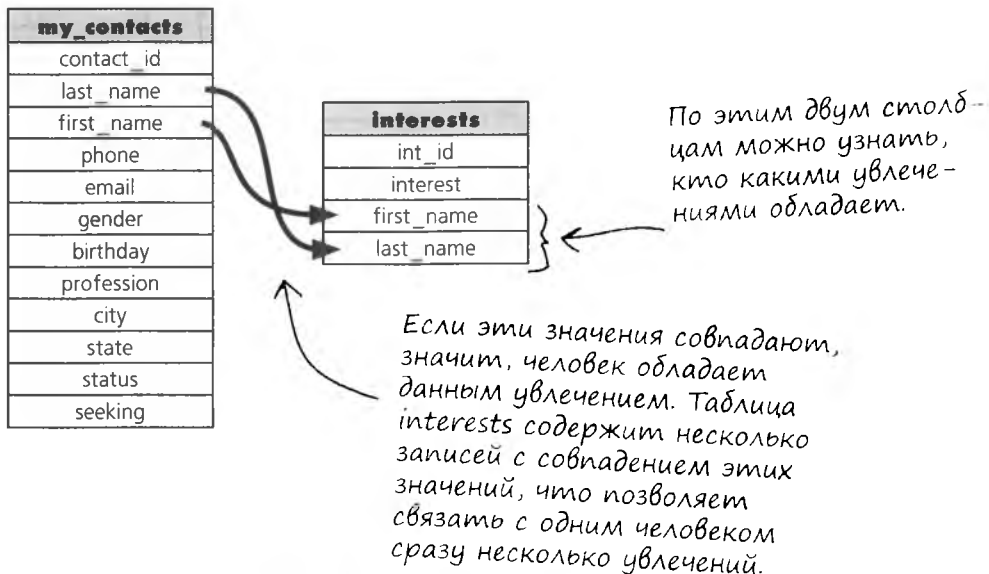
В новой таблице `interests` будут храниться все увлечения из таблицы `my_contacts` (отдельная запись для каждого увлечения).

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`.



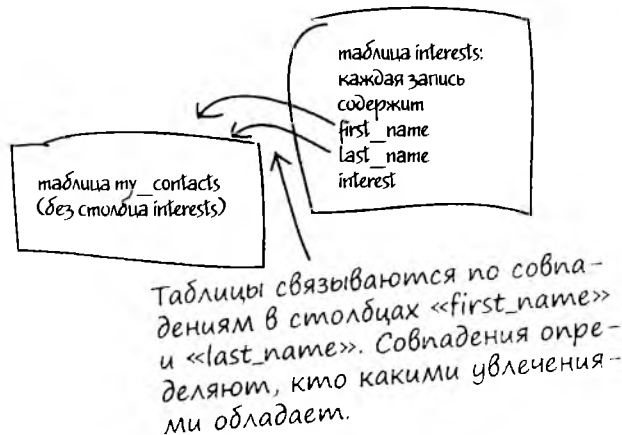
Мы движемся в верном направлении, но `first_name` и `last_name` — не лучшие столбцы для связывания таблиц.

Почему?

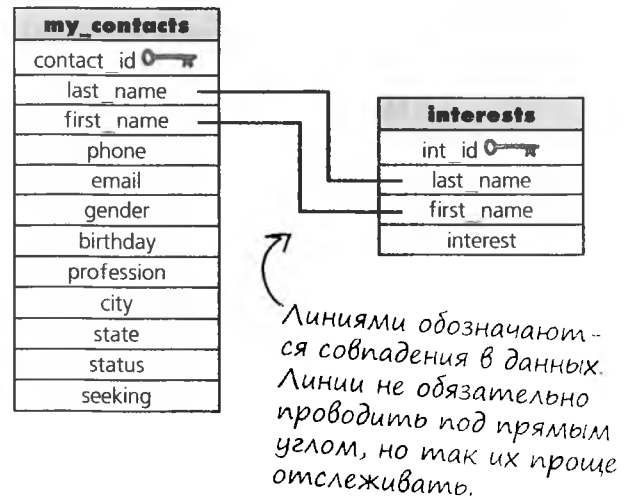
Связывание таблиц на диаграммах

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.

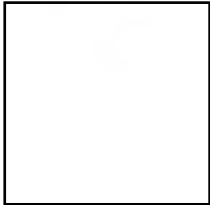
А вот как выглядит серия команд `SELECT`, которая позволит нам использовать данные из обеих таблиц.

1 `SELECT first_name, last_name`
`FROM my_contacts`
`WHERE (условия);`

2 `SELECT interest FROM interests`
`WHERE first_name = 'Имя'`
`AND last_name = 'Фамилия';`

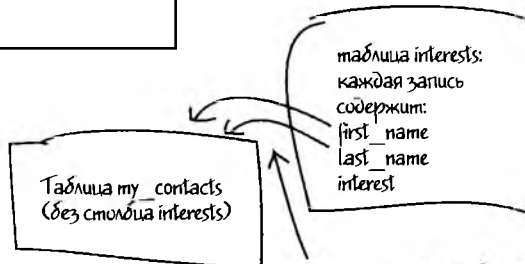
Вам кажется, что эта запись неэффективна? И правильно. Она всего лишь показывает, как использовать данные одной таблицы для извлечения данных из другой таблицы. (Вскоре мы покажем, как сделать то же самое более эффективно.)

Возьми в руку карандаш



Какие еще таблицы стоит добавить в базу данных `gregs_list` для хранения информации о нескольких увлечениях?

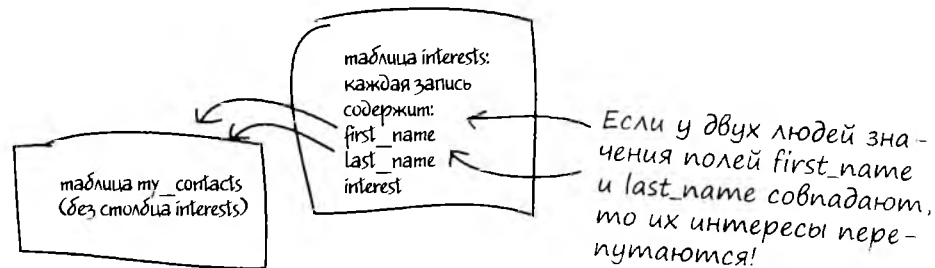
Не старайтесь нарисовать аккуратную схему; сейчас время собирать идеи. Одна идея уже изображена на рисунке, но у нее есть недостаток.



Таблицы связываются по совпадениям в столбцах «first_name» и «last_name». Совпадения определяют, кто какими увлечениями обладает.

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.

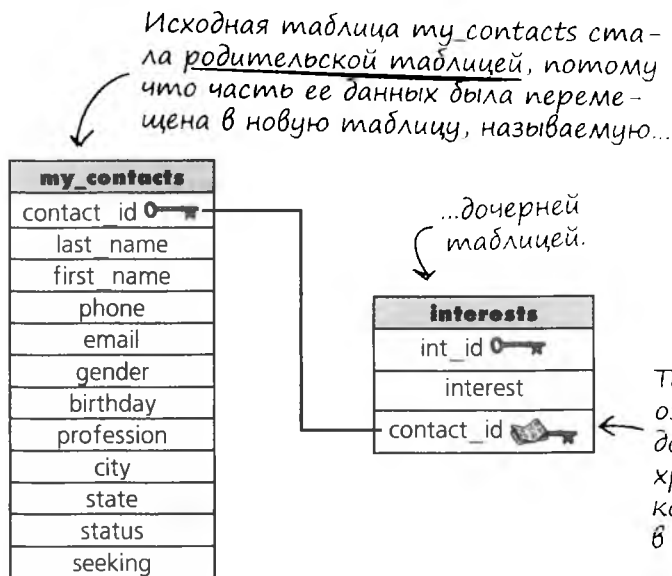
Однако мы можем сделать так, чтобы внешний ключ принимал только осмысленные значения, существующие в родительской таблице. Для этого следует воспользоваться **ограничением**.

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.



Термин «целостность данных» означает, что во внешнем ключе дочерней таблицы могут сохраняться только те значения, которые уже существуют в родительской таблице.

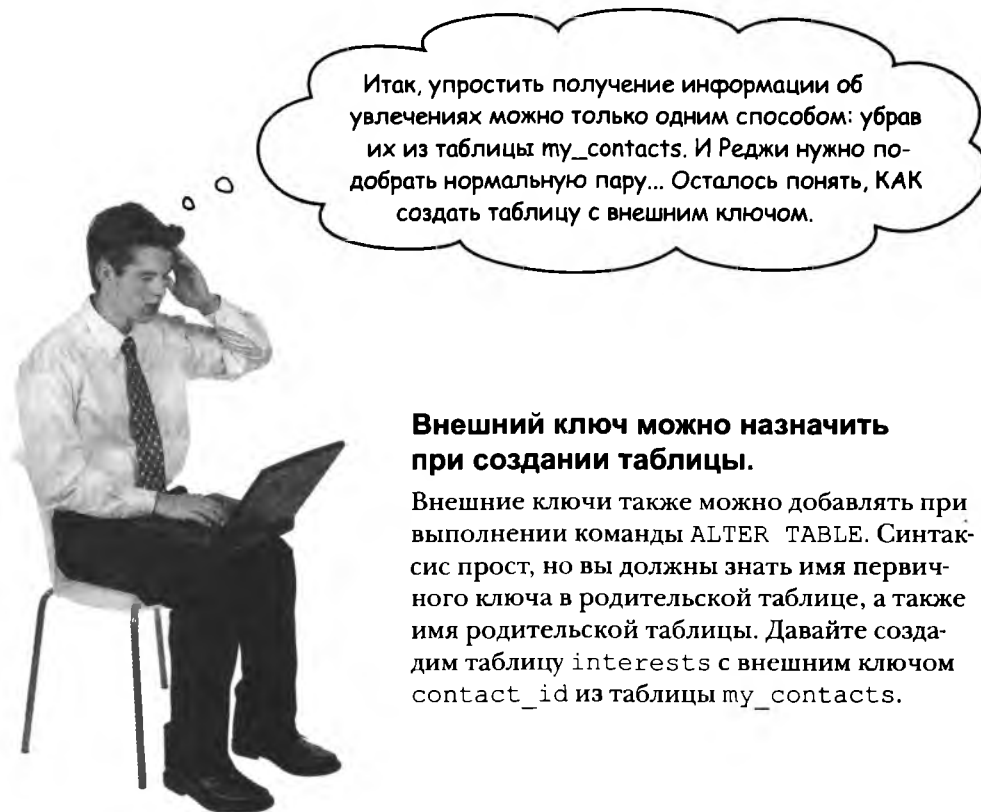
Создание ВНЕШНЕГО КЛЮЧА как ограничения таблицы дает определенные преимущества.

При попытке нарушения правила вы получите сообщение об ошибке; таким образом предотвращаются случайные нарушения связей между таблицами.

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.

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.

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.

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.



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?



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?

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`.

Часть 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.

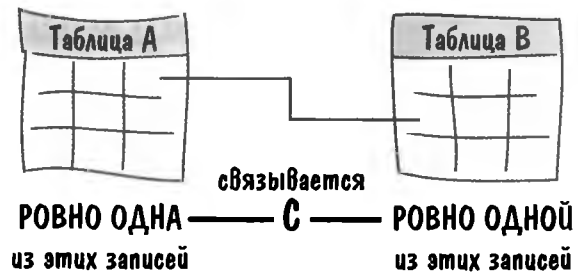
Типы связей: «один-к-одному»

Начнем с первого типа, «один-к-одному», и посмотрим, как он применяется на практике. В связях этого типа запись из таблицы A может быть связана НЕ БОЛЕЕ ЧЕМ С ОДНОЙ записью в таблице B.

Допустим, в таблице A хранится ваше имя, а в таблице B — 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»

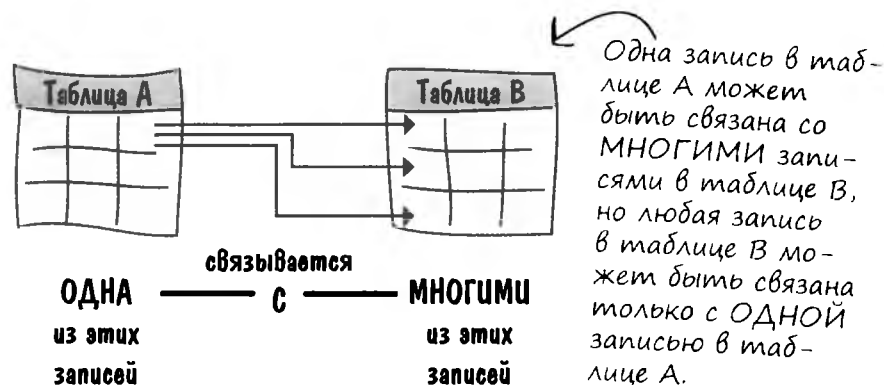
Обычно данные, связанные по типу «один-к-одному», разумнее хранить в основной таблице, однако выделение их в отдельную таблицу иногда приносит некоторые преимущества.

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.



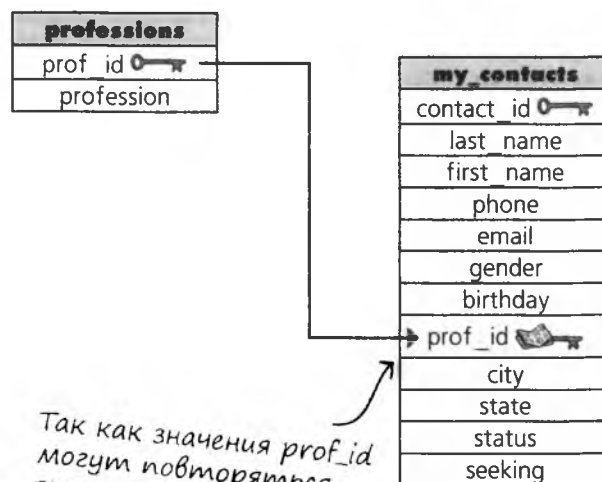
Данные о профессии в таблице `my_contacts` являются хорошим примером связей типа «один-ко-многим». Человек всегда имеет только одну профессию, но несколько человек из таблицы `my_contacts` могут иметь одинаковые профессии.

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`.

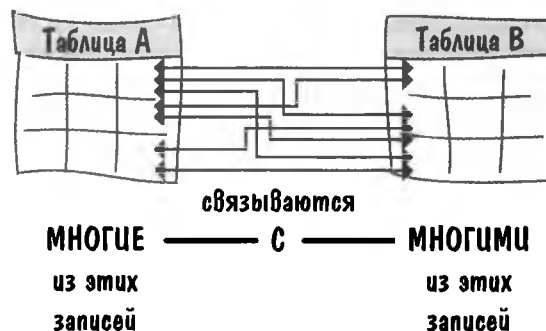


«Один-ко-многим»:
запись в таблице А может быть связана с **МНОГИМИ** записями в таблице В, но запись в таблице В может быть связана только с **ОДНОЙ** записью в таблице А.

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.



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»



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.

<code>woman_id</code>	woman	<code>shoe_id</code>
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

<code>shoe_id</code>	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.



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.

<code>woman_id</code> 🔑	woman
1	Кэрри
2	Саманта
3	Шарлотта
4	Миранда

«многие-ко-многим»

<code>shoe_id</code> 🔑	shoe_name
1	Сандалии
2	Сабо
3	Шлепанцы
4	Туфли «Прада»

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.

<code>woman_id</code>	<code>shoe_id</code>
1	3
1	4
2	1
3	1
3	2
3	3
3	4
4	1
4	3
4	4

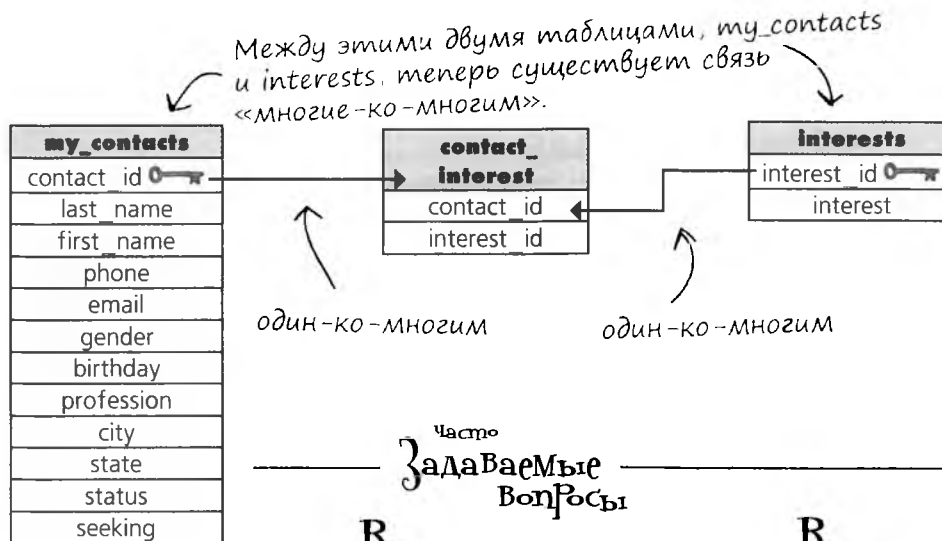
«Многие-ко-многим»: в соединительной таблице хранятся ключи из обеих таблиц.

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

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

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 Грегга

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-ко-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

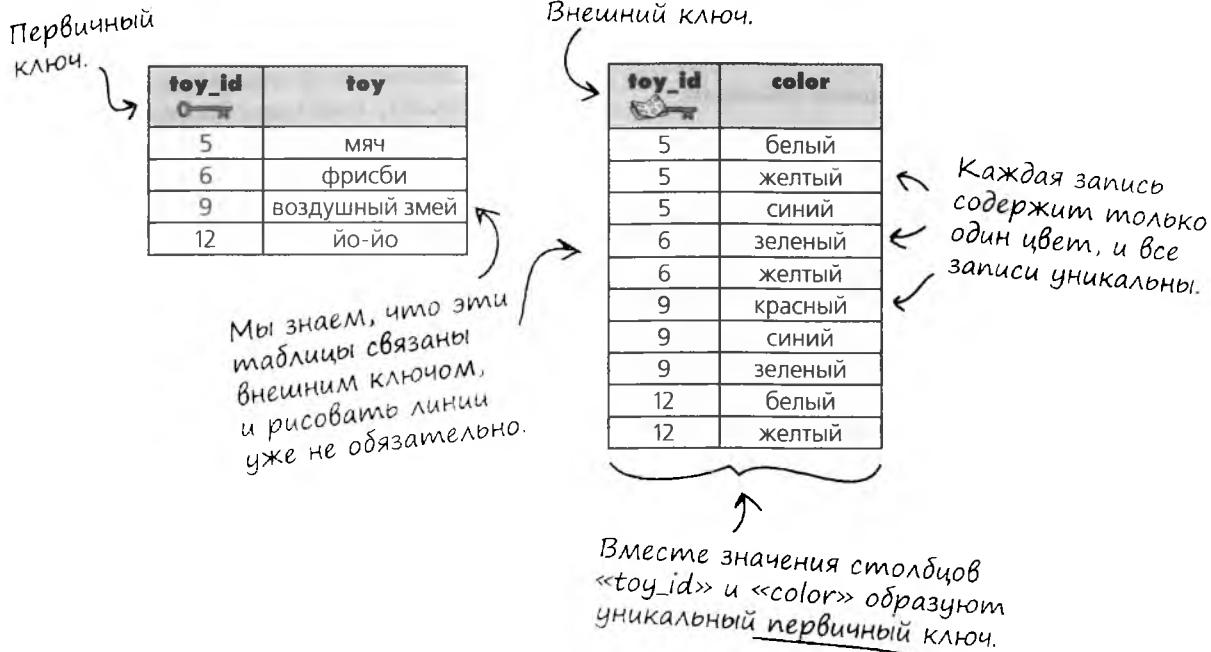
toy_id	toy	color1	color2	color3
5	мяч	белый	желтый	синий
6	фрисби	зеленый	желтый	
9	воздушный змей	красный	синий	зеленый
12	йо-йо	белый	желтый	

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



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 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 содержания одного столбца должно приводить к изменению другого, говорят, что второй столбец функционально зависит от первого.

super_heroes

name 0+*	power 0+*	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 Человек	Изображает людей	игра «Виселица»	Лондон	Англия	Ластик	НЧ

Возьми в руку карандаш



Итак, в таблице супергероев столбец `initials` зависит 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.

<code>initials</code> зависит от <code>name</code>	←	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.
<code>weakness</code> зависит от <code>name</code>	←	
<code>arch_enemy</code> зависит от <code>name</code>	←	
<code>city</code> зависит от <code>country</code>	←	

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 поменяет 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*.



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.



name	power	weakness	city	initials	arch_enemy_id	arch_enemy_city
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	Луисвиль
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 зачатый 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

Предположим, супергерой захотел поменять себе зачатого 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.

author O+K	title O+K	copyright	pub_id	pub_city
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?

Часто 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:

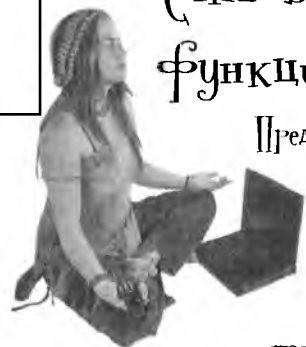
Правило 1. Таблица находится в 1НФ.

Правило 2. Таблица не имеет **частичных функциональных зависимостей**.



Вряд ли в `my_contacts` есть частичные функциональные зависимости, хотя...

Значит, пора поиграть...



Станьте таблицей 2НФ 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

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?

Третья нормальная форма, или 3НФ:

Правило 1. Таблица находится в 2НФ.

Правило 2. Таблица не имеет транзитивных зависимостей.

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 3НФ на первичный ключ можно не обращать внимания.

courses	
→	course_id
	course_name
	instructor
	instructor_phone

Чтобы таблица соответствовала требованиям 3НФ, из нее необходимо убрать столбец «instructor_phone».

Если таблица имеет синтетический первичный ключ и не имеет составного первичного ключа, она находится в 2НФ.

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.



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.)

...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, хранимых 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.