

Приложение «Книжный магазин» Схема базы данных



Авторские права

© Postgres Professional, 2017 год.

Авторы: Егор Рогов, Павел Лузанов

Использование материалов курса

Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

Обратная связь

Отзывы, замечания и предложения направляйте по адресу:

edu@postgrespro.ru

Отказ от ответственности

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или непрямым, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Обзор приложения «Книжный магазин»

Проектирование схемы данных, нормализация

Итоговая схема данных приложения



В этой демонстрации мы показываем приложение «Книжный магазин» в том виде, в котором оно будет после завершения всех практических заданий. Приложение доступно в браузере виртуальной машины курса по адресу <http://loclahost>.

Приложение состоит из нескольких частей, представленных вкладками. «Магазин» — это интерфейс веб-пользователя, в котором он может покупать книги.

Остальные вкладки соответствуют внутреннему интерфейсу, доступному только сотрудникам («админка» сайта).

«Каталог» — интерфейс кладовщика, в котором он может заказывать закупку книг на склад магазина и просматривать операции поступлений и покупок.

«Книги» и «Авторы» — интерфейс библиотекаря, в котором он может регистрировать новые поступления.

В учебных целях вся функциональность представлена на одной общей веб-странице. Если какая-то часть функциональности недоступна из-за того, что на сервере нет подходящего объекта (таблицы, функции и т. п.), приложение сообщит об этом. Также приложение выводит текст запросов, которые оно посылает на сервер.

В ходе курса мы начнем с пустой базы данных и постепенно реализуем все необходимые компоненты.

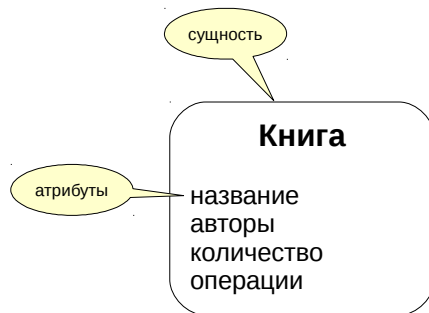
P. S. Исходный код приложения не является темой курса, но может быть получен в git-репозитории <https://git.postgrespro.ru/pub/dev1app.git>

ER-модель для высокоуровневого проектирования

сущности — понятия предметной области

связи — отношения между сущностями

атрибуты — свойства сущностей и связей



После того, как мы посмотрели внешний вид приложения и поняли его функциональность, нам нужно разобраться со схемой данных. Мы не будем хоть сколько-нибудь глубоко вникать в вопросы проектирования баз данных — это отдельная дисциплина, не входящая в этот курс. Но совсем обойти вниманием эту тему тоже нельзя.

Часто для высокоуровневого проектирования баз данных используется так называемая ER-модель (Entity-Relationship). Она оперирует сущностями (понятиями предметной области), связями между ними и атрибутами (свойствами сущностей и связей). Она позволяет рассуждать на логическом уровне, не опускаясь до деталей представления данных на физическом уровне (в виде таблиц).

Первым подходом к проектированию может служить диаграмма, представленная на слайде: можно представить одну большую сущность «Книга», а все остальное сделать ее атрибутами.

id	title	author	qty	operation
1	Муму	Тургенев Иван Сергеевич	10	+11
1	МУМУ	Тургенев Иван Сергеевич	10	-1
2	Отцы и дети	Тургенев Иван Сергеевич	4	+4
3	Трудно быть богом	Стругацкий Аркадий Натанович	7	+7
3	Трудно быть богом	Стругацкий Борис Натанович	7	0

10 = 11 - 1

7,0
или 0,7
или 7,7
?

Данные дублируются

- сложно поддерживать согласованность
- сложно обновлять
- сложно писать запросы

Очевидно, это неправильный подход. На диаграмме это может быть не так заметно, но давайте попробуем отобразить диаграмму на таблицы БД. Это можно сделать разными способами, например так, как показано на слайде: сущность становится таблицей, ее атрибуты — столбцами этой таблицы.

Здесь хорошо видно, что часть данных дублируется (эти фрагменты выделены на рисунке). Дублирование приводит к сложностям с обеспечением согласованности — а это едва ли не главная задача СУБД.

Например, каждая из двух строк, относящихся к книге 3, должна содержать общее количество (7 штук). Что нужно сделать, чтобы отразить покупку книги? С одной стороны, надо добавить строки для отражения операции покупки (а сколько строк добавлять, еще две?). С другой, во всех строках надо уменьшить количество с 7 до 6. А что делать, если в результате какой-нибудь ошибки значение количества в одной из этих строк будет отличаться от значения в другой строке? Как сформулировать ограничение целостности, которое запрещает такую ситуацию?

Также усложняются и некоторые запросы. Как найти общее число книг? Как найти всех уникальных авторов?

Итак, такая схема плохо работает для реляционных баз данных.

Схема данных (вариант)

entity	attribute	value
1	title	Муму
1	author	Тургенев Иван Сергеевич
1	qty	10
1	operation	+11
1	operation	-1
2	title	Отцы и дети
2	author	Тургенев Иван Сергеевич
2	qty	4
2	operation	+4
...

Данные без схемы

поддержка согласованности на стороне приложения

сложно писать запросы

низкая производительность (множественные соединения)

Другой вариант отображения сущности в таблицу — так называемая схема EAV: сущность-атрибут-значение. Она позволяет уложить вообще все, что угодно, в одну-единственную таблицу. Формально мы получаем реляционную базу данных, а фактически здесь отсутствует схема и СУБД не может гарантировать согласованность данных. Поддержка согласованности целиком ложится на приложение, что, безусловно, рано или поздно приведет к нарушению согласованности.

В такой схеме сложно писать запросы (хотя и довольно просто их генерировать), и в результате работа со сколько-нибудь большим объемом данных становится проблемой из-за постоянных многократных соединений таблицы самой с собой.

Так тоже делать не стоит.

(Разумеется, это не категоричное утверждение. См. последнее задание в практике.)

book_id	description
1	{ "title": "Муму", "authors": ["Тургенев Иван Сергеевич"], "qty": 10, "operations": [+11, -1] }
3	{ "title": "Трудно быть богом", "authors": ["Стругацкий Аркадий Натанович", "Стругацкий Борис Натанович"], "qty": 7, "operations": [+7] }
...	...

Данные без схемы

поддержка согласованности на стороне приложения
сложно писать запросы (специальный язык)
индексная поддержка есть

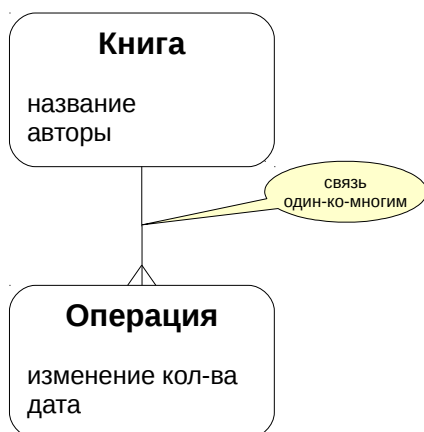
Еще одна вариация на ту же тему — представление данных в виде JSON, в модном духе NoSQL. По сути, тут применимы все замечания, сделанные ранее.

Кроме того, запросы к такой структуре придется писать не на SQL, а используя какой-то специализированный язык ([jQuery](#) или язык, определенный в стандарте SQL2016, когда он будет реализован). Хотя у PostgreSQL есть индексная поддержка JSON, производительность все равно под вопросом.

Такую схему удобно применять, когда от базы данных требуется только получение JSON по идентификатору, но не требуется серьезная работа с данными *внутри* JSON. Это не наш случай.

Нормализация — уменьшение избыточности данных

разбиение крупных сущностей на более мелкие



8

Итак, нам требуется устранить избыточность данных, чтобы привести их в вид, удобный для обработки в реляционной СУБД. Этот процесс называется нормализацией.

Возможно, вы знакомы с понятиями *нормальных форм* (первая, вторая, третья, Бойса-Кодда и т. д.). Мы не будем говорить о них; на неформальном уровне достаточно понимать, что весь этот математический аппарат преследует одну-единственную цель: устранение избыточности.

Средство для уменьшения избыточности — разбиение большой сущности на несколько меньших. Как именно это сделать, подскажет здравый смысл (который все равно нельзя заменить одним только знанием нормальных форм).

В нашем случае все достаточно просто. Давайте начнем с отделения книг от операций. Эти две сущности связаны отношением «один-ко-многим»: каждая книга может иметь несколько операций, но каждая операция относится только к одной книге.

books

book_id	title	author
1	Муму	Тургенев Иван Сергеевич
2	Отцы и дети	Тургенев Иван Сергеевич
3	Трудно быть богом	Стругацкий Аркадий Натанович
3	Трудно быть богом	Стругацкий Борис Натанович

operations

operation_id	book_id	qty_change	date_created
1	1	+10	2017-07-13
1	1	-1	2017-08-25
2	3	+7	2017-07-13
3	2	+4	2017-07-13

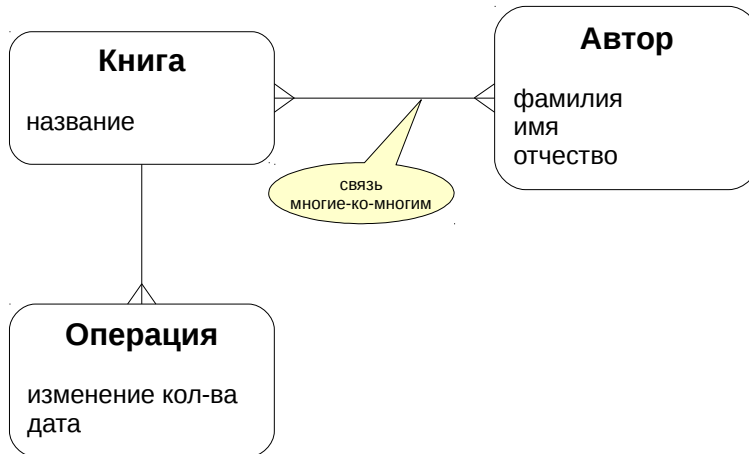
На физическом уровне это может быть представлено двумя таблицами: книги (books) и операции (authors).

Операция состоит в изменении количества книг (положительном при заказе; отрицательном при покупке). Заметьте, что теперь у книги нет атрибута «количество». Вместо этого достаточно просуммировать изменения количества по всем операциям, относящимся к данной книге. Дополнительный атрибут «количество» у книги снова привел бы к избыточности данных.

Не исключено, что такое решение вас насторожило. Удобно ли вместо простого обращения к полю подсчитывать сумму? Но мы можем создать представление, которое для каждой книги будет показывать количество. Это не приводит к появлению избыточности, поскольку представление — это всего лишь запрос.

Второй момент — производительность. Если подсчет суммы приведет к ухудшению производительности, мы можем выполнить обратный процесс — денормализацию: физически добавить поле «количество» и обеспечить его согласованность с таблицей операций. Надо ли этим заниматься — вопрос, ответ на который выходит за рамки этого курса и будет освещен в курсе DEV2. Но из общих соображений понятно, что для нашего «игрушечного» магазина это не требуется. (Но мы еще вернемся к теме денормализации в теме «Триггеры».)

Итак, как видно на слайде, выделение операций в отдельную сущность решило часть проблем дублирования, но не все.



Поэтому надо сделать еще один шаг: отделить от книг авторов и связать их соотношением «многие-ко-многим»: и каждая книга может быть написана несколькими авторами, и каждый автор может написать несколько книг.

Атрибутами автора можно сделать фамилию, имя и отчество. Это имеет смысл, поскольку нам может потребоваться работать отдельно с каждым из этих атрибутов, например, выводить фамилию и инициалы.

books

book_id	title
1	Муму
2	Отцы и дети
3	Трудно быть богом

operations

operation_id	book_id	qty_change	date_created
1	1	+11	2017-07-13
1	1	-1	2017-07-13
2	3	+7	2017-07-13
3	2	+4	2017-07-13

authorship

book_id	author_id	seq_num
1	1	1
2	1	1
3	2	1
3	3	2

порядок
указания авторов

authors

author_id	last_name	first_name	middle_name
1	Тургенев	Иван	Сергеевич
2	Стругацкий	Аркадий	Натанович
3	Стругацкий	Борис	Натанович

Вот так может выглядеть схема данных, соответствующая диаграмме. Здесь появилась таблица для авторов (authors), и, кроме того, для реализации связи «многие-ко-многим» требуется промежуточная таблица авторства (authorship).

В таблице authorship нам нужен дополнительный атрибут, определяющий, в каком порядке следует указывать авторов при выводе, если их несколько (seq_num).

Как видно, такое разделение устраняет избыточность в данных. Поддерживать согласованность в такой схеме становится просто.

Числовые

целые, вещественные с плавающей и фиксированной запятой
целые с автоувеличением (используют последовательности)

Символьные

с фиксированной и **переменной длиной**

Даты

дата, дата и время (с часовым поясом и без), интервалы

Логический — трехзначная логика с NULL

Составные

Массивы

...

12

Столбцы таблиц должны иметь определенный тип данных. Тип определяет, какие значения могут храниться в этом столбце и какие операции определены над этими значениями.

В этом курсе мы будем использовать:

- целые числа (integer);
- целые с автоувеличением (serial);
- символьный тип без ограничения длины (text);
- даты без времени (date);
- логический тип (boolean) с трехзначной логикой (true, false и null).

Псевдотип serial — обычный тип integer, но значение поля по умолчанию выбирается из последовательности — специального объекта БД, который служит для генерации уникальных номеров.

<https://postgrespro.ru/docs/postgresql/9.6/functions-sequence.html>

Кроме того, мы подробно рассмотрим более сложные типы:

- составной — записи, аналогичные строкам таблиц;
- массивы.

PostgreSQL также поддерживает множество других типов, включая геометрические, диапазоны и перечислимые типы, «сложносоставные» типы (XML, JSON) и др. В целом, благодаря расширяемой архитектуре PostgreSQL, можно реализовать и подключить любые типы данных; многие можно найти в сторонних расширениях.

<https://postgrespro.ru/docs/postgresql/9.6/datatype.html>

<https://postgrespro.ru/docs/postgresql/9.6/functions.html>

Первичный ключ (primary key) и уникальность (unique)

поддерживаются индексом

Обязательность (not null)

Внешний ключ (foreign key)

Проверка (check)

на уровне одной строки

Исключение (exclude)

на уровне всех строк таблицы

поддерживается специальным индексом

Для таблиц могут быть заданы ограничения целостности. PostgreSQL поддерживает весь стандартный набор ограничений.

В курсе мы будем использовать:

- первичные ключи (обычно в сочетании с типом serial) — что подразумевает уникальность и обязательность;
- обязательность;
- внешние ключи (для обеспечения ссылочной целостности);
- ограничения проверки (для дополнительной поддержки согласованности).

Специфичным для PostgreSQL является «исключение», которое позволяет задать ограничение не на уровне отдельной строки, а для всей таблицы сразу.

Ограничения могут проверяться постоянно, либо откладываться до конца транзакции. В любом случае PostgreSQL гарантирует свойство согласованности: транзакция должна переводить базу данных из одного целостного состояния в другое.

Дополнительные ограничения согласованности данных, не представимые в виде стандартных ограничений целостности, можно реализовать с помощью триггеров.

<https://postgrespro.ru/docs/postgresql/9.6/ddl-constraints.html>



Проектирование баз данных — отдельная тема

теория важна, но не заменяет здравый смысл

**Отсутствие избыточности в данных делает работу удобнее
и упрощает поддержку согласованности**



1. Создайте базу данных bookstore.
2. Создайте схему bookstore и настройте путь поиска к ней на уровне подключения к базе данных.
3. В схеме bookstore создайте таблицы для приложения с необходимыми ограничениями целостности. Точный список приведен в комментариях к слайду.
4. Вставьте в таблицы данные о нескольких книгах. Проверьте себя с помощью запросов.
5. Зайдите в приложение. Видны ли в нем введенные данные?

books

```
book_id    serial,  первичный ключ
title      text
```

authors

```
author_id  serial,  первичный ключ
last_name  text
first_name text
middle_name text
```

authorship

```
book_id    integer, первичный ключ, внешний ключ для books
author_id  integer, первичный ключ, внешний ключ для authors
seq_num    integer
```

operations

```
operation_id serial,  первичный ключ
book_id      integer, не NULL, внешний ключ для books
qty_change   integer
date_created date,   текущая дата по умолчанию
```


1. Какие дополнительные атрибуты могут появиться у выделенных сущностей при развитии приложения?
2. Допустим, требуется хранить информацию об издательстве. Дополните ER-диаграмму и отобразите ее в таблицы.
3. Некоторые книги могут входить в серии (например, «Библиотека приключений»). Как изменится схема данных?
4. Пусть наш магазин стал торговать компьютерными комплектующими (материнскими платами, процессорами, памятью, жесткими дисками, мониторами и т. п.).
Какие сущности и какие атрибуты вы бы выделили?
Учтите, что на рынке постоянно появляются новые типы оборудования со своими характеристиками.

3. Разные издательства вполне могут иметь серии, названные одинаково.