

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 косвенным, 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.

«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.

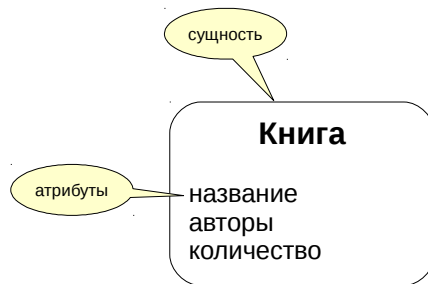
Р. S. Исходный код приложения не является темой курса, но может быть получен в git-репозитории <https://git.postgrespro.ru/pub/dev1app.git>

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.

Часть для высокоуровневого проектирования баз данных используется так называемая ER-модель (Entity-Relationship). Она оперируется 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.

book_id	title	last_name	first_name	surname	qty
1	Муму	Тургенев	Иван	Сергеевич	10
2	Отцы и дети	Тургенев	Иван	Сергеевич	4
3	Трудно быть богом	Стругацкий	Аркадий	Натанович	7
3	Трудно быть богом	Стругацкий	Борис	Натанович	7

Данные дублируются

сложно поддерживать согласованность

сложно обновлять

сложно писать запросы

5

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, такие данные сложно обновлять. Чтобы увеличить количество книги 3, придется обновить две строки. А если в результате какой-нибудь транзакции значение qty 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	last_name	Тургенев
1	first_name	Иван
1	surname	Сергеевич
1	qty	10
2	title	Отцы и дети
2	last_name	Тургенев
2	first_name	Иван
2	surname	Сергеевич
2	qty	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.)

Схема данных (вариант)

book_id	description
1	<pre>{ "title": "Муму", "authors": [{ "last_name": "Тургенев", "first_name": "Иван", "surname": "Сергеевич" }], "qty": 10 }</pre>
3	<pre>{ "title": "Трудно быть богом", "authors": [{ "last_name": "Стругацкий", "first_name": "Аркадий", "surname": "Натанович" }, { "last_name": "Стругацкий", "first_name": "Борис", "surname": "Натанович" }], "qty": 7 }</pre>
...	...

Данные без схемы

поддержка согласованности на стороне приложения

сложно писать запросы (специальный язык)

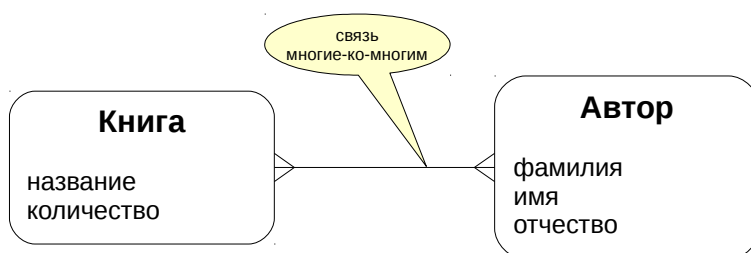
индексная поддержка есть

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



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 знанием нормальных форм).

В нашем случае все просто: надо отделить авторов от книг и связать их соотношением «многие-ко-многим».

books

book_id	title	qty
1	Муму	10
2	Отцы и дети	4
3	Трудно быть богом	7

authorship

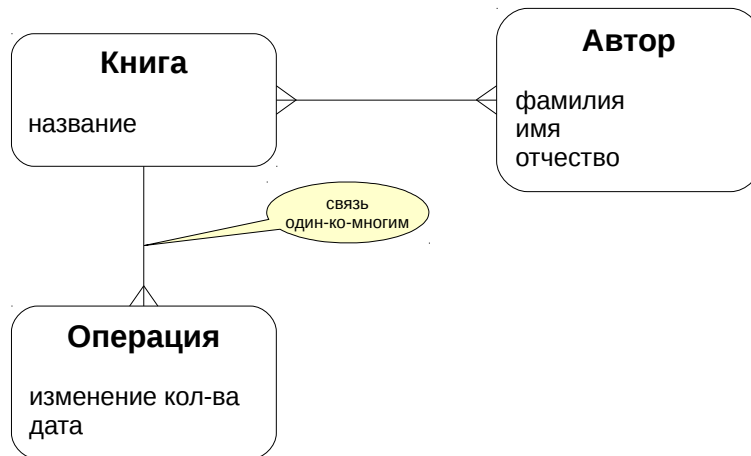
book_id	author_id
1	1
2	1
3	2
3	3

authors

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

На уровне таблиц это может быть представлено тремя таблицами: две для книг (books) и авторов (authors), и одна вспомогательная для реализации связи «многие-ко-многим» — авторство (authorship).

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



Мы пойдем чуть дальше и выделим еще одну сущность — операции.

В нашем магазине приобретение книги и заказ новых книг выполняется мгновенно, но понятно, что это всего лишь упрощение. В реальности и один и другой процесс занимают время и требуют отслеживания.

В нашем случае операция будет состоять 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.

Схема данных

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	10	2017-07-13
2	3	7	2017-07-13
3	2	4	2017-07-13

authorship

book_id	author_id
1	1
2	1
3	2
3	3

authors

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

Вот так может выглядеть схема данных, соответствующая диаграмме. Для реализации связи «один-ко-многим» промежуточная таблица не требуется, достаточно внешнего ключа.

Возможно, вы насторожились: вместо одного поля «количество» теперь придется подсчитывать сумму значений из другой таблицы.

Тут можно выделить несколько моментов. Первый — удобство. Мы можем создать представление, которое будет показывать количество для каждой книги. Это не приводит к появлению избыточности, поскольку представление — это всего лишь запрос.

Второй — 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.

Третий — согласованность данных. Как не допустить ситуации, когда на складе оказывается отрицательное количество книг? Этим мы займемся в теме «Триггеры» и снова придем к денормализации.

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
onhand_qty	integer

authors

author_id	serial, первичный ключ
last_name	text
first_name	text
surname	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, 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.