11 сентября 2013 в 02:02

Руководство по проектированию реляционных баз данных (4-6 часть из 15) [перевод]

перевод

SQL*, MySQL*

Выкладываю продолжение перевода цикла статей для новичков.
В настоящих и последующих — больше информации по существу.
Начало — здесь.

4. ТАБЛИЦЫ И ПЕРВИЧНЫЕ КЛЮЧИ

Как вы уже знаете из прошлых частей, данные хранятся в **таблицах**, которые содержат **строки** или по-другому **записи**. Ранее я приводил пример таблицы, содержащей информацию об уроках. Давайте снова на нее взглянем.

tutorial_id	title	category
1	Access Tutorial	Software
2	Excel Tutorial	Software
3	**Database design tutorial**	**Software**
4	Oracle DBA Course	Software
5	Raid Storage Tutorial	Hardware
6	Network Security Tutorial	Networks

В таблице имеются 6 уроков. Все 6 – разные, но для каждого урока значения одинаковых полей хранятся в таблице, а именно: tutorial_id (идентификатор урока), title (заголовок)и category (категория). **Tutorial_id** – **первичный ключ** таблицы уроков. Первичный ключ – это значение, которое уникально для каждой записи в таблице.
В таблице клиентов ниже customer_id – первичный ключ. В данном случае первичный ключ – также уникальное значение (число) для каждой записи.

customer_id	first_name	last_name	email_address	telephone
0	John	Smith	john.smith@sor	123456789
1	John	Connor	j.connor@anotl	987498473

Первичные ключи в повседневной жизни

В базе данных первичные ключи используются для идентификации. В жизни первичные ключи вокруг нас везде. Каждый раз, когда вы сталкиваетесь с уникальным числом это число может служить первичным ключом в базе данных (может, но не обязательно должно использоваться как таковое. Все

Популярное за сутки

Слив данных 180 тысяч пользователей FL.ru

Сайт с нуля на полном стеке БЭМ-технологий. Методология Яндекса

Роутер от оператора? – Нет, спасибо!

12 игр, которые обучают детей программированию

Арабская локализация: окна и рисование

Consulo: Code Coverage, Unity3D и прочие изменения

Fujitsu ETERNUS CD10000: Ceph без забот

Что должен уметь крутой колл-центр по IT-части и какие вообще бывают опции

Мультивендорная корпоративная сеть: мифы и реальность

Генерация текстур планет как в игре Star Control 2

Еще один термостат на Arduino, но с OpenTherm

Удалённое управление для Arduino, проба пера

Ещё один программный UART на ATtiny13

базы данных способны автоматически генерировать уникальное значение для каждой записи в виде числа, которое автоматически увеличивается и вставляется вместе с каждой новой записью [Т.н. синтетический или суррогатный первичный ключ – прим.перев.]).

Несколько примеров

- Номер заказа, который вы получаете при покупке в интернет-магазине может быть первичным ключом какой-нибудь таблицы заказов в базе данных этого магазина, т.к. он является уникальным значением.
- Номер социального страхования может быть первичным ключом в какой-нибудь таблице в базе данных государственного учреждения, т.к. она также как и в предыдущем примере уникален.
- Номер счета-фактуры может быть использован в качестве первичного ключа в таблице базы данных, в которой хранятся выданные клиентам счета-фактуры.
- Числовой номер клиента часто используется как первичный ключ в таблице клиентов.
- ...

Что объединяет эти примеры? То, что во всех из них в качестве первичного ключа выбирается уникальное, не повторяющееся значение для каждой записи. Еще раз. Значения поля таблицы базы данных, выбранного в качестве первичного ключа, всегда уникально.

Что характеризует первичный ключ? Характеристики первичного ключа.

Первичный ключ служит для идентификации записей.

Первичный ключ используется для **идентификации** записей в таблице, для того, чтобы каждая запись стала уникальной. Еще одна аналогия… Когда вы

звоните в службу технической поддержки, оператор обычно просит вас назвать какой-либо номер (договора, телефона и пр.), по которому вас можно идентифицировать в системе.
Если вы забыли свой номер, то оператор службы технической поддержки попросит предоставить вас какую-либо другую информацию, которая поможет уникальным образом идентифицировать вас. Например, комбинация вашего дня рождения и фамилия. Они тоже могут являться первичным ключом, точнее их комбинация.

Первичный ключ уникален.

Первичный ключ всегда имеет уникальное значение. Представьте, что его значение не уникально. Тогда его бы нельзя было использовать для того, чтобы идентифицировать данные в таблице. Это значит, что какое-либо значение первичного ключа может встретиться в столбце, который выбран в качестве первичного ключа, только один раз. РСУБД устроены так, что не позволят вам вставить дубликаты в поле первичного ключа, получите ошибку. Еще один пример. Представьте, что у вас есть таблица с полями first_name и last_name и есть две записи:

first_name	last_name
vasya	pupkin
vasya	pupkin

Т.е. есть два Васи. Вы хотите выбрать из таблицы какого-то конкретного Васю. Как это сделать? Записи ничем друг от друга не отличаются. Вот здесь и помогает первичный ключ. Добавляем столбец id (классический вариант синтетического первичного ключа) и…

Id | first_name | last_name |
1 | vasya |pupkin |
2 | vasya |pupkin |

Госорганам РФ предлагают закупать только отечественный софт с 1 июля 2015 года

Цифровой аудиоформат 24/192, и почему в нем нет смысла. Часть 4 (и последняя)

все публикации

Лучшее на Мегамозге

5 главных ошибок или почему ваши рациональные решения не работают?

Инвестиционный фонд Runa Capital вкладывается в СУБД MariaDB

Философия системы Asana. 4 принципа работы в системе

Апофеоз неудачника или чем мне нравятся провальные стартапы и их основатели

Parallels приобрела разработчика приложений для удаленного доступа 2X Software

Управление ретроспективой или взгляд в прошлое: Руководство Gov.uk

По итогам 2014 года, игры и социальные сервисы – основные точки роста доходов Mail.ru Group

Forbes составил рейтинг 20 самых дорогих компаний Рунета

Mail.ru Group сообщила о неопределенности в продаже HeadHunter

5 шагов к созданию привлекательного сервиса для приложения

Теперь каждый Вася уникален.

Типы первичных ключей.

Обычно первичный ключ – числовое значение. Но он также может быть и любым другим типом данных. Не является обычной практикой использование строки в качестве первичного ключа (строка – фрагмент текста), но теоретически и практически это возможно.
Составные первичные ключи.
Часто первичный ключ состоит из одного поля, но он может быть и комбинацией нескольких столбцов, например, двух (трех, четырех…). Но вы помните, что первичный ключ всегда уникален, а значит нужно, чтобы комбинация n-го количества полей, в данном случае 2-х, была уникальна. Подробнее об этом расскажу позднее.

Автонумерация.

Поле первичного ключа часто, но не всегда, обрабатывается самой базой данных. Вы можете, условно говоря, сказать базе данных, чтобы она сама автоматически присваивала уникальное числовое значение каждой записи при ее создании. База данных, обычно, начинает нумерацию с 1 и увеличивает это число для каждой записи на одну единицу. Такой первичный ключ называется автоинкрементным или автонумерованным. Использование автоинкрементных ключей – хороший способ для задания уникальных первичных ключей. Классическое название такого ключа – суррогатный первичный ключ [Как и упоминалось выше. – прим. перев.]. Такой ключ не содержит полезной информации, относящейся к сущности (объекту), информация о которой хранится в таблице, поэтому он и называется суррогатным.

5. СВЯЗЫВАНИЕ ТАБЛИЦ С ПОМОЩЬЮ ВНЕШНИХ КЛЮЧЕЙ

Когда я начинал разрабатывать базы данных я часто пытался сохранять информацию, которая казалась родственной, в одной таблице. Я мог, например, хранить информацию о заказах в таблице клиентов. Ведь заказы принадлежат клиентам, верно? Нет. Клиенты и заказы представляют собой отдельные сущности в базе данных. И тому и другому нужна своя собственная таблица. А записи в этих двух таблицах могут быть связаны для того, чтобы установить отношения между ними. Проектирование базы данных – это решение двух вопросов:

- определение того, какие сущности вы хотите хранить в ней
- какие связи между этими сущностями существуют

Один-ко-многим.

Клиенты и заказы имеют связь (состоят в отношениях) **один-ко-многим** потому, что **один** клиент может иметь **много** заказов, но каждый конкретный заказ (их **множество**) оформлен только **одним** клиентом, т.е. может иметь только одного клиента. Не беспокойтесь, если на данный момент понимание этой связи смутно. Я еще расскажу о связях в следующих частях.

Одно является важных сейчас – то, что для связи один-ко-многим необходимо **две** отдельные таблицы. Одна для клиентов, другая для заказов. Давайте немного попрактикуемся, создавая эти две таблицы.

Какую информацию мы будем хранить? Решаем первый вопрос.

Для начала мы определимся какую информацию о **заказах** и о **клиентах** мы будем хранить. Чтобы это сделать мы должны задать себе вопрос: **“Какие единичные блоки информации относятся к клиентам, а какие единичные блоки информации относятся к заказам?”**

Проектируем таблицу клиентов.

Заказы действительно принадлежат клиентам, но заказ – это это не **минимальный блок информации**, который относится к клиентам (т.е. этот блок можно разбить на более мелкие: дата заказа, адрес доставки заказа и пр., к примеру).
Поля ниже – это минимальные блоки информации, которые относятся к клиентам:

- customer_id (primary key) – идентификатор клиента
- first_name — имя
- last_name — отчество
- address — адрес
- zip_code – почтовый индекс
- country — страна
- birth_date – дата рождения
- username – регистрационное имя пользователя (логин)
- password – пароль

Давайте перейдем к непосредственному созданию этой таблицы в SQLyog (естественно, что вы можете использовать любую другую программу). Ниже приведен пример того, как могла бы выглядеть таблица в программе SQLyog после создания. Все графические приложения для управления базами данных имеют приблизительно одинаковую структуру интерфейса. Вы также можете создать таблицу с помощью командной строки без использования графической утилиты.

Field Name	Datatype	Len	Default	PK?	Not Null?	Unsigned?	Auto Incr?
customer_id	int	11		☑	☑	☐	☑
first_name	varchar	30		☐	☑	☐	☐
last_name	varchar	30		☐	☑	☐	☐
address	varchar	50		☐	☐	☐	☐
zip_code	varchar	10		☐	☐	☐	☐
country	char	2		☐	☐	☐	☐
birthdate	date			☐	☐	☐	☐
username	varchar	15		☐	☐	☐	☐
password	varchar	35		☐	☐	☐	☐
				☐	☐	☐	☐

Создание таблицы в SQLyog. Обратите внимание, что выбран флажок первичного ключа (PK) для поля customer_id. Поле customer_id является первичным ключом. Также выбран флажок Auto Incr, что означает, что база данных будет автоматически подставлять уникальное числовое значение, которое, начиная с нуля, будет каждый раз увеличиваться на одну единицу.

Проектируем таблицу заказов.
Какие минимальные блоки информации, необходимые нам, относятся к заказу?

- order_id (primary key) – идентификатор заказа
- order_date – дата и время заказа
- customer – клиент, который сделал заказ

Ниже – пример таблицы в SQLyog.

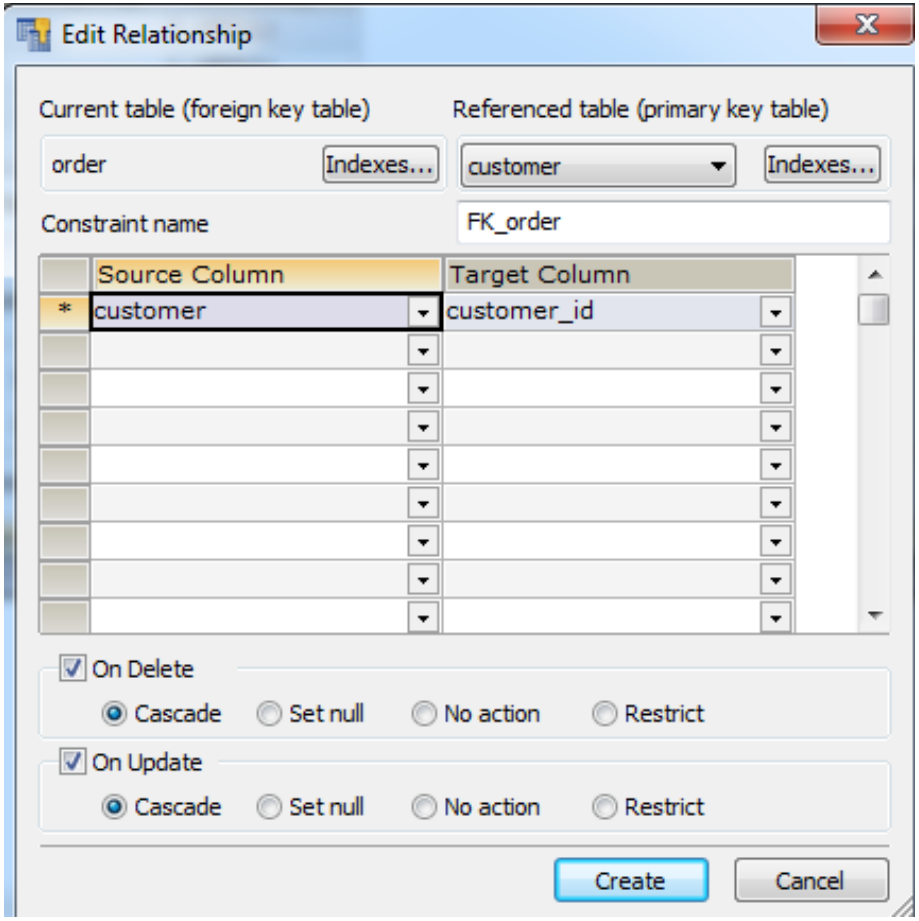
Field Name	Datatype	Len	Default	PK?	Not Null?	Unsigned?	Auto Incr?
order_id	int	10		☑	☑	☐	☑
order_date	datetime			☐	☑	☐	☐
customer	int	10		☐	☑	☐	☐
				☐	☐	☐	☐

Проект таблицы. Поле customer является ссылкой (внешним ключом) для поля customer_id в таблице клиентов.

Эти две таблицы (**клиентов** и **заказов**) связаны потому, что поле **customer** в таблице заказов ссылается на первичный ключ (**customer_id**) таблицы клиентов. Такая связь называется **связью по внешнему ключу**. Вы должны представлять себе внешний ключ как простую копию (копию значения) первичного ключа другой таблицы. В нашем случае значение поля **customer_id** из таблицы **клиентов** копируется в таблицу **заказов** при вставке каждой записи. Таким образом, у нас каждый заказ привязан к клиенту. И заказов у каждого клиента может быть много, как и говорилось выше.

Создание связи по внешнему ключу.

Вы можете задаться вопросом: “Каким образом я могу убедиться или как я могу увидеть, что поле customer в таблице заказов ссылается на поле customer_id в таблице клиентов”. Ответ прост – вы не можете сделать этого потому, что я еще не показал вам как создать связь.
Ниже – окно SQLyog с окном, которое я использовал для создания связи между таблицами.



Создание связи по внешнему ключу между таблицами заказов и клиентов.

В окне выше вы можете видеть, как поле customer таблицы заказов слева связывается с первичным ключом (customer_id) таблицы клиентов справа.

Теперь, когда вы посмотрите на данные, которые могли бы быть в таблицах, вы увидите, что две таблицы связаны.

customer_id	first_name	last_name	address	zip_code
1	john	de boer	monkeystreet 33	1234er
2	mary	lucas	wallstreet 66	9283kk
3	pablo	picasso	painterstreet 28	9829jk

order_id	order_date	customer
1	2012-02-04 17:33:47	2
2	2012-02-10 17:34:01	3
3	2012-02-24 17:34:08	2
4	2012-02-02 17:34:16	2

Заказы связаны с клиентами через поле customer, которое ссылается на таблицу клиентов.

На изображении вы видите, что клиент **mary** поместила три заказа, клиент **pablo** поместил один, а клиент **john** – ни одного.
Вы можете спросить: "А **что** же именно заказали все эти люди?" Это хороший вопрос. Вы возможно ожидали увидеть заказанные товары в таблице заказов. Но это плохой пример проектирования. Как бы вы поместили множественные продукты в единственную запись? **Товары** – это отдельные сущности, которые должны храниться в отдельной таблице. И связь между таблицами **заказов** и **товаров** будет являться связью один-ко-многим. Я расскажу об этом далее.

6. СОЗДАНИЕ ДИАГРАММЫ СУЩНОСТЬ-СВЯЗЬ

Ранее вы узнали как записи из разных таблиц связываются друг с другом в реляционных базах данных. Перед созданием и связыванием таблиц важно, чтобы вы подумали о **сущностях**, которые существуют в вашей системе (для которой вы создаете базу данных) и решили каким образом эти сущности бы **связывались** друг с другом. В проектировании баз данных сущности и их отношения обычно предоставляются в **диаграмме сущность-связь (англ. entity-relationship diagram, ERD)**. Данная диаграмма является результатом

процесса проектирования базы данных.

Сущности.

Вы можете задаться вопросом, что же такое сущность. Нуу… это “вещь” в системе. Там. Моя Мама всегда хотела, чтобы я стал учителем потому, что я очень хорошо объясняю различные вещи.

В контексте **проектирования баз данных** сущность – это нечто, что **заслуживает** своей собственной таблицы в модели вашей базы данных. Когда вы проектируете базу данных, вы должны определить эти **сущности** в системе, для которой вы создаете базу данных. Это скорее вопрос диалога с клиентом или с собой с целью выяснения того, с какими данными будет работать ваша система.

Давайте возьмем интернет-магазин для примера. Интернет-магазин продает **товары**. **Товар** мог бы стать очевидной сущностью в системе интернет-магазина. Товары **заказываются клиентами**. Вот мы с вами и увидели еще две очевидных сущности: **заказы** и **клиенты**.

Заказ оплачивается клиентом… это интересно. Мы собираемся создавать отдельную таблицу для платежей в базе данных нашего интернет-магазина? Возможно. Но разве платежи – это минимальный блок информации, который относится к заказам? Это тоже возможно.

Если вы не уверены, то просто подумайте о том, какую информацию о платежах вы хотите хранить. Возможно, вы захотите хранить **метод платежа** или **дату платежа**. Но это все еще минимальные блоки информации, которые могли бы относиться к **заказу**. Можно изменить формулировки. Метод платежа — метод платежа заказа. Дата платежа – дата платежа заказа. Таким образом, я не вижу необходимости выносить **платежи** в отдельную таблицу, хотя концептуально вы бы могли выделить платежи как сущность,

т.к. вы могли бы рассматривать платежи как контейнер информации (метод платежа, дата платежа).

Давайте не будет слишком академичными.

Как вы видите, есть разница между сущностью и непосредственно таблицей в базе данных, т.е. это не одно и то же. Специалисты отрасли информационных технологий могут быть ОЧЕНЬ академичными и педантичными в этом вопросе. Я не такой специалист. Эта разница зависит от вашей точки зрения на ваши данные, вашу информацию. Если вы смотрите на моделирование данных с точки зрения программного обеспечения, то вы можете прийти к множеству сущностей, которые нельзя будет перенести напрямую в базу данных. В данном руководстве мы смотрим на данные строго с точки зрения баз данных и в нашем маленьком мире сущность – это таблица.

Держитесь там, вы действительно близки к получению вашей ученой степени по базам данных.

Как вы видите определение того, какие сущности имеет ваша система – это немного интеллектуальный процесс, который требует некоторого опыта и часто – это предмет для внесения изменений, пересмотров, раздумий, но,

конечно, это не ракетостроение.

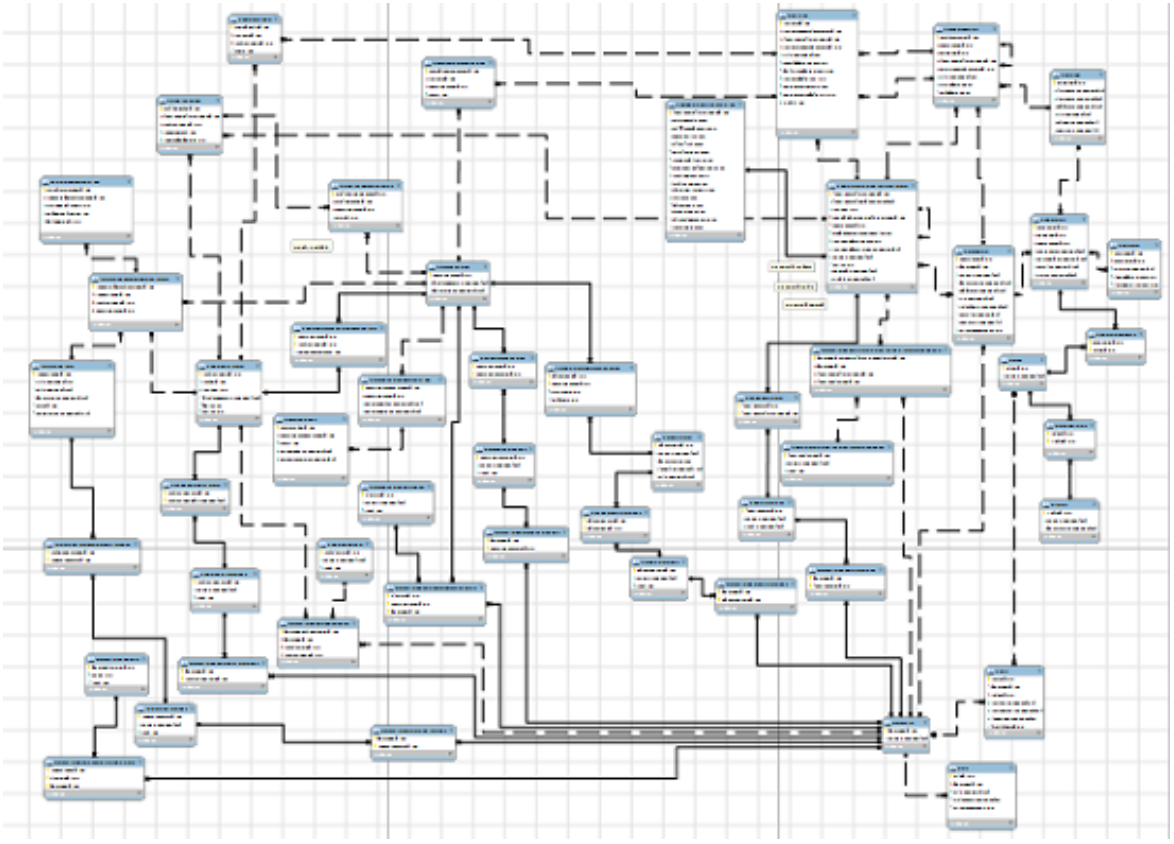


Диаграмма сущность-связь может быть достаточно большой, если вы работаете над сложным приложением. Некоторые диаграммы могут содержать сотни или даже тысячи таблиц.

Связи.

Второй шаг в проектировании баз данных – это выбор того, какие связи существуют между сущностями в вашей системе. Сейчас это может быть немного сложно для понимания, но, повторюсь еще раз, это не ракетостроение. С приобретением некоторого опыта и переосмысления выполненной работы вы будете завершать очередную модель базы данных верным или почти верным образом.

Итак. Я рассказал вам о связи **один-ко-многим** и я расскажу вам больше о связях в дальнейших частях этого руководства, поэтому сейчас я больше не буду останавливаться на этом. Просто запомните, что решение о том, какие связи будут иметь ваши сущности – важная часть **проектирования баз данных** и эти связи отображаются в диаграмме **сущность-связь**.

mysql, sql, проектирование баз данных

Похожие публикации

SQLdep поможет обуздать базы данных 30 ноября 2014 в 23:52

Проектирование баз данных. Дизайн и метод 14 апреля 2014 в 12:18

Как ответить запросом на запрос, или Базы данных не для чайников 2 октября 2013 в 15:22

Хранение деревьев в базе данных. Часть первая, теоретическая 10 сентября 2013 в 16:27

NoSQL базы данных: понимаем суть 27 сентября 2012 в 12:16

Миграция базы данных в Zend Framework: Akrabat_Db_Schema_Manager 1 августа 2012 в 10:48

Creative Commons и базы данных 2 апреля 2012 в 00:15

Бесплатный хостинг реляционных баз данных для скриптов Node.js 16 марта 2012 в 11:55

Базы данных с приватной информацией в легальной продаже 20 июня 2011 в 17:06

Версионная миграция структуры базы данных: основные подходы 14 июня 2011 в 06:55

Комментарии (7)

 **hbuser** 11 сентября 2013 в 02:08 # +1

Страсти-мордасти. Пока не совсем разобрался как пользоваться сервисом. Старый пост почему-то затерся новым… хотя у меня в постах присутствуют оба…

 **aamuvirkku** 11 сентября 2013 в 21:39 # +1

>В окне выше вы можете видеть, как поле customer таблицы заказов слева связывается с первичным ключом (customer_id) таблицы клиентов справа.

Но ни слова не сказано про каскадное удаление и какие есть альтернативы.

 **GlukKazan** 12 сентября 2013 в 09:14 # 0

> Возможно, вы захотите хранить метод платежа или дату платежа. Но это все еще минимальные блоки информации, которые могли бы относиться к заказу. Можно изменить формулировки. Метод платежа — метод платежа заказа. Дата платежа – дата платежа заказа. Таким образом, я не вижу необходимости выносить платежи в отдельную таблицу

А потом, когда БД уже основательно заполнится, внезапно выясняется, что, например, оплата заказа может производиться частями, после чего, начинаются индийские народные танцы вокруг изначально кривой схемы.

Я же говорю, вредная статья.

 **LimeOrange** 23 сентября 2013 в 09:23 # ↑ 0

Эмм… может я чего-то не понимаю, но почему сразу танцы? Делаем отдельную таблицу «payments», заполняем её данными из orders, деплоим новую версию кода, которая использует «payments», домигрируем то, что успело насоздаваться в orders за время деплоя, сносим старые поля в orders. Всё без особого геморроя, и даже с zero downtime, скорее всего.

 **GlukKazan** 23 сентября 2013 в 13:07 # ↑ 0

Все зависит от объемов данных. В любом случае, легче сразу правильно спроектировать, а не вводить совершенно искусственное ограничение 1 заказ = 1 платеж

LimeOrange 23 сентября 2013 в 13:17 # ↑ 0

«Знал бы заранее где упаду, соломки подстелил бы.»

Невозможно сразу учесть **всё**. И, опять же, из попыток учесть ВСЁ иногда рождаются монстры, ярко иллюстрирующие пословицу *«из пушки по воробьям».* Мне в этом плане очень нравится философия 37signals — «Do less». «Делай только то, что необходимо». Конечно, надо всегда помнить что потенциально какую-то часть приложения надо будет расширить или отрефакторить. Но когда клиент просит сделать ему рогатку — не стоит строить экзоскелет с встроенным гранатометом и с заделом под терминатора на будущее.

GlukKazan 23 сентября 2013 в 13:50 *(комментарий был изменён)* # ↑ 0

Речь не о том чтобы предусмотреть все, а как раз о том, чтобы не делать лишних предположений, таких как «я не вижу необходимости выносить платежи в отдельную таблицу». Лично я не вижу необходимости НЕ выносить платежи в отдельную таблицу, если понятно о чем я.

На мой взгляд, цикл этих статей характеризуется тем, что автор сознательно старается «понизить планку» порога вхождения, стараясь объяснить максимально просто, порой, достаточно сложные вещи. Мне не нравится такой популизм. Честно говоря, я с ужасом ожидаю объяснения транзакций.

Если оно вообще будет, конечно

Яндекс.Метро следит за тобой

Генератор своими руками на 220 вольт. Теперь отключения света не страшны

VNC-рулетка. Srsly?

Brainstorage

Требуется ведущий программист asp

Project manager в Веб Студию

Ведущий HTML-Верстальщик

PHP Developer

Веб-программист (для CMS Joomla)

.NET разработчик

Веб-программист

Platform engineer

Андроид разработчик Рокетбанка

JavaScript developer

все вакансии

ФРИЛАНСИМ

Нужно разработать мобильное приложение на Xamarin

Нарисовать закрепленный пост для сайта storytut.ru

Разработать сайт с калькулятором на стороне клиента на...

Нужен SMM для проекта (5 проектов)

Прикладной протокол поверх udp, клиент и сервер

Разработать проект на Python/Django

Оптимизировать сайт. Внести исправления, уменьшить в...

Сайт - биржа предметов Steam

Верстка сайта на Bitrix

Дизайн портфолио для программиста

все заказы