14 сентября 2013 в 15:43

Руководство по проектированию реляционных баз данных (10-13 часть из 15) [перевод]

перевод

SQL*, MySQL*

Продолжение.
Предыдущие части: 1-3, 4-6, 7-9

10. Нормализация баз данных

Указания для правильного проектирования **реляционных баз данных** изложены в реляционной модели данных. Они собраны в 5 групп, которые называются **нормальными формами**. Первая нормальная форма представляет самый низкий уровень нормализации баз данных. Пятый уровень представляет высший уровень нормализации.

Нормальные формы – это *рекомендации* по проектированию баз данных. Вы не обязаны придерживаться всех пяти нормальных форм при проектировании баз данных. Тем не менее, рекомендуется нормализовать базу данных в некоторой степени потому, что этот процесс имеет ряд существенных преимуществ с точки зрения эффективности и удобства обращения с вашей базой данных.

- В нормализованной структуре базы данных вы можете производить сложные выборки данных относительно простыми SQL-запросами.
- **Целостность данных**. Нормализованная база данных позволяет надежно хранить данные.
- Нормализация **предотвращает появление избыточности хранимых**

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

Форматирование Python-кода

Фрактальное пламя — алгоритм построения

Смайлики в доменных именах

данных. Данные всегда хранятся только в одном месте, что делает легким процесс вставки, обновления и удаления данных. Есть исключение из этого правила. Ключи, сами по себе, хранятся в нескольких местах потому, что они копируются как внешние ключи в другие таблицы.
- Масштабируемость – это возможность системы справляться с будущим ростом. Для базы данных это значит, что она должна быть способна работать быстро, когда число пользователей и объемы данных возрастают. Масштабируемость – это очень важная характеристика любой модели базы данных и для РСУБД.

Вот некоторые из основных пунктов, которые связаны с **нормализацией баз данных**:

- Упорядочивание данных в логические группы или наборы.
- Нахождение связей между наборами данных. Вы уже видели примеры связей один-ко-многим и многие-ко-многим.
- Минимизация избыточности данных.

Очень малое количество баз данных следуют всем пяти нормальным формам, предоставленным в реляционной модели данных. Обычно базы данных нормализуются до второй или третьей нормальной формы. Четвертая и пятая формы используются редко. Поэтому я ограничусь тем, чтобы рассказать вам лишь о первых трех.

11. Первая нормальная форма (1НФ)

Первая нормальная форма гласит, что таблица базы данных – это представление **сущности** вашей системы, которую вы создаете. Примеры сущностей: заказы, клиенты, заказ билетов, отель, товар и т.д. Каждая запись

в базе данных представляет один экземпляр сущности. Например, в таблице клиентов каждая запись представляет одного клиента.

Первичный ключ.

Правило: каждая таблица имеет первичный ключ, состоящий из наименьшего возможного количества полей.

Как вы знаете, первичный ключ может состоять из нескольких полей. Вы, к примеру, можете выбрать имя и фамилию в качестве первичного ключа (и надеяться, что эта комбинация будет уникальной всегда). Будет намного более хорошим выбором номер соц. Страхования в качестве первичного ключа, т.к. это единственное поле, которое уникальным образом идентифицирует человека.
Еще лучше, когда нет очевидного кандидата на звание первичного ключа, создайте **суррогатный** первичный ключ в виде числового автоинкрементного поля.

Атомарность.

Правило: поля не имеют дубликатов в каждой записи и каждое поле содержит только одно значение.

Возьмем, например, сайт коллекционеров автомобилей, на котором каждый коллекционер может зарегистрировать его автомобили. Таблица ниже хранит информацию о зарегистрированных автомобилях.

id	first_name	last_name	car1	car2	car3	car4	car5
1	paul	johnson	mitsubishi	(NULL)	(NULL)	paul	johnson
2	frank	black	subaru imp	daihatsu	(NULL)	(NULL)	(NULL)
3	robert	smith	Mercedes S	Ferrari f	Maserati	Toyota Pr	Spyker C8
4	john	bonham	Mazda 626	(NULL)	(NULL)	(NULL)	(NULL)

Горизонтальное дублирование данных – плохая практика.

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

все публикации

С таким вариантом проектирования вы можете сохранить только пять автомобилей и если у вас их менее 5, то вы тратите впустую свободное место в базе данных на хранение пустых ячеек.
Другим примером плохой практики при проектировании является хранение множественных значений в ячейке.

id	first_name	last_name	cars
1	john	bonham	Mazda 626
2	robert	smith	Mercedes SL450,Ferrari f40, Maserati...
3	frank	black	subaru impreza, daihatsu cuore
4	paul	johnson	mitsubishi lancer

Множественные значения в одной ячейке.

Верным решением в данном случае будет выделение автомобилей в отдельную таблицу и использование внешнего ключа, который ссылается на эту таблицу.

Порядок записей не должен иметь значение.

Правило: порядок записей таблицы не должен иметь значения.

Вы можете быть склонны использовать порядок записей в таблице клиентов для определения того, какой из клиентов зарегистрировался первым. Для этих целей вам лучше создать поля даты и времени регистрации клиентов. Порядок записей будет неизбежно меняться, когда клиенты будут удаляться, изменяться или добавляться. Вот почему вам никогда не следует полагаться на порядок записей в таблице.

В следующей части рассмотрим вторую нормальную форму (2НФ).

12. Вторая нормальная форма.

Вопросы по теме

Как при использовании EAV сделать добавление полей по категориям?

Как правильно составить SQL запрос?

Как задать каскадное удаление записей в MySQL?

Как построить такой запрос?

Посоветуйсте веб-интерфейс для выполнения заранее сохраненных MySQL-запросов с параметрами?

Что делать в данном случае?

Как составить такой запрос?

Рекомендуется ли делать такую выборку из базы?

Как добавить внешний ключ в таблицу на колонку с null с уже существующими записями?

Как расширить DISTINCTIVE?

Можно ли сделать такой запрос?

Mysql Как удалить от символа до символа?

MYSQL сложная выборка многое ко многим, как организовать?

Что почитать про нормализацию БД?

Как составить правильно SQL запрос?

Почему не работает SQL запрос?

Как ускорить выполнения SQL запросов?

Как правильно сформировать SQL запрос?

Насколько "живучие" БД?

Для того, чтобы база данных была нормализована согласно второй нормальной форме, она должна быть нормализована согласно первой нормальной форме. Вторая нормальная форма связана с избыточностью данных.

Избыточность данных.

Правило: поля с не первичным ключом не должны быть зависимы от первичного ключа.

Может звучать немного заумно. А означает это то, что вы должны хранить в таблице только данные, которые напрямую связаны с ней и не имеют отношения к другой сущности. Следование второй нормальной форме – это вопрос нахождения данных, которые часто дублируются в записях таблицы и которые могут принадлежать другой сущности.

car_id	brand	type	color	store	price
1	Maserati	Quattropor	black	Amsterdam South	203000
2	Lada	1118	yellow	Amsterdam South	150
3	Volkswagen	Golf	green	Amsterdam North	14800
4	Volkswagen	Polo	black	Amsterdam West	10200
5	Jaguar	e type	green	The Hague	82399
6	Jaguar	e type	blue	The Hague	22374

Дублирование данных среди записей в поле store.

Таблица выше может принадлежать компании, которая продает автомобили и имеет несколько магазинов в Нидерландах.

Если посмотрите на эту таблицу, то вы увидите множественные примеры дублирования данных среди записей. Поле **brand** могло бы быть выделено в отдельную таблицу. Также, как и поле **type** (модель), которое также могло бы быть выделено в отдельную таблицу, которая бы имела связь многие-к-одному с таблицей **brand** потому, что у бренда могут быть разные модели.

Как сформировать sql запрос?

Что обсуждают?

MVC и Модель 2. Знания и обязанности компонентов 17

Как мы домены мониторить начали и что из этого получилось 2

Фрактальное пламя — алгоритм построения 8

Правоохранительные органы обрушили ботнет Ramnit 17

Странности реализации Wi-Fi в метро Москвы 31

Ещё один программный UART на ATtiny13 10

Роутер от оператора? – Нет, спасибо! 87

Честные приватные свойства в прототипе 29

WEB Server на базе ENC28j60 + Arduino — проще не бывает 47

Генерация текстур планет как в игре Star Control 2 7

все публикации

Компания дня

Zfort Group

Колонка **store** содержит наименование магазина, в котором в настоящее время находится машина. **Store** – это очевидный пример избыточности данных и хороший кандидат для отдельной сущности, которая должна быть связана с таблицей автомобилей **связью по внешнему ключу**.
Ниже пример того, как бы вы моги смоделировать базу данных для автомобилей, избегая избыточности данных.

car_id	type	color	store	price
1	5	black	1	203000
2	2	yellow	1	150
3	3	green	3	14800
4	4	black	2	10200
5	1	green	4	82399
6	1	blue	4	22374

type_id	brand_id	name
1	3	e type
2	2	1118
3	4	Golf
4	4	Polo
5	1	Quattroporte s

brand_id	name	country_of_origin
1	Maserati	Italy
2	Lada	Russia
3	Jaguar	United Kingdom
4	Volkswagen	Germany

store_id	name	street	hou...	zip...	phone
1	Amsterdam South	Churchil	14	1079HA	020373
2	Amsterdam West	Mercator	27	1056 RT	020838
3	Amsterdam North	Buikslote	76	1031 AB	020387
4	The Hague	Neherstre	82	2491JJ	070387

В примере выше таблица **car** имеет внешний ключ – ссылку на таблицы **type**

и **store**. Столбец brand исчез потому, что на бренд есть неявная ссылка через таблицу **type**. Когда есть ссылка на type, есть ссылка и на brand, т.к. type принадлежит brand.

Избыточность данных была существенным образом устранена из нашей модели базы данных. Если вы достаточно придирчивы, то вы, возможно, еще не удовлетворены этим решением. А как насчет поля **country_of_origin** в таблице **brand**? **Пока** дубликатов нет потому, что есть только четыре бренда из разных стран. Внимательный разработчик базы данных должен выделить названия стран в отдельную таблицу **country**.

И даже сейчас вы не должны быть удовлетворены результатом потому, что вы также могли бы выделить поле **color** в отдельную таблицу.

Насколько строго вы подходите к созданию ваших таблиц – решать вам и зависит от конкретной ситуации. Если вы планируете хранить огромное количество единиц автомобилей в системе и вы хотите иметь возможность производить поиск по цвету (color), то было бы мудрым решением выделить цвета в отдельную таблицу так, чтобы они не дублировались.

Существует другой случай, когда вы можете захотеть выделить цвета в отдельную таблицу. Если вы хотите позволить работникам компании вносить данные о новых автомобилях вы захотите, чтобы они имели возможно **выбирать** цвет машины из заранее заданного списка. В этом случае вы захотите хранить все возможные цвета в вашей базе данных. Даже **если еще нет машин с таким цветом**, вы захотите, чтобы эти цвета присутствовали в базе данных, чтобы работники могли их выбирать. Это определенно тот случай, когда вам нужно выделить цвета в отдельную таблицу.

13. Третья нормальная форма.

Третья нормальная форма связана с **транзитивными зависимостями**.

Транзитивные зависимости между полями базы данных существует тогда, когда значения не ключевых полей зависят от значений других не ключевых полей. Чтобы база данных была в третьей нормальной форме, она должна быть во второй нормальной форме.

Транзитивные зависимости.

Правило: не может быть транзитивных зависимостей между полями в таблице.
Таблица клиентов (мои клиенты – игроки немецкой и французской футбольной команды) ниже содержит транзитивные зависимости.

client_id	first_name	last_name	province	city	postal_code
23	Khalid	Boulahrouz	Noord-Holland	Alkmaar	1825HH
24	ZinÉdine	Zidane	Noord-Holland	Langedijk	1834DK
25	Ruud	van Nistelrooy	Noord-Holland	Schermer	1844JJ
19	Phillip	Cocu	Noord-Holland	Heilo	1850WI

В этой таблице не все поля зависят исключительно от первичного ключа. Существует отдельная связь между полем postal_code и полями города (city) и провинции (province). В Нидерландах оба значение: город и провинция – определяются почтовым кодом, индексом. Таким образом, нет необходимости хранить город и провинцию в клиентской таблице. Если вы знаете почтовый код, то вы уже знаете город и провинцию.

Такая транзитивной зависимости следует избегать, если вы хотите, чтобы ваша модель базы данных была в третьей нормальной форме.

В данном случае устранение транзитивной зависимости из таблицы может быть достигнуто путем удаления полей города и провинции из таблицы и хранение их в отдельной таблице, содержащей почтовый код (первичный ключ), имя провинции и имя города. Получение комбинации почтовый код-город-провинция для целой страны может быть весьма нетривиальным занятием. Вот почему такие таблицы зачастую продаются.

Другим примером для применения третьей нормальной формы может служить (слишком) простой пример таблицы заказов интернет-магазина ниже.

order_number	total_ex_vat	total_inc_vat
23	13,44	16,00
24	34,70	41,30
25	543,44	645,00
19	34,50	41,06

НДС (value added tax) – это процент, который добавляется к цене продукта (19% в данной таблице). Это означает, что значение total_ex_vat может быть вычислено из значения total_inc_vat и vice versa. Вы должны хранить в таблице одно из этих значений, но не оба сразу. Вы должны возложить задачу вычисления total_inc_vat из total_ex_vat или наоборот на программу, которая использует базу данных.

Третья нормальная форма гласит, что вы не должны хранить данные в таблице, которые могут быть получены из других (не ключевых) полей таблицы. Особенно в примере с таблицей клиентов следование третьей нормальной форме требует либо большого объема работы, либо приобретения коммерческой версии данных для такой таблицы.

Третья нормальная форма не всегда используется при проектировании баз данных. Когда разрабатываете базу данных вы всегда должны сравнивать преимущества от более высокой нормальной формы в сравнении с объемом работ, которые требуются для применения третьей нормальной формы и поддержания данных в таком состоянии. В случае с клиентской таблицей лично я бы предпочел не нормализовать таблицу до третьей нормальной формы. В последнем примере с НДС я бы использовал третью нормальную форму. Хранение данных, **воспроизводимых из существующих**, обычно плохая идея.

sql, mysql, проектировние баз данных

Похожие публикации

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

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

Реляционные базы данных обречены? 28 января 2011 в 00:51

Комментарии (14)

Megaprog 14 сентября 2013 в 19:52 # 0

Правило 2-й нормльной формы здесь совершенно не понятно, либо перевод такой.

hbuser 14 сентября 2013 в 20:04 # 0

Само правило — да, для новичка. Пример чуть упрощает понимание.

> **Win32Sector** 14 сентября 2013 в 20:29 # ↑ 0
>
> Вот подробно про 2НФ
> habrahabr.ru/post/113260/

Win32Sector 14 сентября 2013 в 20:29 # 0

Опять же спасибо за перевод.

vovik99 14 сентября 2013 в 21:13 # +1

Про 2НФ написано нечто, абсолютно к ней не относящееся. Вторая нормальная форма — это когда нет полей, зависящих только от части составного ПК.

Не знаю уж, оригинал это такой или последствия перевода. Но в любом случае, если бы переводчик был в теме — такого появиться не должно было бы. Переводить материал про НФ, не разбираюсь в них — сложно ожидать хорошего результата.

> **dimarick** 15 сентября 2013 в 15:52 *(комментарий был изменён)* # ↑ 0
>
> Не согласен. 2НФ — данные в таблице зависят только от первичного ключа. Ваше замечание — частный случай 2НФ. У автора определено более обобщенно.

> > **vovik99** 15 сентября 2013 в 21:26 # ↑ +1

У автора — определенно ерунда.

Да и Вам было бы полезно найти четкое определение 2НФ, понять его, потом уже комментировать. Без обид, но тут нет ничего, с чем можно соглашаться или нет. Все очевидно.

dimarick 15 сентября 2013 в 21:56 # ↑ 0

«Отношение находится во 2НФ, если оно находится в 1НФ и каждый неключевой атрибут функционально полно зависит от ключа.»
Пруфы:
www.intuit.ru/studies/courses/3439/681/lecture/8482
ru.wikipedia.org/wiki/Вторая_нормальная_форма
www.mstu.edu.ru/study/materials/zelenkov/ch_4_2.html
citforum.ru/database/dbguide/4-5.shtml

К автору можно придраться только с точки зрения русского языка, но суть определения передана верно.
В вашем определении не описан случай не составного ПК и случай когда поле не зависит ни от какой части ПК.

vovik99 15 сентября 2013 в 22:15 # ↑ +1

Определение Вы нашли, но, похоже, не потрудились все же его понять.

Во всех перечисленных пруфах при определении 2НФ явно идет речь о составном ключе. Если первичный ключ состоит из одного поля (и таблица в 1НФ), то она уже автоматически находится в 2НФ.

Если поле не зависит ни от какой части ПК, то это уже не называется отношением, не является реляционной моделью данных и никакие нормальные формы тут не применимы.

dimarick 15 сентября 2013 в 23:29 # ↑ 0

> Определение Вы нашли, но, похоже, не потрудились все же его

> понять.

Потрудитесь и вы.
Полная функциональная зависимость — неключевой атрибут функционально полно зависит от составного ключа если он функционально зависит от всего ключа в целом, но не находится в функциональной зависимости от какого-либо из входящих в него атрибутов.
Отношение находится во 2НФ, если оно находится в 1НФ и каждый неключевой атрибут функционально полно зависит от ключа.

Ваше определение нельзя назвать не верным, но полным его тоже назвать нельзя.

В вашем определении упущен важный момент — поле должно зависеть от ключа в целом и только от него. Кроме того для применения 2НФ ключ не обязан быть составным.

> Если поле не зависит ни от какой части ПК, то это уже не называется отношением

Поле может зависеть от другого поля(«страна» может зависеть от поля «город» в таблице магазинов), не являющегося частью ключа. Этот случай полностью соответствует вашему определению, но тем не менее не является 2НФ.

vovik99 16 сентября 2013 в 00:07 # ↑ **+1**

> Я вообще-то никаких определений не давал. Только описал максимально краткую суть 2НФ, чтобы показать некомпетентность статьи автора. Строгое определение, естественно, гораздо подробнее и приведено в вышеупомянутых пруфах.
>
> Страна, зависящая от города в таблице магазинов, является транзитивной зависимостью страны от первичного ключа. При этом поле от первичного ключа, тем не менее, зависит. В таком виде

таблица не соответствует 3НФ, но находится в полном согласии с 2НФ.

Не готов здесь проводить ликбез по нормальным формам, тут нужна отдельная хорошая статья. Но исходный материал точно таким не является :)

dimarick 16 сентября 2013 в 01:03 # ↑ 0

Вынужден полностью согласиться.
Не частый случай когда теория сложнее чем практика.
Еще хороший материал по теме:
www.libma.ru/kompyutery_i_internet/bazy_dannyh_konspekt_lekcii/p10

TIEugene 15 сентября 2013 в 00:17 *(комментарий был изменён)* # 0

Я бы это вывесил на gnivc — но нет технической возможности
(FoxPro позабыл слегка потому что)

dimarick 15 сентября 2013 в 15:40 # +1

Хочу отметить что все три нормальные формы — это следование принципу DRY. И наоборот: следуя принципу DRY, ваша бд будет нормализованой, даже если вы специально не думаете об этом.

> Хранение данных, воспроизводимых из существующих, обычно плохая идея.

Это либо плохая идея, либо денормализация.

Только зарегистрированные пользователи могут оставлять комментарии. Войдите, пожалуйста.

Ускоряем понимание коммерческого или технического текста: как перестать бояться писать просто

Снимаем образы с картриджей для Dendy/Famicom/NES

Почему 1С это плохо и почему так не любят 1С программистов

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