11 сентября 2013 в 17:53

Руководство по проектированию реляционных баз данных (7-9 часть из 15) [перевод]

перевод

SQL*, MySQL*

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

7. Связь один-ко-многим.

Я уже показал вам как данные из разных таблиц могут быть связаны при помощи **связи по внешнему ключу**. Вы видели как заказы связываются с клиентами путем помещения customer_id в качестве внешнего ключа в таблице заказов.

Другой пример связи один-ко-многим – это связь, которая существует между матерью и ее детьми. Мать может иметь множество детей, но каждый ребенок может иметь только одну мать.

(Технически лучше говорить о женщине и ее детях вместо матери и ее детях потому, что, в контексте связи один-ко-многим, мать может иметь 0, 1 или множество потомков, но мать с 0 детей не может считаться матерью. Но давайте закроем на это глаза, хорошо?)

Когда одна запись в таблице A может быть связана с 0, 1 или множеством записей в таблице B, вы имеете дело со связью **один-ко-многим**. В реляционной модели данных связь один-ко-многим использует две таблицы.

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

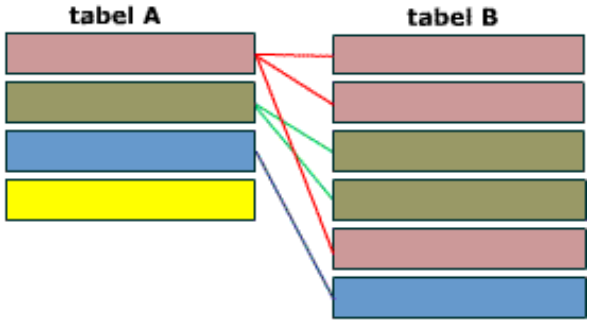
Удалённое управление для Arduino, проба пера

Ещё один программный UART на ATtiny13

Форматирование Python-кода

Фрактальное пламя — алгоритм построения

Смайлики в доменных именах



Схематическое представление связи один-ко-многим. Запись в таблице A имеет 0, 1 или множество ассоциированных ей записей в таблице B.

Как опознать связь один-ко-многим?

Если у вас есть две сущности спросите себя:
1) Сколько объектов и B могут относится к объекту A?
2) Сколько объектов из A могут относиться к объекту из B?

Если на первый вопрос ответ – **множество**, а на второй – **один** (или возможно, что ни одного), то вы имеете дело со связью один-ко-многим.

Примеры.

Некоторые примеры связи один-ко-многим:

- Машина и ее части. Каждая часть машины единовременно принадлежит только одной машине, но машина может иметь множество частей.
- Кинотеатры и экраны. В одном кинотеатре может быть множество экранов, но каждый экран принадлежит только одному кинотеатру.
- Диаграмма сущность-связь и ее таблицы. Диаграмма может иметь больше, чем одну таблицу, но каждая из этих таблиц принадлежит только одной диаграмме.
- Дома и улицы. На улице может быть несколько домов, но каждый дом принадлежит только одной улице.

Компанию Lenovo атаковали в отместку за шпионскую программу Superfish

«Hero Image» — баннеры в параллаксе

Правоохранительные органы обрушили ботнет Ramnit

OpenCage — самый мощный инструмент для геокодирования

все лучшие

Лучшее на Geektimes

ИИ от Google самостоятельно освоил 49 старых игр Atari

Как я имплантировал RFID себе в руку, а потом еще NFC. Часть 1

Krita 2.9: релиз, осуществленный благодаря Kickstarter

Вышел Unreal Engine 4.7 с поддержкой HTML5 и WebGL

Заря электромобилей: XIX век

Премьера фильма «Вселенная Стивена Хокинга» в РФ

В Великобритании запретили «сексуально оскорбительную» рекламу смартфона

Gemalto отвергла обвинения в массовой краже ключей доступа к SIM-картам

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

Цифровой аудиоформат 24/192, и почему в

В данном случае все настолько просто, что только поэтому может оказаться трудным понимание. Возьмем последний пример с домами. На улице ведь действительно может быть любое количество домов, но у каждого дома именно на этой улице может быть только одна улица (не берем дома, которые на практике принадлежат разным улицам, возьмем, к примеру, дом в центре улицы). Ведь не может конкретно этот дом быть одновременно в двух местах, на двух разных улицах, а мы говорим не про какой-то абстрактный дом вообще, а про конкретный.

8. Связь многие-ко-многим.

Связь многие-ко-многим – это связь, при которой множественным записям из одной таблицы (A) могут соответствовать множественные записи из другой (B). Примером такой связи может служить школа, где учителя обучают учащихся. В большинстве школ каждый учитель обучает многих учащихся, а каждый учащийся может обучаться несколькими учителями.

Связь между поставщиком пива и пивом, которое они поставляют – это тоже связь многие-ко-многим. Поставщик, во многих случаях, предоставляет более одного вида пива, а каждый вид пива может быть предоставлен множеством поставщиков.

Обратите внимание, что при проектировании базы данных вы должны спросить себя не о том, существуют ли определенные связи в данный момент, а о том, возможно ли существование связей вообще, в перспективе. Если в настоящий момент все поставщики предоставляют множество видов пива, но каждый вид пива предоставляется только одним поставщиком, то вы можете подумать, что это связь один-ко-многим, но… Не торопитесь реализовывать связь один-ко-многим в этой ситуации. Существует высокая вероятность того, что в будущем два или более поставщиков будут поставлять один и тот

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

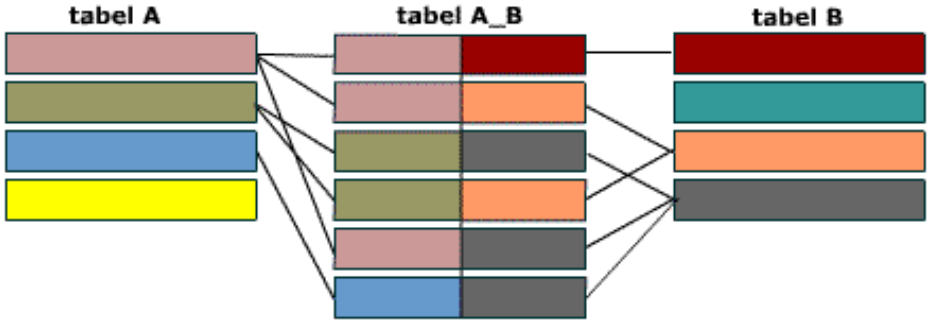
5 шагов к созданию привлекательного сервиса для приложения

все публикации

же вид пива и когда это случится ваша база данных — со связью один-ко-многим между поставщиками и видами пива – не будет подготовлена к этому.

Создание связи многие-ко-многим.

Связь многие-ко-многим создается с помощью трех таблиц. Две таблицы – “источника” и одна соединительная таблица. Первичный ключ соединительной таблицы A_B – **составной**. Она состоит из двух полей, двух внешних ключей, которые ссылаются на первичные ключи таблиц A и B.



Все первичные ключи должны быть уникальными. Это подразумевает и то, что комбинация полей A и B должна быть уникальной в таблице A_B.

Пример проект базы данных ниже демонстрирует вам таблицы, которые могли бы существовать в связи многие-ко-многим между бельгийскими брендами пива и их поставщиками в Нидерландах. Обратите внимание, что все комбинации beer_id и distributor_id уникальны в соединительной таблице.

Таблицы “о пиве”.

Вопросы по теме

- Как организовать хранение пользователей в базе данных с разными полями?
- Как организовать структуру таблиц в БД?
- Есть ли способ отложенно обновлять записи в БД?
- MySql какой тип данных выбрать для url ?
- Как правильно спроектировать БД?
- Внешний ключ на несколько таблиц
- Каталог с сортировкой: как решить проблему при выводе с сортировкой?
- Как лучше хранить большой список параметров в MySQL? Проектирование структуры
- Хранение типизированных данных в базе данных
- 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?

Table beer

beer_id	name
157	Gentse Tripel
158	Uilenspiegel
146	Duvel
123	Leffe
222	Brugse Tripel
160	Sint Bernardus Pater
163	Jupiler

Table beer_distributor

beer_id	distributor_id
157	**AC001**
157	**AB899**
157	**AC009**
158	**AC009**
163	**AC009**
160	**AB999**
163	**AB999**
222	**AB999**
123	**AB999**
146	**AB999**
158	**AB999**
157	**AB999**

Table distributor

distributor_id	name
AB999	De vrolijke drinker
AC001	Horeca Import NL
AC002	van Gent bierimport
AB899	Jansen Horeca
AC008	De bierleverancier
AC009	Petersen Drankenhandel

Таблицы выше связывают поставщиков и пиво связью **многие-ко-многим**, используя соединительную таблицу. Обратите внимание, что пиво 'Gentse Tripel' (157) поставляют Horeca Import NL (157, AC001) Jansen Horeca (157,

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?

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

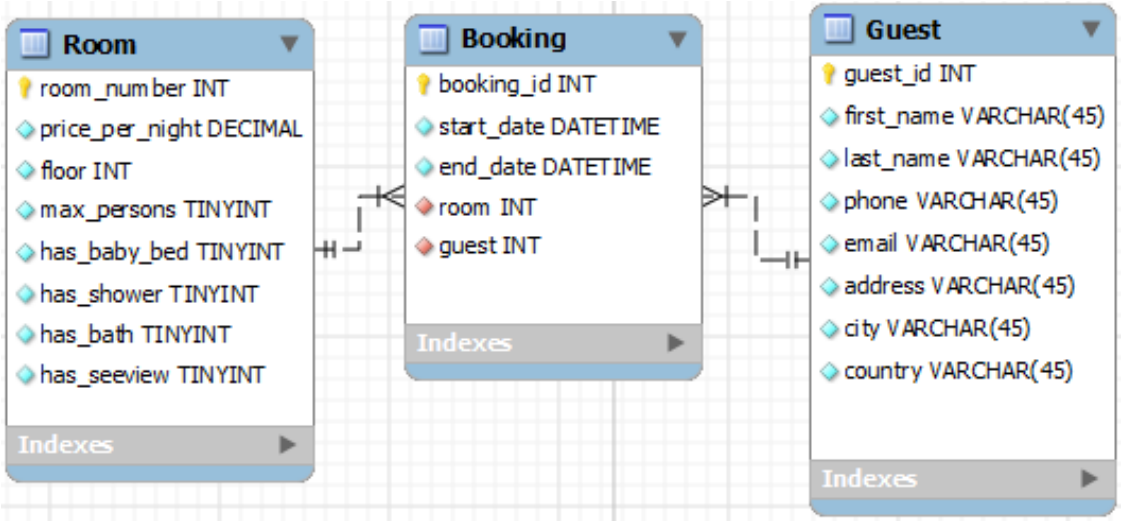
AB899) и Petersen Drankenhandel (157, AC009). И vice versa, Petersen Drankenhandel является поставщиком 3 видов пива из таблицы, а именно: Gentse Tripel (157, AC009), Uilenspiegel (158, AC009) и Jupiler (163, AC009).

Еще обратите внимание, что в таблицах выше поля первичных ключей окрашены в синий цвет и имеют подчеркивание. В модели проекта базы данных первичные ключи обычно подчеркнуты. И снова обратите внимание, что соединительная таблица beer_distributor имеет первичный ключ, составленный из двух внешних ключей. Соединительная таблица всегда имеет составной первичный ключ.

Есть еще одна важная вещь на которую нужно знать. Связь многие-ко-многим состоит из **двух связей один-ко-многим**. Обе таблицы: поставщики пива и пиво – имеют связь один-ко-многим с соединительной таблицей.

Другой пример связи многие-ко-многим: заказ билетов в отеле.

В качестве последнего примера позвольте мне показать как бы могла быть смоделирована таблица заказов номеров гостиницы посетителями.



Соединительная таблица связи многие-ко-многим имеет дополнительные

Control 2 7

все публикации

Компания дня

Zfort Group

Последняя публикация: PHP-Дайджест № 57 – интересные новости, материалы и инструменты (9 – 22 февраля 2015)

4657 подписчиков

поля.

В этом примере вы видите, что между таблицами гостей и комнат существует связь многие-ко-многим. Одна комната может быть заказана многими гостями с течением времени и с течением времени гость может заказывать многие комнаты в отеле. Соединительная таблица в данном случае является не классической соединительной таблицей, которая состоит только из двух внешних ключей. Она является отдельной сущностью, которая имеет связи с двумя другими сущностями.

Вы часто будете сталкиваться с такими ситуациями, когда совокупность двух сущностей будет являться новой сущностью.

9. Связь один-к-одному.

В связи один-к-одному каждый блок сущности A может быть ассоциирован с 0, 1 блоком сущности B. Наемный работник, например, обычно связан с одним офисом. Или пивной бренд может иметь только одну страну происхождения.

В одной таблице.

Связь один-к-одному легко моделируется в одной таблице. Записи таблицы содержат данные, которые находятся в связи один-к-одному с первичным ключом или записью.

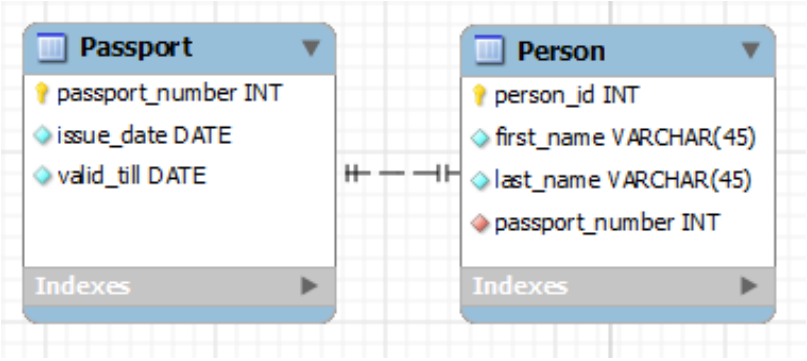
В отдельных таблицах.

В редких случаях связь один-к-одному моделируется используя две таблицы. Такой вариант иногда необходим, чтобы преодолеть ограничения РСУБД или с целью увеличения производительности (например, иногда — это вынесение поля с типом данных blob в отдельную таблицу для ускорения поиска по

родительской таблице). Или порой вы можете решить, что вы хотите разделить две сущности в разные таблицы в то время, как они все еще имеют связь один-к-одному. Но обычно наличие двух таблиц в связи один-к-одному считается дурной практикой.

Примеры связи один-к-одному.

- Люди и их паспорта. Каждый человек в стране имеет только один действующий паспорт и каждый паспорт принадлежит только одному человеку.



Проект реляционной базы данных – это коллекция таблиц, которые перелинковываются (связываются) первичными и внешними ключами. Реляционная модель данных включает в себя ряд правил, которые помогают вам создать верные связи между таблицами. Эти правила называются “нормальными формами”. В следующих частях я покажу как нормализовать вашу базу данных.

Какой же вид связи вам нужен?

Примеры связей таблиц на практике. Когда какие-то **данные являются уникальными для конкретного объекта**, например, человек и номера его паспортов, то имеем дело со **связью один-ко-многим**. Т.е. в одной таблице

мы имеем список неких людей, а в другой таблице у нас есть перечисление номеров паспортов этого человека (напр., паспорт страны проживания и загранпаспорт). И эта комбинация данных уникальная для каждого человека. Т.е. у каждого человека может быть несколько номеров паспортов, но у каждого паспорта может быть только один владелец. Итого: **нужны две таблицы.**

А если есть некие **данные, которые могу быть присвоены любому человеку**, то имеем дело со **связью многие-ко-многим**. Например, есть таблица со списком людей и мы хотим хранить информацию о том, какие страны посетил каждый человек. В данном случае имеется две сущности: люди и страны. Любой человек может посетить любое количество стран равно, как и любая страна может быть посещена любым человеком. Т.е., в данном случае, страна не является уникальными данными для конкретного человека и может использоваться повторно.

В таких случаях использование связи **многие-ко-многим с использованием трех таблиц** и с хранением общей информации централизованно очень удобно. Ведь если общие данные меняются, то для того, чтобы информация в базе данных соответствовала действительности достаточно подправить ее только в одном месте, т.к. хранится она только в одном месте (таблице), в остальных таблицах имеются лишь ссылки на нее.

А когда у вас есть набор уникальных данных, которые имеют отношение только друг к другу, то храните все в одной таблице. Ваш выбор – связь один-к-одному. Например, у вас есть небольшая коллекция автомобилей и вы хотите хранить информацию о них (цвет, марка, год выпуска и пр.).

sql, mysql, 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 (27)

sdevalex 11 сентября 2013 в 19:56 # **–2**

Мне проектирование баз данных никогда не казалось сложным. Да, есть тонкости РСУБД и моменты связанные с производительностью, но в 95% случаях все кристально ясно. Может кто-то описать какие-либо сложные случаи? (чтобы понять, может мне всегда простые проекты попадаются).

asArtem 12 сентября 2013 в 00:12 # ↑ **+1**

нашим архитекторам тоже так не казалось. Куча таблиц (100, 200 — хз. очень много), которые благодаря джоинам покрываются локами и вешают базу, малоселективные индексы на битовые поля, вьюхи из вьюх и логика в хранимках —

этот ад не справился с нагрузкой после релиза. А проект на 60 человек и большая фин система.
Проективровать базы нужно не на бумаге
Вторая система которую поддерживаю — партнёрка для рекламы. Те же грабли. На локальной машине выборки 500 записей по 20 секунд.

 **kingu** 12 сентября 2013 в 03:15 # ↑ **+1**

А проектирование как таковое вообще было?
Судя по личным наблюдениям, большинство относится к БД как к черному ящику и вообще не хочет вникать в вопросы реализации.

Насчет сложных мест — советую посмотреть www.slideshare.net/billkarwin/sql-antipatterns-strike-back

 **wrmax** 13 сентября 2013 в 11:58 # ↑ **0**

Да уж. Представления в базе данных это очевидное зло по двум причинам:
1. Вы не понимаете как работают представление.
2. Другие разработчики не понимают как работают представления.

 **multik** 12 сентября 2013 в 14:54 # ↑ **+1**

спроектируйте структуру данных для хранения номеров телефонов, как мобильных так и домашних, имеющих возможность хранить внутри себя любой телефонный номер мира (любая страна и прочее), а также подумайте о возможных изменениях кода стран, городов, мобильных операторов, и сделайте вашу структуру таким образом, что бы при таком изменении, сам идентификатор записи телефонного номера не изменился, и что бы можно было посмотреть историю этого телефона во времени…

когда справитесь могу подкинуть вам задачку про адрес…

 **niga** 15 сентября 2013 в 04:02 # ↑ **0**

Не добавил в базу информацию о типе телефона (мобильный/домашний), но получилось что-то вроде этого:

content.screencast.com/users/nigasam/folders/Jing/media/940fcf20-7000-499c-8b2e-d35a63dc506e/00000003.png
Укажите, пожалуйста, на неточности.
Спасибо!

multik 15 сентября 2013 в 19:54 # ↑ 0

Вначале поясните немного схему: какие сущности хранят в себе таблицы locality и locality_code — это что город/название мобильного оператора? Поясните пожалуйста.

Далее объясните пожалуйста с какой целью появилась таблица region? Насколько я понял она не хранит в себе ничего кроме имени и ссылки на страну. Или вы попытались объединить задачу с телефоном и адресом? Поясните пожалуйста.

Также, ваша реализация хранения истории — перечёркивает всю красоту точечных изменений телефонных кодов: к примеру у страны поменялся телефонный код в таблице phone — всё хорошо, никаких апдейтов делать не нужно данные будут корректны, но для каждой записи из этой таблицы, которая ссылается на страну, с изменившимся телефонным кодом необходимо будет создавать новую запись в таблице phone_history — что не есть хорошо. Даже если код страны изменится 1 раз — но страна будет к примеру Россия или США — то прикиньте сколько новых записей вам придётся создать для отслеживания этой истории?

Но вначале всё-таки поясните пожалуйста назначение основных таблиц — историю обсудим позже.

niga 15 сентября 2013 в 20:10 # ↑ 0

locality — населённый пункт, сначала думал назвать city, но это не верно.
locality_code — коды номеров, которые присутствуют в населённом пункте.

region введён для понимания о каком населённом пункте идет речь. К примеру, населённый пункт Александровка присутствует на карте России 166 раз:

ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B5%D0%BA%D1%81%D0%B0%D

Остальные сущности, надеюсь, понятны.

Про историю согласен. Должна быть информация только о замене кодов. Соответственно таблица для кодов стран и кодов населённых пунктов.

multik 17 сентября 2013 в 13:44 *(комментарий был изменён)* # ↑ 0

region вводить для понимания не совсем верно, не надо смешивать базу телефонных номеров с адресной базой (населённых пунктов и прочего) — это добавит только больше путаницы. То что связь между телефонами и адресами существует это понятно, и её даже лучше будет выстроить на уровне связей между таблицами, но перед тем как выстраивать связи между двумя подсистемами надо вначале построить каждую по отдельности, и выстроить связи внутри неё.

Задача подсистемы телефонных номеров, на уровне БД, заключается в двух вещах:
1) быстро выдавать готовый телефонный номер по idPhone (полностью или частично, например без кода страны, или без кода города);
2) минимизировать количество записей при изменении (не потеряв важных данных)

Country — безусловно нужная таблица — потому что телефонный код страны — обязательная составная часть любого полного номера телефона.

locality — нормальное название, особенно если под ним мы можем понимать также и мобильного оператора к примеру Билайн или МТС — и совершенно верно что у одного оператора(в одном городе) может быть несколько телефонных кодов, однако надо помнить что один оператор может быть представлен в нескольких странах — но эти вопросы мы сознательно упускаем, и понимаем что в БД оператор Билайн — встретится и в Украине и в России и в других странах (будет хранится в нескольких записях табилицы locality) — если нам потом потребуется отдельно это анализировать и использовать — то ни что

не помешает создать таблицу «Оператор связи» — и выстроить связь между этой таблицей и таблицей locality.

Насчёт того что поле code в locality_code имееют длинну 4 символа — это вы явно поспешили, во многих мелких населённых пунктах Украины (и наверное России) до сих пор есть 5ти и 4х значные телефонные номера — то есть коды городов у них будут 5-ти и 6-ти значные.

Часто хочется конечно сразу угадать и на уровне строгой типизации сделать невозможным ошибочный ввод данных — но необходимо либо полностью собрать предварительную информацию по предметной области, либо, при невозможности этого, заложится на менее строгую типизацию.

В остальном всё верно. Получается из вашей схемы необходимо выкинуть таблицу region и phone_history и изменить длинну locality_code — если вам интересно можем дальше обсудить возможность и целесообразность реализации хранения истории изменений.

niga 17 сентября 2013 в 15:28 # ↑ 0

Я не совсем с Вами согласен по поводу region.

Данная база имеет информативность стремящуюся к нулю. Из неё мы можем узнать только информацию о смене кодов для телефона. По хорошему из неё было бы не плохо получить информацию о владельце номера. Я не говорю сейчас про адрес. В этом случае без таблицы region есть шансы получить номера для 166-ти жителей Александровки (при наличии в каждом населённом пункте однофамильцев). И это только для России.

Таблица locality не должна содержать информацию о мобильных операторах. Только населённые пункты. Для мобильных операторов необходимо создать дополнительную таблицу. И связать её с locality_code. На данный момент все коды мобильных операторов, как и городские коды, хранятся в locality_code.

По поводу длинны кода для населённого кода согласен. Самый длинный телефонный код (как минимум для России) принадлежит селу Средние Пахачи (Камчатка). Состоит из 8-ми символов: 41544513.

Немного доделал схему:
content.screencast.com/users/nigasam/folders/Jing/media/a0148b59-5da8-43be-9a09-2614bcd10341/00000004.png

Для каждого кода теперь указывается идентификатор оператора. Городской номер также должен ссылаться на эту таблицу.

Давайте продолжим. Эта тема интересна мне.

multik 17 сентября 2013 в 15:38 # ↑ 0

Вы включаете в схему таблицы адреса потому что не хотите для адреса отдельно разрабатывать схему? Вы хотите соединить две подсистемы в одной?

niga 17 сентября 2013 в 15:55 # ↑ 0

Я с радостью сделаю схему и для адресов. Отдельно. Я хочу сделать подсистему отдельно, но правильно. С возможностью дальнейшего роста.

multik 17 сентября 2013 в 16:12 # ↑ 0

а зачем тогда в эту подсистему вы включаете элементы адреса, избыточные? вдень когда я хочу позвонить кому-то мне в принципе всё-равно в какую Александровку я звоню, главное понимать какая это страна (сколько мне это будет стоить денег) и какой у Александровки телефонный код и номер абонента.

Другое дело что внутрисетевой роуминг внутри страны, и

разница в тарификации при звонке в другой регион — действительно имеет место в России, если вы для этого вводили таблицу region — тогда это ещё можно понять. Но с другой стороны — тогда эту задачу вы не решите для мобильных телефонов в той схеме (вариант 2) который вы привели.

Вы со мной в принципе не согласны, что разумнее вначале выстроить отдельно таблицы для «чистой» подсистемы телефонных номеров, а затем для адреса и только потом выстроить связи позволяющие решать в том числе и задачи выявления однофамильцев в городах (здесь потребуется ещё одна подсистема — Люди)?

multik 17 сентября 2013 в 15:41 # ↑ 0

Важно понимать что это ещё не база — это только кусок схемы относящейся к телефонным номерам, и адрес также должен быть и таблицы для него также должны быть, но если вам кажется что в вашей схеме уже есть и адрес и телефоны — то можем подискутировать на эту тему (для адреса этих таблиц недостаточно), я же хочу разбить схему и задачу на две части а потом выстроить связи между адресом и телефонами

niga 17 сентября 2013 в 16:16 # ↑ 0

Мне не кажется, что в схеме есть адрес. В схеме есть привязка телефона к населённому пункту. Населённый пункт помимо привязки к стране имеет привязку к региону/краю/району/etc, которые можно объединить в одну таблицу.

Даже если смотреть на это как на схему, то поставьте себе задачу получить код того же населённого пункта Александровка.

multik 17 сентября 2013 в 16:18 # ↑ 0

Если мы исходим из того что в схеме есть Адрес (элементы адреса конечно же в ней есть), но у меня сразу вопрос — а это весь адрес? Его будет достаточно? Или это не весь, и потом мы будем добавлять таблицы и связи и возможно изменять существующие связи?

niga 17 сентября 2013 в 16:44 # ↑ 0

В схеме нет адреса. Давайте пока вообще о нём забудем. И таблицу регионов удалим. Сделаем такую небольшую подсистему телефонных номеров. И отдадим в пользование. Потом сделаем подсистему адресов и будем связывать.

multik 0

17 сентября 2013 в 18:19 *(комментарий был изменён)* # ↑

Хорошо, тогда давайте посмотрим на любой телефонный номер и определим его составные части:

как видим он состоит из трёх частей код страны, код оператора/города/… и собственно номера.

Будем исходить из предположения что первая и вторая составная часть может изменятся, но сам номер останется прежним и айдишник у него не изменится (вопросы смены номера экстренных служб города и им подобные для простоты не рассматриваем).

Если решать задачу сразу в лоб — то первым решением будет создать три таблицы и так их и назвать «код страны», «код оператора или города» «телефонный номер». Решение не такое плохое как может показаться на первый взгляд. Но попробуем сделать ещё несколько предположений.

Предположение первое — у каждой страны в один момент времени есть только один телефонный код страны: — если верить этой статье ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81 то это действительно так.

Что нам может дать это утверждение? Оно может дать нам ровно следующее — нам не требуется создавать отдельную сущность «код страны» — и в будущем связывать её с сущностью «страна» — потому что связь между страной и кодом страны 1 к 1 — значит мы можем просто создать таблицу Страна и добавить в неё одно дополнительное поле — код страны — таблица country — полностью этому соответствует.

С первой таблицей определились. Теперь забежим вперёд и сразу подумаем насчёт истории изменений этой таблицы. Если мы хотим иметь возможность отслеживать изменения, и при необходимости вытаскивать телефонный код страны на момент какой-то даты определённым специфическим запросом — то где-то нам надо хранить эти изменения. У вас это country_code_history. Мне кажется что это не самое лучшее название, раз уж основная таблица называется country — то и историю лучше хранить в таблице country_history — потому что так нам одной таблицы хватить на хранение любых изменений одной сущности (к примеру смену названий государства также можно будет отследить по этой таблице). Какие поля должны быть в таблице истории кроме, естественно country_id? Если мы предполагаем отслеживать все изменения, то можно хранить все поля из той таблицы в нашем случае добавить также название страны. Что ещё требуется? Ещё потребуется всего два параметра дата старта и дата финиша (то есть время «с» «по» когда этот

экземпляр истории был активной текущей записью страны с определённым айдишником). Хранить old_code и new_code — мне кажется не самым лучшим решением, потому что мне даже не понятно какой код был у страны на определённую дату взглянув на вашу схему? Возможно вы закладывались на что-то иное — и я просто неправильно вас понял в таком случае поясните.

Также сразу оговорим какие изменения наша схема не отследит стандартным способом а именно это распады и соединения стран. То есть варианты с СССР(была одна страна стало много) ФРГ и ГДР (две страны слились в одну) и их телефонными кодами стран, красиво в эту схему не лягут (придётся докручивать напильником и прочее), но мы согласны на такое несовершенство, и готовы в будущем доработать требуемые структуры и механизмы при наличии таких требований.

Также важно сразу обговорить, каким образом будут происходить записи в таблицу изменений country_history — (хоть написание методов CRUD и не входит в задачу разработки схемы, но на кое-какие моменты необходимо обратить внимание) а именно — при любом апдейте записи в таблице country — старое значение должно попадать в таблицу country_history (естественно я тут не рассматриваю пустые или ошибочные изменения — отслеживание этого — отдельный вопрос, ещё мы понимаем что в нашей системе только у очень ограниченного круга лиц будут права изменить название или код страны — это не частая операция).

Если вам мои рассуждения по поводу первых двух таблиц понятны, и возражений нет и вам интересно продолжить я продолжу рассуждать далее, если я

где-то ошибся или неправильно вас понял или вы со мной не согласны, напишите пожалуйста.

niga 25 сентября 2013 в 15:45 # ↑ 0

По поводу истории страны согласен. Изначально считал, что там будет храниться только код. Без названия страны. old_code и new_code — коды до и после изменения. Согласен, что это не очень правильно. Ваш вариант правильней.

Каким образом можно отследить распады и соединения стран? У меня только одна идея: путём создания/удаления и изменения записей.

Давайте продолжим. Мне достаточно интересно.

multik 3 октября 2013 в 18:42 # ↑ 0

Хорошо, продолжаем, итак у нас есть две первые таблицы country и country_history — в таблице country будет содержаться первая часть телефонного номера: код страны.

Теперь добавим ещё одну таблицу и назовём её secondary_telephone_code — эта таблица будет содержать вторую часть нашего телефонного номера, а сам значимый текстовый код может быть как кодом населённого пункта так и кодом мобильного оператора. В этой таблице связка полей country_id и code будет уникальной (я сейчас не говорю о том как прописывать первичные ключи — а просто указываю на логику таблицы). То есть по сути ваша таблица locality_code превращается в secondary_telephone_code только она теперь напрямую связана с country. Также в таблицу

secondary_telephone_code помимо полей country_id и code — можно добавить поле type или typeId — которое будет указывать на то чем именно является этот код — кодом города? мобильного оператора? или ещё чем-нибудь? а также необязательное поле text — в котором можно хранить любое примечание к коду (название города мобильного оператора или что-то ещё).

Обращаю ваше внимание что type и text не являются обязательным полями (как минимум для начала функционирования системы телефонных номеров).

Также, на данном этапе, пока у нас ещё нет адреса — нам не надо выстраивать связи между таблицей secondary_telephone_code и например таблицами region или другими. То есть выстроить связи в будущем мы сможем — но наши основные связи не поменяются и для логики хранения и изменения телефонных номеров такая сущность как регион не играет никакой роли (не путаем с логикой выбора или вноса информации пользователем/оператором). Название таблицы secondary_telephone_code — может показаться не самым благозвучным — но оно действительно удачное — потому что эта таблица содержит в себе именно то что означает её название вторую часть телефонного кода (можно добавить в название part и превратить таблицу в secondary_part_telephone_code — а можно этого и не делать). То что эта вторая часть может быть кодом города и выбираться из списка после выбора страны и региона, либо кодом оператора — не имеет никакого значения на этапе проектирования структуры телефонного номера.

По аналогии с country_history — мы создаём таблицу secondary_telephone_code_history

Итак у нас есть 4 таблицы — две основные и две для хранения истории изменений основных таблиц.

Добавляем третью основную таблицу phone — как у вас на схеме — и связываем её с таблицей secondary_telephone_code — всё структура данных системы телефонных номеров готова. В ней можно хранить реальные номера телефонов и отслеживать изменения, например если данные поступают из внешних источников. Конечно же, для того, что бы привязать эту структуру к интерфейсу вноса и редактирования информации — её необходимо доработать, и связать с другими подсистемами (адресом, операторами связи и прочими) — но с нашей задачей, хранить в себе любые телефонные номера мира и отслеживать изменения кодов стран городов и мобильных операторов — данная структура справляется. и не содержит ничего лишнего.

Теперь предлагаю вам поставить следующую задачу, исходя из того что у нас уже есть, если вы согласны с тем что мы получили и для чего мы это получили, и того чего вы хотите от системы на следующем шаге. Чем меньше будет шаг тем лучше.

multik 3 октября 2013 в 18:48 # ↑ 0

Также можете выложить текущую схему таблиц что бы мы понимали что находимся в одной точке.

skywatcher 11 сентября 2013 в 20:45 *(комментарий был изменён)* # 0

было бы неплохо, если бы вы добавили еще как проектировать циркулярные зависимости. То есть когда нужна ссылка в двух таблицах друг на друга. Вроде как что я слышал — это плохо. Но вот почему — так и не понял (ну кроме время от времени возникающих deadlock'ов в базе данных)

hbuser 11 сентября 2013 в 21:36 # ↑ 0

Автором являюсь не я. У автора статей, к сожалению, нет ничего на указанную тему. Могу лишь предложить переведенные статьи с зарубежных ресурсов, что-то свое писать, снова к сожалению, не имею времени. На зарубежных ресурсах много качественной и полезной информации имеется. В некоторых случаях больше, чем на отечественных.

aamuvirkku 11 сентября 2013 в 21:31 # +1

Не понимаю ценности данной работы, есть множество учебников, где всё подробно разобрано.

horlon 12 сентября 2013 в 12:37 *(комментарий был изменён)* # 0

Чесно говоря, я, не читав ничего по РСУБД, взялся проэктировать сложные БД и пришел к тому, что описано в данной статье. Не потому, что такой умный, а потому что это логично. Хотел подчерпнуть со статьи что-то новое, но ничего нового не подчерпнул, лишь получил уверенность в том, что делаю. Спасибо за перевод!

Nartis 12 сентября 2013 в 13:47 # +1

> Но обычно наличие двух таблиц в связи один-к-одному считается дурной практикой.

Есть таблица с дополнительными полями профиля пользователя. Эти поля используются только в личном кабинете, да и заполнены хорошо если у 3%. Смысл их хранить в основной таблице пользователя, если большая часть это null-данные?

sdevalex 15 сентября 2013 в 14:32 # ↑ 0

Для сферической базы данных нет разницы, заполнены все поля, насколько их много, время поиска не важно. А разбиение на две таблицы — это элемент оптимизации под конкретную РСУБД.

Только зарегистрированные пользователи могут оставлять комментарии. Войдите, пожалуйста.

Ёжик во фрактальном тумане

Полная энергетическая автономия или как выжить с солнечными батареями в глубинке (часть 1. теоретическая)

Официальный интернет-магазин Panasonic пересылает пароль в открытом виде и в копию 3 адресатам