СОДЕРЖАНИЕ

ВСТУПЛЕНИЕ

Кто-то сравнил данные с сырой нефтью. Раз так, то базы данных – это месторождения, буровые установки, насосы и нефтеочистительные заводы. Данные хранятся в базах и, если вы хотите добраться до них, то должны для начала познакомиться с современным оборудованием.

Базы данных – это инструменты, средства достижения конечного результата. У каждой базы есть своя история и свой взгляд на мир. Чем глубже вы их понимаете, тем лучше оснащены для обуздания мощи, скрытой в постоянно растущем корпусе доступных данных.

Почему именно семь баз данных?

Еще в марте 2010 года мы захотели написать книгу о NoSQL-технологиях. Тогда этот термин только входил в обиход, о нем много говорили, но общего понимания еще не было. Что на самом деле означает слово *NoSQL*? Какие типы систем включать в эту категорию? Какое влияние новые технологии окажут на практику создания программ? Именно на эти вопросы мы и собирались ответить – как для самих себя, так и для других.

Прочитав книгу Брюса Тейта «Seven Languages in Seven Weeks: A Pragmatic Guide to Learning Programming Languages»[1] [Tat10], мы поняли, что он нашел правильный подход. Прогрессивная методика введения в языки нашла отклик в наших сердцах. Мы решили, что подобный способ можно применить и к базам данных, и это позволит доходчиво ответить на поставленные вопросы о технологиях NoSQL.

Что есть в этой книге?

Эта книга адресована опытным разработчикам, которые хотели бы получить общее представление о положении дел в современных тех-

1 Семь языков за семь недель. Прагматическое руководство по изучению языков программирования. *Прим. перев.*

нологиях баз данных. Опыт работы с базами данных приветствуется, хотя и необязателен.

После краткого введения следуют семь глав, посвященных семи базам данных, представляющим пять разных жанров, или стилей, упомянутых во введении: PostgreSQL, Riak, Apache HBase, MongoDB, Apache CouchDB, Neo4J и Redis.

Предполагается, что на проработку каждой главы потребуется полный выходной – три дня. Каждый день завершается упражнениями, в которых развиваются затронутые идеи и темы, а в конце каждой главы приведено резюме, где обсуждаются сильные и слабые стороны базы данных. Вы можете продвигаться быстрее или медленнее – как будет удобнее, но перед тем как продолжить чтение, важно как следует усвоить изложенные концепции. Мы старались подобрать примеры, наглядно высвечивающие характерные особенности каждой базы данных. Чтобы по-настоящему понять, что может предложить та или иная база данных, необходимо некоторое время поработать с ней, то есть засучить рукава и попрактиковаться.

Возможно, у вас возникнет искушение пропустить некоторые главы, однако имейте в виду, что книга рассчитана на последовательное чтение. Некоторые концепции, например распределение-редукция (mapreduce), подробно рассматриваются в начальных главах и лишь бегло – в последующих. Цель книги – дать полное представление о положении дел в современной индустрии баз данных, поэтому мы рекомендуем изучить все главы.

Чего нет в этой книге?

Прежде чем приступать к чтению книги, вы, наверное, захотите узнать, чего в ней нет.

Это не руководство по установке

Установка описанных в книге баз данных иногда осуществляется просто, иногда – с некоторыми сложностями, а иногда – откровенно безобразно. Для одних баз имеются готовые пакеты, другие придется компилировать из исходного кода. Кое-где мы будем давать полезные советы, но, вообще говоря, вы предоставлены сами себе. Опустив описание процедуры установки, мы смогли включить больше полезных примеров и пояснений, а ведь именно это вам и нужно, не так ли?

Руководство администратора? Пожалуй, нет

В этой книге вы не найдете и того, что обычно входит в состав руководства по администрированию. У каждой базы данных бесчисленное множество параметров, флагов и тонкостей настройки; по большей части, все это хорошо документировано в Сети. Нас больше интересует обучение полезным идеям и полное погружение, нежели рутинные повседневные операции. Хотя характеристики работы базы данных могут зависеть от параметров – и в некоторых местах эти характеристики обсуждаются, – мы не собираемся вдаваться в тонкости всех возможных конфигураций. На это просто нет места!

Замечание для пользователей Windows

Эта книга принципиально посвящена разнообразию выбора, преимущественно из различного ПО с открытым исходным кодом на платформах *nix. Корпорация Microsoft стремится к созданию интегрированных сред, что ограничивает возможность выбора более узким набором предопределенных компонентов. Поэтому базы данных с открытым исходным кодом, которые мы здесь рассматриваем, разрабатываются пользователями (и преимущественно *для* пользователей) *nix-систем. Это не наше собственное предубеждение, а отражение текущего положения дел. Поэтому предполагается, что все учебные примеры будут запускаться из оболочки *nix. Если вы работаете в Windows, но всё же хотите поэкспериментировать, рекомендуем настроить среду Cygwin[2]. Можно также запустить виртуальную Linux-машину.

Примеры кода и соглашения

В этой книге встречается код на различных языках. Отчасти это диктуется конкретной рассматриваемой базой данных. Мы старались ограничиться только языками Ruby/JRuby и JavaScript. Мы предпочитаем командные утилиты скриптам, но, когда это нужно для решения поставленной задачи, применяем и другие языки, например PL/pgSQL (Postgres) или Gremlin/Groovy (Neo4J). Мы также рассмотрим программирование некоторых серверных приложений на языке JavaScript с применением Node.js.

2 http://www.cygwin.com/

Если явно не оговорено противное, листинги содержат законченный код, который можно выполнить на досуге. В примерах и фрагментах синтаксические конструкции графически выделены в соответствии с правилами языка. Команды оболочки начинаются со знака `$`.

Сетевые ресурсы

Ценным ресурсом является посвященная этой книге страница на сайте издательства Pragmatic Bookshelf[3]. Там вы найдете весь представленный в книге исходный код, а также средства обратной связи – форум сообщества и форму для отправки сообщений о найденных ошибках, где заодно можно внести предложения по изменениям в будущих изданиях этой книги.

Благодарим всех, кто готов сопровождать нас в путешествии по миру современных баз данных.

Эрик Редмонд и Джим Р. Уилсон

3 http://pragprog.com/book/rwdata/seven-databases-in-seven-weeks

ГЛАВА 1.
Введение

Мы являемся свидетелями поворотного момента в мире баз данных. В течение многих лет реляционная модель была стандартом де факто для решения любых задач – больших и малых. Мы не думаем, что в обозримом будущем реляционные СУБД полностью сойдут со сцены, но многие люди оглядываются по сторонам в поисках альтернатив, в частности: структур данных без схемы, нетрадиционных структур данных, простой репликации, высокой доступности, горизонтального масштабирования и новых способов формулирования запросов. Все эти технологии получили собирательное название *NoSQL*, и именно о них пойдет речь в этой книге.

Мы рассмотрим семь баз данных, представляющих широкий спектр технологий. Вы узнаете о функциональности разных СУБД и принятых при их проектировании компромиссах – долговечность данных или быстродействие, абсолютная согласованность или согласованность в конечном счете и т. д, а также поймете, как принимать оптимальное решение о выборе базы в каждом конкретном случае.

1.1. Все начинается с вопроса

Главный вопрос настоящей книги таков: какая база данных или их комбинация решает задачу оптимальным способом? Если после ее прочтения вы будете понимать, как сделать выбор, принимая во внимание конкретные требования и располагаемые ресурсы, то мы будем довольны.

Но чтобы ответить на этот вопрос, надо знать, какие имеются варианты. Для этого мы собираемся глубоко погрузиться в каждую из семи баз данных, показав ее сильные и слабые стороны. Вам предстоит попрактиковаться в операциях CRUD[1], освежить знания о схемах и найти ответы на следующие вопросы.

1 Create, Read, Update, Delete (создание, чтение, обновление, удаление). *Прим. перев.*

- *К какому типу относится это хранилище данных?* Существует много «жанров» баз данных: реляционные, ключи и значения, столбцовые, документо-ориентированные, графовые. Популярные СУБД, в том числе рассматриваемые в этой книге, обычно можно отнести к одной из этих категорий. Вы узнаете о том, для каких задач предназначена каждая категория. Мы специально выбрали базы, покрывающие все категории: одну реляционную (Postgres), два хранилища ключей и значений (Riak, Redis), столбцовую (HBase), две документо-ориентированных (MongoDB, CouchDB) и одну графовую (Neo4J).
- *Что было побудительным мотивом?* Базы данных создаются не в вакууме. Они предназначаются для решения конкретных практических задач. РСУБД появились, когда гибкость запросов считалась важнее, чем гибкость схемы. С другой стороны, столбцовые базы данных прекрасно приспособлены для хранения больших объемов данных на нескольких машинах; связи между данными при этом отходят на второй план. Мы рассмотрим сценарии применения каждой базы данных и соответствующие примеры.
- *Как обращаться к базе?* Существуют различные способы подключения к базе данных. Если предлагается интерактивная командная утилита, то мы начинаем изучение с нее и только потом переходим к другим вариантам. Если требуется программирование, то мы стараемся ограничиться языками Ruby и JavaScript, хотя временами прибегаем и к другим, например PL/pgSQL (Postgres) или Gremlin (Neo4J). На более низком уровне обсуждаются протоколы типа REST (CouchDB, Riak) и Thrift (HBase). В последней главе рассматривается более сложная конфигурация баз данных, объединенных серверным скриптом для Node.js, написанным на JavaScript.
- *В чем состоит уникальность?* Любое хранилище данных поддерживает запись и обратное считывание данных. Все остальное изменяется в широких пределах. Иногда поддерживаются запросы по произвольным полям. Иногда – индексирование для ускорения поиска. Некоторые базы поддерживают произвольные запросы, для других запросы следует планировать заранее. Схема может быть жесткой и контролироваться СУБД, а может быть просто набором рекомендаций, которые пересматриваются по желанию. Понимание возможностей и огра-

ничений станет существенным подспорьем при выборе СУБД, которая в наибольшей степени отвечает решаемой задаче.

- *Как обстоит дело с производительностью?* Как функционирует база данных и во что это обходится? Поддерживается ли сегментирование? Как насчет репликации? Равномерно ли распределены данные за счет хороших функций хеширования или все данные оказываются в одном узле? Для чего оптимизирована база: для чтения, для записи или для какой-то другой операции? Есть ли какие-нибудь средства настройки и насколько они развиты?
- *Как масштабируется база данных?* Масштабируемость тесно связана с производительностью. Говорить о масштабируемости вне контекста – не указывая, что именно вы хотите *масштабировать*, в общем случае бессмысленно. В этой книге мы расскажем, как задавать правильные вопросы, устанавливающие такой контекст. Вопрос о том, *как* масштабируются разные базы, намеренно не акцентируется, но мы тем не менее поясним, к какому виду масштабирования наиболее приспособлена та или иная база: горизонтальному (MongoDB, HBase, Riak), традиционному вертикальному (Postgres, Neo4J, Redis) или чему-то среднему.

Наша цель заключается не в том, чтобы довести начинающего до уровня эксперта по всем рассматриваемым СУБД. Для детального рассмотрения любой из них потребовалась бы целая книга – и не одна (и такие книги есть). Однако к концу этой книги вы будете ясно представлять себе сильные стороны каждой базы данных и понимать, чем они отличаются друг от друга.

1.2. Жанры

Как и музыкальные произведения, базы данных можно отнести к одному или нескольким стилям. В отдельно взятой композиции используются те же самые ноты, что и в любой другой, но предназначены они для разных слушателей. Мало кто станет слушать мессу си-минор Баха в открытом кабриолете, мчащемся по скоростной трассе 405 в Южной Калифорнии. Вот так и с базами данных – одни лучше приспособлены для решения одних задач, другие – других. Поэтому задавать следует не вопрос «Можно ли использовать эту СУБД для хранения и повышения качества данных?», а вопрос «Нужно ли это делать?».

В этом разделе мы изучим пять основных жанров баз данных, а также скажем, какие базы будем использовать для иллюстрации каждого жанра.

Важно помнить, что большая часть задач, с которыми вы сталкиваетесь на практике, может быть решена с помощью большинства, если не всех, рассматриваемых в этой книге, а также многих других СУБД. Вопрос не столько в том, можно ли приспособить СУБД конкретного стиля к моделированию данных, сколько в том, насколько эффективно будет ее применение в данной предметной области при данных сценариях использования и располагаемых ресурсах. Вы овладеете искусством предсказывать, насколько полезной окажется для вас конкретная база данных.

Реляционные СУБД

Реляционная модель первой приходит на ум большинству разработчиков с опытом в области баз данных. Реляционные системы управления базами данных (РСУБД) основаны на теории множеств, в основе их реализации лежат двумерные таблицы, состоящие из строк и столбцов. Канонический способ взаимодействия с РСУБД – написание запросов на языке Structured Query Language (SQL). Значения данных типизированы, это могут быть числа, строки, даты, неструктурированные двоичные объекты (BLOB) и т. п. Тип данных контролируется системой. Существенно, что благодаря математическим основаниям реляционной модели (теории множеств) исходные таблицы можно соединять и трансформировать в новые, более сложные.

Существует немало реляционных СУБД с открытым исходным кодом – MySQL, H2, HSQLDB, SQLite и многие другие, так что выбирать есть из чего. В этой книге мы остановимся на СУБД PostgreSQL (глава 2).

PostgreSQL

Испытанная в боях СУБД PostgreSQL – самая старая и надежная из всех рассматриваемых в этой книге. Совместимая со стандартом SQL, она покажется знакомой любому, кто раньше работал с реляционными базами данных, и послужит эталоном для сравнения с другими базами. Мы также рассмотрим некоторые малоизвестные средства SQL и уникальные достоинства Postgres. Здесь каждый – от новичка до эксперта – найдет себе что-то по душе.

Хранилища ключей и значений

Хранилище ключей и значений (КЗ-хранилище) – простейшая из всех рассматриваемых моделей. Как следует из названия, КЗ-хранилище сопоставляет значения ключам, как словарь (или хеш-таблица) в любом популярном языке программирования. Некоторые КЗ-хранилища допускают в качестве значений составные типы данных, например хеши или списки, но это необязательно. Есть реализации, в которых ключи можно перебирать, но это также считается дополнительным бонусом. Примером КЗ-хранилища можно считать файловую систему, если рассматривать путь к файлу как ключ, а его содержимое – как значение. Поскольку от КЗ-хранилища требуется так мало, то базы данных этого типа могут демонстрировать невероятно высокую производительность, но в общем случае бесполезны, когда требуются сложные запросы и агрегирование.

Как и в случае реляционных СУБД, имеется много продуктов с открытым исходным кодом. Из наиболее популярных отметим memcached (и родственные ему memcachedb и membase), Voldemort и две системы, рассматриваемые в этой книге: Redis и Riak.

Riak

Riak (глава 3) – это не просто хранилище ключей и значений, система изначально поддерживает такие веб-технологии, как HTTP и REST. Это точный повтор системы Dynamo, используемой компанией Amazon, с некоторыми дополнительными функциями, например, векторные часы для разрешения конфликтов. Значением в Riak может быть всё что угодно: простой текст, XML-документ, изображение и т. д., а связи между ключами описываются именованными структурами, которые называются *ссылками* (links). Riak – не самая известная из рассмотренных в этой книге баз данных, но постепенно она набирает популярность, и на ее примере мы впервые рассмотрим исполнение запросов с помощью технологии распределения-редукции (mapreduce).

Redis

Система Redis поддерживает составные типы данных, в частности отсортированные множества и хеши, а также базовые коммуникационные средства, в том числе публикацию-подписку и блокирующие очереди. Она также располагает одним из самых развитых механизмов запросов среди всех КЗ-хранилищ. А за счет кэширования операций

записи в памяти Redis достигает впечатляющей производительности ценой повышенного риска потери данных в случае аппаратного сбоя. Благодаря этой характеристике она может служить неплохим средством кэширования некритических данных, а также брокером сообщений. Мы рассмотрим эту систему в конце книги (глава 8), где построим приложение, в котором гармонично сочетаются Redis и другие базы данных.

Столбцовые базы данных

Столбцовые, или ориентированные на хранение данных по столбцам базы данных получили свое название благодаря одному существенному аспекту дизайна: данные, принадлежащие одному столбцу (в смысле двумерных таблиц) хранятся рядом. Напротив, в строковых базах данных (к числу которых относятся реляционные), рядом хранятся данные, принадлежащие одной строке. Различие может показаться второстепенным, однако последствия такого проектного решения весьма глубоки. В столбцовых базах данных добавление нового столбца обходится дешево и производится построчно. В каждой строке набор столбцов может быть разным, возможно даже, что в некоторых строках столбцов вообще нет, и, значит, таблица может быть *разреженной* без накладных расходов на хранение null-значений. С точки зрения структуры, столбцовые базы данных занимают промежуточное положение между реляционными СУБД и КЗ-хранилищами.

На рынке столбцовых баз данных конкуренция меньше, чем среди реляционных СУБД и хранилищ ключей и значений. Наиболее популярным три системы: HBase (мы рассмотрим ее в главе 4), Cassandra и Hypertable.

HBase

Из всех рассмотренных нами нереляционных СУБД эта столбцовая база данных ближе всего к реляционной модели. HBase спроектирована по образцу системы Google BigTable, построена на базе Hadoop (механизм распределения-редукции) и предназначена для горизонтального масштабирования на кластерах, составленных из стандартного оборудования. HBase дает строгие гарантии непротиворечивости данных и предоставляет таблицы, состоящие из строк и столбцов – поклонникам SQL это придется по нраву. Готовая поддержка версионирования и сжатия ставит эту СУБД на первое место в категории «большие данные».

Документо-ориентированные базы данных

В документо-ориентированных, или просто документных базах данных хранятся – естественно – документы. В двух словах документ – это некий аналог хеша, в котором имеется поле уникального идентификатора, а в качестве значения могут выступать данные произвольного типа, в том числе другие хеши. Документы могут содержать вложенные структуры и обладают высокой гибкостью, что делает их пригодными для применения в разных предметных областях. Система налагает немного ограничений на входные данные при условии, что они удовлетворяют базовым требованиям к представимости в виде документа. В различных документных базах данных применяются различные подходы к индексированию, формулированию произвольных запросов, репликации, обеспечению согласованности и другим аспектам. Для правильного выбора системы нужно хорошо понимать эти различия и их влияние на конкретный сценарий использования.

Два основных игрока на поле документных баз данных с открытым исходным кодом – MongoDB (рассматривается в главе 5) и CouchDB (глава 6).

MongoDB

СУБД MongoDB проектировалась для хранения *гигантских* объемов данных (*mongo* – часть слова hu*mongo*us – «монструозный»[2]). При настройке сервера Mongo предпочтение отдается согласованности – после операции записи все последующие операции чтения извлекут одно и то же значение (до следующего обновления). Эта особенность делает MongoDB привлекательной альтернативой для тех, кто имеет опыт работы с РСУБД. Кром того, MongoDB поддерживает атомарные операции чтения-записи, в том числе инкрементирование значения и запросы к вложенным документам. Благодаря использованию JavaScript в качестве языка запросов MongoDB поддерживает как простые запросы, так и сложные задания с распределением-редукцией.

CouchDB

Система CouchDB рассчитана на разнообразные сценарии развертывания – от центра обработки данных до настольного ПК и даже

2 Объединение слов huge и monstrous. *Прим. перев.*

смартфона. Написанная на языке Erlang, CouchDB может похвастаться уровнем живучести, нечасто встречающимся среди баз данных. Ее файлы данных почти невозможно повредить, при этом CouchDB сохраняет высокую доступность даже в условиях спорадической потери связи или аппаратных сбоев. Как и в Mongo, языком запросов в CouchDB является JavaScript. Представления описываются функциями mapreduce, которые хранятся в виде документов и реплицируются между узлами как обычные данные.

Графовые базы данных

Из реже используемых стилей следует отметить графовые базы данных, достоинства которых ярко проявляются при обработке данных с большим количеством связей. Графовая база состоит из узлов и связей между ними. Как с узлами, так и со связями можно ассоциировать свойства – пары ключ-значение, – в которых хранятся данные. Истинная сила графовых баз данных заключается в возможности обхода узлов, следуя связям.

В главе 7 мы рассмотрим наиболее популярную на сегодня графовую базу данных – Neo4J.

Neo4J

Операция, на которой другие базы данных часто сдаются, – это обход данных со ссылками на себя или с другими сложно устроенными связями. Именно здесь достоинства Neo4J проявляются во всем блеске. Преимущество графовой базы данных в том и состоит, что обеспечивается быстрый просмотр узлов и связей для поиска нужных данных. Такие базы часто используются в социальных сетях и завоевали признание за свою гибкость, кульминацией которой служит Neo4j.

Многостороннее хранение

На практике различные базы данных часто используются в сочетании. Все еще нетрудно встретить приложение, где применяется только реляционная СУБД, но со временем все популярнее становятся комбинации разных баз данных, в которых сильные стороны каждой позволяют создать экосистему, которая оказывается более мощной, функциональной и надежной, чем сумма ее частей. Эту практику, получившую название *многостороннее хранение* (polyglot persistence) мы рассмотрим в главе 9.

1.3. Вперед и вверх

Мы сейчас находимся в середине кембрийского взрыва разнообразия способов хранения данных; трудно предсказать, куда пойдет эволюция. Но есть достаточные основания полагать, что доминирование какой-то одной стратегии (реляционной или нет) маловероятно. Скорее, мы станем свидетелями появления различных высоко специализированных баз данных, каждая из которых приспособлена к решению задачи из некоторой идеализированной предметной области (хотя, конечно, будут и пересечения). И если сейчас имеются рабочие места, требующие квалификации в администрировании реляционных баз данных, то в будущем все больше будет спрос на специалистов по нереляционным системам.

Базы данных, как языки программирования и библиотеки, являются инструментарием, которым должны владеть любой разработчик. Всякий хороший плотник должен знать, что находится в его поясе для инструментов. И ни один строитель не может надеяться стать прорабом, не владея информацией об имеющихся технологиях.

Считайте эту книгу экспресс-курсом. Прочитав ее, вы научитесь колотить молотком, сверлить отверстия дрелью, работать с гвоздезабивным пистолетом и в конце концов сумеете построить нечто куда более серьезное, чем скворечник. Ну а теперь без дальнейших предисловий приступим к изучению нашей первой базы данных: PostgreSQL.

ГЛАВА 2. PostgreSQL

PostgreSQL – молоток в мире баз данных. Ее хорошо знают, она легко доступна, стабильна и при должном старании и умении способна решать на удивление разнообразные задачи. Нельзя рассчитывать стать опытным строителем, не овладев этим самым распространенным инструментом.

PostgreSQL – реляционная система управления базами данных, то есть основана на теории множеств, реализована в виде двумерных таблиц, где данные хранятся по строкам, и строго контролирует типы столбцов. Несмотря на растущий интерес к новым тенденциям в области баз данных, реляционный стиль является самым популярным и, вероятно, останется таким еще довольно долго.

Преобладание реляционных СУБД объясняется не только встроенным в них обширным инструментарием (триггеры, хранимые процедуры, развитые индексы), безопасностью данных (благодаря свойствам транзакционности ACID[1]), количеством специалистов (многие программисты говорят и думают в реляционных терминах), но и гибкостью формулирования запросов. В отличие от некоторых других хранилищ данных, не требуется заранее планировать, как будут использоваться данные. Если реляционная схема нормализована, то можно предъявлять практически произвольные запросы. PostgreSQL – прекрасный пример системы с открытым исходным кодом, следующей традициям РСУБД.

2.1. Произносится Post-greS-Q-L

PostgreSQL – самая старая и проверенная временем СУБД из всех рассматриваемых в этой книге. К ней прилагаются подключаемые модули для разбора запросов на естественном языке, для построения многомерных индексов, для запросов к географическим данным,

1 Атомарность, непротиворечивость, изолированность, долговечность. *Прим. перев.*

для создания собственных типов данных и для многого другого. В ней реализованы хитроумные механизмы обработки транзакций, она позволяет писать хранимые процедуры на десятке языков и работает на самых разных платформах. В PostgreSQL встроена поддержка Unicode, последовательностей, наследования таблиц, подзапросов, а реализация SQL точнее следует стандарту ANSI, чем любая другая из представленных на рынке. СУБД является быстрой и надежной, способна хранить терабайты данных и доказала работоспособность в таких высоконагруженных проектах, как Skype, французская Национальная касса по выплате пособий многодетным семьям (Caisse Nationale d'Allocations Familiales – CNAF) и Федеральное управление гражданской авиации США.

Что в имени тебе моем?

Проект PostgreSQL существует в современном воплощении с 1995 года, но его корни гораздо глубже. Первоначально проект был написан в университете Беркли в начале 1970-х годов и назывался Interactive Graphics Retrieval System (диалоговая система графического поиска), сокращенно «Ingres». В 1980-х годах была выпущена улучшенная версия post-Ingres, сокращенно Postgres. В 1993 году Беркли прекратил работу над проектом, но ее продолжило сообщество, выпустив систему с открытым исходным кодом Postgres95. В 1996 году проект был переименован в PostgreSQL, чтобы подчеркнуть поддержку новых возможностей SQL и с тех пор так и называется.

Установить PostgreSQL можно разными способами – в зависимости от операционной системы[2]. Помимо базовой системы, нам понадобятся следующие дополнительные пакеты: `tablefunc`, `dict_xsyn`, `fuzzystrmatch`, `pg_trgm` и `cube`. Инструкции по установке приведены на сайте[3].

Установив Postgres, создайте схему `book`, выполнив следующую команду:

```
$ createdb book
```

Схема `book` будет использоваться до конца этой главы. Затем выполните команду, которая проверит корректность установки дополнительных пакетов:

```
$ psql book -c "SELECT '1'::cube;"
```

Если будет выдано сообщение об ошибке, обратитесь к онлайновой документации.

2 http://www.postgresql.org/download/
3 http://www.postgresql.org/docs/9.0/static/contrib.html

2.2. День 1: отношения, операции CRUD и соединения

Не рассчитывая встретить в вас эксперта по реляционным базам данных, мы все же предполагаем, что с одной-другой базой вам доводилось сталкиваться. Более чем вероятно, что эта база была реляционной. Мы начнем с создания и заполнения схемы. Затем мы познакомимся с запросами и, наконец, обсудим аспект, который выделяет реляционные базы данных среди прочих: соединение таблиц.

Как и большинство баз данных, о которых мы будем говорить, Postgres включает сервер, выполняющий всю работу, и командную оболочку, позволяющую подключиться к серверу. По умолчанию сервер прослушивает порт 5432, к которому можно подключиться из оболочки `psql`.

```
$ psql book
```

PostgreSQL выводит приглашение, состоящее из имени базы данных, за которой следует знак решетки, если вы работаете от имени администратора, или знак доллара – если от имени обычного пользователя. В оболочку встроена самая лучшая документация, которую можно получить на консоли. Набрав `\h`, вы получите информацию о командах SQL, а набрав `\?` – информацию о специфичных для `psql` командах, начинающихся со знака обратной косой черты. Подробные сведения о конкретной команде SQL можно получить следующим образом:

```
book=# \h CREATE INDEX
Command: CREATE INDEX
Description: define a new index
Syntax:
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ name ] ON table [ USING method ]
( { column | ( expression ) } [ opclass ] [ ASC | DESC ] [ NULLS { FIRST | ...
[ WITH ( storage_parameter = value [, ... ] ) ]
[ TABLESPACE tablespace ]
[ WHERE predicate ]
```

Перед тем как погружаться в недра Postgres, было бы полезно поближе познакомиться с этим инструментом. Имеет смысл изучить (или освежить в памяти) несколько наиболее употребительных команд, например `SELECT` или `CREATE TABLE`.

Введение в SQL

В PostgreSQL используется принятое в SQL соглашение о том, что отношения называются таблицами (TABLE), атрибуты – столбцами (COLUMN), а кортежи – строками (ROW). Мы будем последовательно придерживаться этой терминологии, хотя в специализированной литературе можно встретить и строгие математические термины: *отношения*, *атрибуты*, *кортежи*. Дополнительные сведения об этих понятиях см. на врезке «Математические отношения».

Работа с таблицами

Принадлежа к реляционным СУБД, PostgreSQL нуждается в предварительном проектировании схемы. Сначала создается схема, а затем вводятся данные, совместимые с определением схемы.

Для создания схемы необходимо задать ее имя и список столбцов с указанием типов и необязательных ограничений. В каждой таблице желательно также завести столбец, содержащий уникальный идентификатор, который позволяет отличить данную строку от всех прочих. Этот идентификатор называется первичным ключом (PRIMARY KEY). Команда SQL для создания таблицы `countries` выглядит следующим образом:

```
CREATE TABLE countries (
  country_code char(2) PRIMARY KEY,
  country_name text UNIQUE
);
```

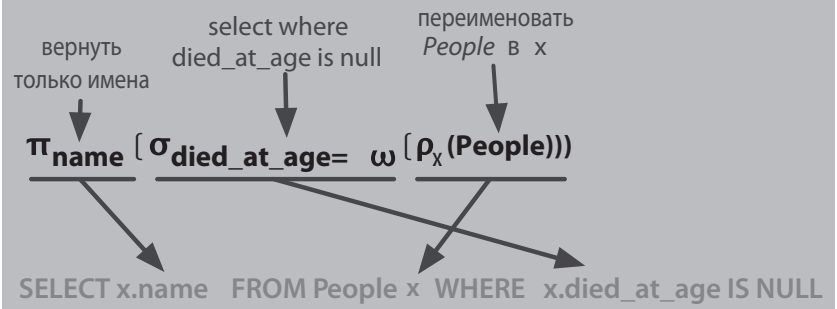
Математические отношения

Реляционные базы данных получили свое название от слова *relation* (отношения, иначе говоря таблицы). Отношение состоит из множества кортежей (*tuple*), или строк, которые сопоставляют *атрибутам* (*attribute*) атомарные значения (например, `{name: 'Genghis Khan', p.died_at_age: 65}`). Состав допустимых атрибутов определяется *заглавным* кортежем, которые отображается на некоторую область определения (*domain*), или ограничивающий тип (то есть на набор столбцов, например `{name: string, age: int}`). Это и есть существо реляционной структуры в двух словах.

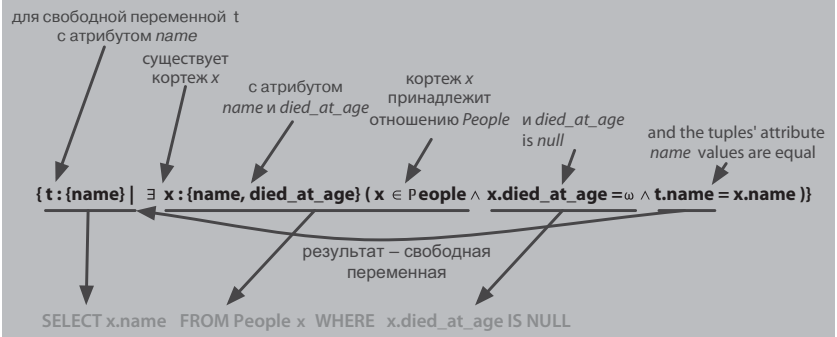
В реализациях применяется гораздо более прагматичная фразеология. Так зачем вообще упоминать эти термины? Затем, чтобы показать, что реляционные базы данных получили свое название благодаря слову *relation* в его математическом смысле. А вовсе не потому, что таблицы «соотносятся» друг с другом посредством внешних ключей, – существуют такие ограничения в действительности или нет, к делу совершенно не относится.

Хотя математическая подоплека в значительной мере скрыта от пользователя, своей мощью модель, безусловно, обязана математике. Именно она

позволяет формулировать сложные запросы, а затем поручать системе оптимизировать их выполнение, применяя определенные алгоритмы. РСУБД строятся на базе раздела теории множеств, называемого *реляционной алгеброй*, в которой определены операции выборки (`WHERE ...`), проецирования (`SELECT ...`), декартова произведения (`JOIN ...`) и другие. Определения устроены, как показано на рисунке ниже.



Интерпретация отношения как физической таблицы (массива массивов, как неизменно повторяют на начальных занятиях по базам данных) может привести к недоразумениям на практике, например попытке написать кода обхода всех строк. Реляционные запросы формулируются гораздо более декларативно – на основе раздела математики, который называется *реляционное исчисление*. Можно показать, что реляционное исчисление эквивалентно реляционной алгебре. PostgreSQL и другие РСУБД оптимизируют запросы, выполняя сведение одного к другому и применяя упрощающие преобразования. Легко видеть, что представление команды SQL на следующем рисунке эквивалентно приведенному выше.



Новая таблица будет содержать набор строк, каждая из которых идентифицируется двузначным кодом, причем все имена уникальны. На оба столбца наложены *ограничения*. Ограничение `PRIMARY KEY` на столбец `country_code` запрещает появление одинаковых кодов

стран. В таблице может существовать только одна строка с кодом `us` и только одна строка с кодом `gb`. Аналогичное ограничение уникальности налагается на столбец `country_name`, хотя он и не является первичным ключом. Вставим в таблицу `countries` несколько строк.

```
INSERT INTO countries (country_code, country_name)
VALUES ('us','United States'), ('mx','Mexico'), ('au','Australia'),
       ('gb','United Kingdom'), ('de','Germany'), ('ll','Loompaland');
```

Проверим, как работает ограничение уникальности. Попытка добавить строку с повторяющимся значением в столбце `country_name` нарушает ограничение уникальности, поэтому вставка не выполняется. Именно с помощью ограничений реляционные базы данных – и PostgreSQL в том числе – гарантируют корректность данных.

```
INSERT INTO countries
VALUES ('uk','United Kingdom');

ERROR: duplicate key value violates unique constraint "countries_country_name_key"
DETAIL: Key (country_name)=(United Kingdom) already exists.
```

Чтобы проверить, те ли строки были вставлены, мы можем прочитать их с помощью команды `SELECT...FROM table`.

```
SELECT *
FROM countries;

 country_code | country_name
--------------+---------------
us            | United States
mx            | Mexico
au            | Australia
gb            | United Kingdom
de            | Germany
ll            | Loompaland
(6 rows)
```

Ни на какой карте вы не найдете страны Loompaland, поэтому давайте удалим ее из таблицы. Чтобы указать, какую строку удалять, мы используем фразу `WHERE`. Следующая команда удалит строку, в которой столбец `country_code` содержит значение `ll`.

```
DELETE FROM countries
WHERE country_code = 'll';
```

Теперь, когда в таблице `countries` остались только реально существующие страны, создадим таблицу `cities`. Чтобы быть уверенными, что код страны `country_code` в любой вставляемой строке при-

сутствует в таблице `countries`, добавим ключевое слово `REFERENCES`. Поскольку столбец `country_code` ссылается на ключ в другой таблице, такое ограничение называется *ограничением внешнего ключа*.

```
CREATE TABLE cities (
  name text NOT NULL,
  postal_code varchar(9) CHECK (postal_code <> ''),
  country_code char(2) REFERENCES countries,
  PRIMARY KEY (country_code, postal_code)
);
```

Об операциях CRUD

`CRUD` – это мнемоническая аббревиатура для запоминания основных операций работы с данными: *Create*, *Read*, *Update*, *Delete* (создание, чтение, обновление, удаление). Все операции, кроме вставки новых записей (*create*), модификации существующих (*update*) и удаления ненужных (*delete*), – сколь бы хитроумными ни были соответствующие запросы – называются *операциями чтения* (*read*). Имея в своем распоряжении операции CRUD, вы можете делать с данными всё что угодно.

На этот раз мы наложили ограничение на столбец `name` в таблице `cities`, запретив значения `NULL`. На столбец `postal_code` наложено ограничение, проверяющее, что строка не пуста (`<>` означает *не равно*). Далее, поскольку первичный ключ (`PRIMARY KEY`) уникально идентифицирует строку, мы создали составной ключ: `country_code + postal_code`. В сочетании эти два поля (код страны и почтовый индекс) определяют строку однозначно.

Postgres располагает широким набором типов данных. Выше вы видели три разных представления строк: `text` (строка произвольной длины), `varchar(9)` (строка переменной длины, не превышающей 9 символов) и `char(2)` (строка, состоящая ровно из двух символов). Подготовив схему, попробуем вставить строку, описывающую город Торонто в Канаде:

```
INSERT INTO cities
VALUES ('Toronto','M4C1B5','ca');

ERROR: insert or update on table "cities" violates foreign key constraint
"cities_country_code_fkey"
DETAIL: Key (country_code)=(ca) is not present in table "countries".
```

Не получилось – и это правильно! Поскольку столбец `country_code` ссылается (`REFERENCES`) на таблицу `countries`, то в этой таблице должна существовать строка с указанным значением `country_code`. Этот механизм, называемый *поддержанием ссылочной целостности*,

проиллюстрирован на рис. 1. Он гарантирует правильность данных. Стоит отметить, что NULL в столбце cities.country_code допускается, так как ключевое слово NULL обозначает отсутствие какого-либо значения. Чтобы запретить NULL-ссылки в столбце country_code, нужно было бы определить его следующим образом:

```
country_code char(2) REFERENCES countries NOT NULL.
```

Теперь попробуем вставить город в США:

```
INSERT INTO cities
VALUES ('Portland','87200','us');

INSERT 0 1
```

Эта операция завершилась успешно. Но по ошибке мы ввели неправильное значение в поле postal_code. Почтовый индекс Портленда на самом деле равен *97205*. Вместо того чтобы удалять и снова вставлять строку, мы можем обновить ее на месте:

```
UPDATE cities
SET postal_code = '97205'
WHERE name = 'Portland';
```

Итак, мы продемонстрировали все операции CRUD – создание, чтение, обновление и удаление строк таблицы – в действии.

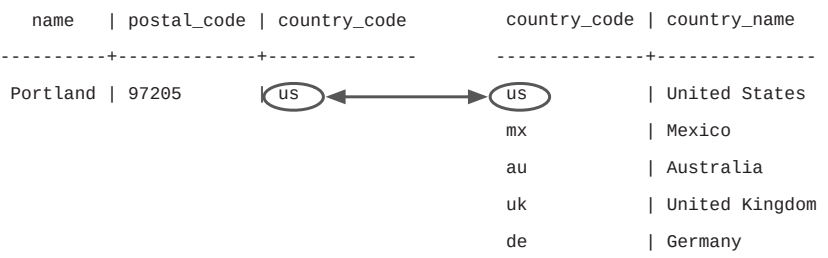


Рис. 1. Ключевое слово REFERENCES говорит, что значение поля в одной таблице должно совпадать с первичным ключом какой-то строки другой таблицы

Чтение с соединением

Все базы данных, о которых мы расскажем в этой книге, поддерживают операции CRUD. Но что выделяет реляционные СУБД типа PostgreSQL из общего ряда – так это возможность соединять таблицы при чтении. Соединением называется операция, которая принимает на входе две таблицы и, комбинируя их определенным об-

разом, возвращает новую таблицу. Что-то вроде составления новых слов из существующих в игре «скрэбл».

Наиболее распространенной является операция *внутреннего соединения*. В простейшем случае с помощью ключевого слова `ON` задаются два столбца (по одному из каждой таблицы), значения которых должны совпадать:

```
SELECT cities.*, country_name
FROM cities INNER JOIN countries
  ON cities.country_code = countries.country_code;

 country_code |   name   | postal_code | country_name
--------------+----------+-------------+---------------
           us | Portland | 97205       | United States
```

Соединение возвращает одну таблицу, в которой присутствуют значения всех столбцов из таблицы `cities` плюс значения столбцов из той строки таблицы `countries`, в которой значение `country_name` такое же, как в соответствующей строке таблицы `cities`.

Можно также выполнить соединение с таблицей типа `cities`, в которой первичный ключ составной. Для проверки составного соединения создадим еще одну таблицу, в которой будет храниться список мест проведения мероприятий.

Место проведения мероприятия существует в конкретной *стране* на территории с конкретным *почтовым индексом*. Поэтому в состав *внешнего ключа* должны входить два столбца, ссылающиеся на оба столбца, образующих *первичный ключ* таблицы `cities`. Ограничение `MATCH FULL` гарантирует, что оба значения существуют или оба содержат `NULL`.

```
CREATE TABLE venues (
  venue_id SERIAL PRIMARY KEY,
  name varchar(255),
  street_address text,
  type char(7) CHECK ( type in ('public','private') ) DEFAULT 'public',
  postal_code varchar(9),
  country_code char(2),
  FOREIGN KEY (country_code, postal_code)
    REFERENCES cities (country_code, postal_code) MATCH FULL
);
```

Столбец `venue_id` – пример часто встречающегося способа определения первичного ключа: автоматически увеличивающиеся целые числа (1, 2, 3, 4 ...). Для задания такого идентификатора употребляется ключевое слово `SERIAL` (в MySQL аналогичная конструкция обозначается ключевым словом `AUTO_INCREMENT`).