Содержание

Об авторах

Салахалдин Джуба более десяти лет работает в промышленности и академических учреждениях, уделяя основное внимание разработкам приложений баз данных в крупномасштабных и корпоративных системах. Имеет степень магистра по управлению природоохранной деятельностью, а также степень бакалавра по проектированию компьютерных систем. Является сертифицированным разработчиком программных решений на основе продуктов Microsoft (MCSD).

Работал преимущественно с базами данных SQL Server, PostgreSQL и Greenplum. Разрабатывал для научных сообществ приложения, связанные с распределенной обработкой географической информации, во время работы в академических учреждениях участвовал во многих международных проектах и разработке стандартов, имеющих отношение к обработке изображений.

Работая программистом, занимался в основном определением ETL-процессов обработки данных, полученных извне, постановкой программных задач, популяризацией передовых практик работы с SQL, проектированием приложений OLTP и OLAP, исследованием и оценкой новых технологий, преподавательской деятельностью и консультированием.

Выражаю глубочайшую благодарность своему коллеге Андрею Волкову, без которого эта книга не состоялась бы. Также благодарю всех, кто оказывал поддержку, а особенно коллектив издательства Packt за советы, корректуру и замечания по оформлению книги. Хочу также поблагодарить свою семью, которая не оставляла меня заботами, хотя я был вынужден уделять много внимания книге в ущерб близким. И наконец, самые теплые слова и глубокая признательность моему покойному отцу, Икраему Джубе, который всегда поддерживал меня, помогал и наставлял в жизни.

Андрей Волков изучал банковские информационные системы. Свою карьеру начал финансовым аналитиком в коммерческом банке. Базы данных были его основным рабочим инструментом, и скоро он понял, что прямые запросы к базе данных и мастерское владение SQL гораздо эффективнее визуальных отчетов, когда нужно произвести ситуативный анализ. Он перешел в группу хранилищ данных и спустя некоторое время возглавил ее, заняв должность архитектора хранилищ данных.

Работал в основном с Oracle и занимался разработкой логических и физических моделей финансовых и бухгалтерских данных, реализацией их с помощью СУБД, разработкой ETL-процессов и аналитикой. Отвечал за обучение пользователей работе с хранилищем данных и инструментами бизнес-аналитики. Преподавание SQL также входило в его обязанности.

Проработав 10 лет в финансовом секторе, он сменил поле деятельности и теперь работает старшим разработчиком баз данных в телекоммуникационной компании. Здесь он имеет дело в основном с СУБД PostgreSQL и отвечает за моделирование данных и реализацию физических структур, разработку хранимых процедур, интеграцию баз данных с другими программными компонентами и разработку хранилища данных.

Имея большой опыт работы с Oracle и PostgreSQL – ведущей коммерческой РСУБД и одной из самых технически продвинутых РСУБД с открытым исходным кодом, – он может сравнивать их и понимает, в чем преимущества той и другой. Опыт реализации различных типов приложений баз данных, а также работы в роли бизнес-аналитика, использующего базы данных как инструмент, научил его понимать, как лучше применять средства СУБД в различных ситуациях. Накопленным опытом он с удовольствием делится в этой книге.

Я благодарен жене и сыну, которые поддерживали меня и давали возможность работать по вечерам и в выходные. Огромное спасибо редакторам за советы и помощь в организации материала. Но больше всего я благодарен основному автору, Салахалдину Джубе, который предложил мне принять участие в работе над книгой, включил в команду и, по сути дела, сделал большую часть работы.

О рецензентах

Д-р Изабель М. Д. Роза – обладатель стипендии имени Марии Склодовской-Кюри, с мая 2016 г. работает в Немецком исследовательском центре интегративного биоразнообразия (iDiv). Родилась в Лиссабоне, Португалия, в 1986 году. Получила степень бакалавра по лесотехнике (2007) и степень магистра по управлению природными ресурсами (2009) в Лиссабонском университете, а также степень доктора по вычислительной экологии (2013) в Имперском колледже Лондона. Является руководителем исследовательского проекта «Применение моделей изменения растительного покрова для решения важных вопросов охраны природы», финансируемого в рамках гранта H2020-MSCA-IF-2015. С 2013 года принимала участие в двух международных проектах, включая финансируемый ЕС проект с бюджетом 1,5 миллиона евро (Terragenesis, ERC-2011-StG_20101109). Автор 15 публикаций в рецензируемых научных журналах, в т. ч. *Nature Ecology and Evolution*, *Current Biology and Global Change Biology*, на которые имеется 252 ссылки (Google citations, октябрь 2017), индекс Хирша равен 8. За время работы в академических учреждениях приобрела ряд навыков, в т. ч. в области статистического анализа, программирования (на языках R, C++ и Python), работы с геоинформационными системами (ArcGIS и QGIS) и создания баз данных (PostgreSQL/PostGIS и SQLServer). Рецензировала книгу «Learning PostgreSQL», также выпущенную издательством Packt.

Шелдон Штраух – ветеран с 23-летним опытом консультирования таких компаний, как IBM, Sears, Ernst & Young, Kraft Foods. Имеет степень бакалавра по организации управления, применяет свои знания, чтобы помочь компаниям самоопределиться. В сферу его интересов входят сбор, управление и глубокий анализ данных, карты и их построение, бизнес-аналитика и применение анализа данных в целях непрерывного улучшения. В настоящее время занимается разработкой сквозного управления данными и добычей данных в компании Enova, оказывающей финансовые услуги и расположенной в Чикаго. В свободное время увлекается искусством, особенно музыкой, и путешествует со своей женой Мэрилин.

Предисловие

Выбор правильной системы управления базами данных – трудная задача из-за большого количества предложений на рынке. В зависимости от бизнес-модели можно выбрать коммерческую СУБД или базу данных с открытым исходным кодом и коммерческой поддержкой. Следует также принимать во внимание ряд технических и нетехнических факторов. Когда речь заходит о реляционной СУБД, PostgreSQL оказывается на вершине рейтинга по нескольким причинам. Лозунг PostgreSQL – *самая передовая база данных с открытым исходным кодом* – отражает развитость технических средств и уверенность сообщества.

PostgreSQL – объектно-реляционная система управления базами данных с открытым исходным кодом. К ее сильным сторонам относится расширяемость, она успешно конкурирует с основными реляционными СУБД: Oracle, SQL Server и MySQL. Благодаря разнообразию расширений и открытой лицензии PostgreSQL часто применяется в исследовательских проектах, но ее код также лежит в основе многих открытых и коммерческих СУБД, например Greenplum и Vertica. К тому же стартапы часто отдают предпочтение PostgreSQL в силу условий лицензии и изобилия компаний, оказывающих коммерческую поддержку.

Существуют версии PostgreSQL для большинства современных операционных систем, включая Windows, Mac и различные дистрибутивы Linux. Имеется также несколько расширений для доступа к данным, управления и мониторинга работы кластеров PostgreSQL, например pgAdmin, OmniDB и psql. Установка и настройки PostgreSQL достаточно просты и поддерживаются большинством диспетчеров пакетов, в т. ч. yum и apt. Разработчики баз данных не испытывают трудностей в освоении PostgreSQL, поскольку она совместима со стандартами **ANSI SQL**. Да и помимо стандартов существует масса ресурсов, которые помогут изучить PostgreSQL, – СУБД отлично документирована и может похвастаться очень активным и хорошо организованным сообществом.

PostgreSQL пригодна как для **OLTP**, так и для OLAP-приложений. Она совместима с **ACID**-транзакциями и для использования в **OLTP**-приложениях не нуждается ни в каких дополнениях. Что касается OLAP-приложений, то PostgreSQL поддерживает оконные функции, адаптеры внешних данных (**FDW** – Foreign Data Wrapper) и наследование таблиц; кроме того, существует немало внешних расширений для этой цели.

Несмотря на совместимость с ACID-транзакциями, PostgreSQL демонстрирует отличную производительность, поскольку в ней применены самые современные алгоритмы и методы. Например, в PostgreSQL используется архитектура **MVCC** (MultiVersion Concurrency Control – управление параллельным доступом с помощью многоверсионности) для обеспечения параллельного доступа к данным. Вдобавок PostgreSQL поддерживает как пессимистическую,

так и оптимистическую конкурентность, а поведение механизма блокировок можно изменять в зависимости от ситуации. Кроме того, в PostgreSQL имеется великолепный анализатор и такие передовые средства, как секционирование данных с помощью наследования таблиц и исключение в силу ограничений, позволяющие ускорить обработку очень больших объемов данных. PostgreSQL поддерживает несколько типов индексов, в т. ч. **B-Tree**, **GiN**, **GiST** и **BRIN**. А начиная с версии PostgreSQL 9.6 поддерживается параллельное выполнение запросов. Наконец, репликация позволяет балансировать нагрузку на различные узлы кластера.

PostgreSQL допускает масштабирование благодаря многочисленным представленным на рынке методам репликации, например Slony и pgpool-II. Дополнительно PostgreSQL изначально поддерживает синхронную и асинхронную потоковую репликацию, а также логическую репликацию. Это делает PostgreSQL чрезвычайно привлекательным решением, поскольку ее можно использовать для создания высокодоступных и высокопроизводительных систем.

Краткое содержание книги

Глава 1 «Реляционные базы данных» содержит введение в концепции реляционных систем управления базами данных, в т. ч. реляционную алгебру и моделирование данных. Здесь же описываются различные типы СУБД: графовые, документные, столбцовые, а также хранилища ключей и значений.

В главе 2 «PostgreSQL в действии» мы расскажем, как установить сервер и клиенты PostgreSQL на различных платформах. Также мы познакомимся с некоторыми возможностями PostgreSQL, в т. ч. встроенной поддержкой репликации и очень богатым набором типов данных.

В главе 3 «Основные строительные блоки PostgreSQL» описываются рекомендации по кодированию, в т. ч. принятые соглашения об идентификаторах. Здесь рассматриваются основные структурные элементы и взаимодействие между ними: шаблонные базы данных, пользовательские базы данных, табличные пространства, роли и настройки. Описываются также основные типы данных и таблицы.

Глава 4 «Дополнительные строительные блоки PostgreSQL» посвящена представлениям, индексам, функциям, пользовательским типам данных, триггерам и правилам. Рассматриваются различные применения этих элементов и сравниваются различные элементы, применимые для решения задачи, например правила и триггеры.

Глава 5 «Язык SQL» – введение в структурированный язык запросов (SQL), используемый для взаимодействия с базой данных: создания и обслуживания структур данных, а также для ввода данных в базу, их изменения, выборки и удаления. В SQL есть команды, относящиеся к трем подъязыкам: **языку определения данных** (**DDL**), **языку манипулирования данными** (**DML**) и **языку управления данными** (**DCL**). В этой главе описаны четыре команды

SQL, составляющие основу языка DML. Особое внимание уделено команде `SELECT`, на примере которой объясняются концепции группировки и фильтрации для демонстрации того, что такое выражения и условия SQL и как используются подзапросы. Здесь же рассматриваются некоторые вопросы реляционной алгебры в применении к соединению таблиц.

В главе 6 «Дополнительные сведения о написании запросов» описаны такие средства SQL, как общие табличные выражения и оконные функции. Они позволяют реализовать вещи, которые без них были бы невозможны, например рекурсивные запросы. Рассматриваются и другие конструкции, как то: фразы `DISTINCT ON` и `FILTER`, а также латеральные подзапросы. Без них, в принципе, можно обойтись, но с их помощью запросы получаются компактнее, проще и быстрее.

В главе 7 «Серверное программирование на PL/pgSQL» рассматриваются параметры функций, в т. ч. количество возвращенных строк и стоимость функции, которые нужны главным образом планировщику запросов. Здесь же представлены управляющие конструкции, например условные команды и циклы. Наконец, объясняется, что такое динамический SQL и как им лучше пользоваться.

В главе 8 «OLAP и хранилища данных» речь пойдет о применении реляционных баз данных для аналитической обработки. Обсуждаются различия между двумя типами рабочей нагрузки: OLTP и OLAP, а также вопросы моделирования в OLAP-приложениях. Кроме того, рассматриваются некоторые технические приемы выполнения **ETL** (extract, transform, load – извлечение, преобразование, загрузка), в т. ч. команда COPY. Также описываются средства PostgreSQL, предназначенные для повышения скорости выборки, в частности просмотр только индексов и секционирование таблиц.

Глава 9 «За пределами традиционных типов данных» посвящена некоторым нестандартным типам: массивам, хешам, JSON-документам и полнотекстовому поиску. Описываются операции и функции для каждого типа данных: инициализация, обновление, доступ и удаление. Наконец, продемонстрировано, как объединить PostgreSQL и Nginx для обслуживания REST-совместимых запросов чтения.

В главе 10 «Транзакции и управление параллельным доступом» подробно обсуждаются свойства ACID и их связь с управлением параллельным доступом. Здесь же рассмотрены уровни изоляции и их побочные эффекты – с демонстрацией этих эффектов на примерах SQL. Также уделено внимание различным методам блокировки, в т. ч. стратегиям пессимистической блокировки: блокировке на уровне строк и рекомендательным блокировкам.

Глава 11 «Безопасность в PostgreSQL» посвящена аутентификации и авторизации. Описываются методы аутентификации в PostgreSQL и объясняется структура конфигурационного файла аутентификации по адресу узла. Также обсуждаются разрешения на доступ к таким объектам базы данных, как схемы, таблицы, представления, индексы и столбцы. Наконец, описано, как защитить секретные данные, в т. ч. пароли, с помощью одностороннего и двустороннего шифрования.

В главе 12 «Каталоги PostgreSQL» приведено несколько рецептов обслуживания кластера базы данных, в т. ч. очистка данных, обслуживание пользовательских процессов, удаление индексов и неиспользуемых объектов базы данных, определение и добавление индексов по внешним ключам и т. д.

В главе 13 «Оптимизация производительности базы данных» обсуждается несколько подходов к оптимизации производительности. Описаны конфигурационные параметры кластера PostgreSQL, используемые для настройки производительности кластера в целом. Также разобраны типичные ошибки при написании запросов и предложено несколько способов повышения производительности, включая построение индексов, секционирование таблиц и исключение в силу ограничений.

В главе 14 «Тестирование» освещены некоторые аспекты процесса тестирования программного обеспечения и его особенности применительно к базам данных. Автономные тесты для баз данных можно писать в виде SQL-скриптов или хранимых функций. Существует несколько каркасов для написания автономных тестов и обработки результатов тестирования.

В главе 15 «PostgreSQL в приложениях на Python» обсуждаются различные продвинутые механизмы, в т. ч. организация пулов соединений, асинхронный доступ и **объектно-реляционное отображение** (**ORM**). Приведен пример, демонстрирующий подключение к базе данных, отправку запроса и выполнение обновления из программы на языке Python. Наконец, дается введение в различные технологии взаимодействия с PostgreSQL, чтобы разработчик имел полную картину современного состояния дел.

В главе 16 «Масштабируемость» рассматриваются проблема масштабируемости и теорема CAP. Здесь же обсуждается репликация в PostgreSQL – физическая и логическая. Описаны различные сценарии масштабирования и их реализация в PostgreSQL.

Что необходимо для чтения книги

Вообще говоря, для сервера и клиентов PostgreSQL не нужно какое-то особое оборудование. PostgreSQL устанавливается на все современные платформы, включая Linux, Windows и Mac. Если необходима некая конкретная библиотека, то мы приводим инструкции по ее установке.

Вам понадобится версия PostgreSQL 10; впрочем, большинство примеров будет работать и с предыдущими версиями. Для выполнения примеров кода и скриптов необходим какой-нибудь клиент PostgreSQL на вашей машине, предпочтительно psql, и доступ к удаленному серверу PostgreSQL.

В среде Windows команда cmd.exe не очень удобна, лучше бы поставить Cygwin (http://www.cygwin.com/) или еще какую-нибудь альтернативу, например PowerShell.

При чтении некоторых глав понадобится дополнительное ПО. Например, в главе 15 нужно будет установить Python и библиотеки, необходимые для работы с PostgreSQL. А в главе 16 удобнее всего будет работать с Docker.

И еще – в главах 9 и 11 рекомендуется использовать Linux, поскольку работать с некоторыми программами, например Nginx и GnuPG, в Windows не очень комфортно. Для запуска Linux на машине под управлением Windows можете воспользоваться программой Virtual Box (https://www.virtualbox.org/).

На кого рассчитана эта книга

Если вам интересно узнать о PostgreSQL – одной из самых популярных в мире реляционных баз данных, – то вы попали по адресу. Те, кого интересует создание добротных приложений для работы с базой данных или хранилищем данных, тоже найдут в книге много полезного. Предварительного опыта программирования или администрирования баз данных для чтения книги не требуется.

Графические выделения

В этой книге тип информации обозначается шрифтом. Ниже приведено несколько примеров с пояснениями.

Фрагменты кода внутри абзаца, имена таблиц базы данных, папок и файлов, данные, которые вводит пользователь, и адреса в Твиттере выделяются следующим образом: «В следующих строчках читается ссылка и результат передается функции `open`».

Кусок кода выглядит так:

```
fin = open('data/fake_weather_data.csv','r',newline='')
reader = csv.reader(fin)
for row in reader:
    myData.append(row)
```

Входная и выходная информация командных утилит выглядит так:

```
$ mongoimport --file fake_weather_data.csv
```

Новые термины и **важные фрагменты** выделяются полужирным шрифтом. Например, элементы графического интерфейса в меню или диалоговых окнах выглядят в книге так: «Чтобы загрузить новые модули, выберите пункт меню **Files** | **Settings** | **Project Name** | **Project Interpreter**».

Предупреждения и важные примечания выглядят так.

Советы и рекомендации выглядят так.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг — возможно, ошибку в тексте или в коде, — мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии данной книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Глава 1

Реляционные базы данных

Эта и следующая главы содержат общий обзор вопросов разработки приложений баз данных, в т. ч. теоретические аспекты реляционных баз данных. Знакомство с теорией поможет не только создавать качественные проекты, но и овладеть тонкостями работы с реляционными базами.

Эта глава касается не только PostgreSQL, но реляционных баз вообще. Будут рассмотрены следующие вопросы:

- **системы управления базами данных:** понимание классификации баз данных дает разработчику возможность использовать лучшее из имеющегося для решения конкретной задачи;
- **реляционная алгебра:** знакомство с реляционной алгеброй позволит овладеть языком SQL, особенно техникой переписывания запросов;
- **моделирование данных:** применяя методы моделирования, вы сможете лучше донести свои идеи до коллег.

Системы управления базами данных

Разные системы управления базами данных (СУБД) поддерживают различные сценарии и требования. У СУБД долгая история. Сначала мы сделаем краткий обзор недавней истории, а затем расскажем о преобладающих на рынке категориях СУБД.

Историческая справка

Термином «база данных» обозначается много разных понятий. Более того, он наводит на мысли о других терминах: данные, информация, структура данных и управление. Базу данных можно определить как набор, или репозиторий данных, обладающий определенной структурой и управляемый **системой управления базами данных (СУБД)**. Данные могут быть структурированными таблицами, слабоструктурированными XML-документами или вовсе не иметь структуры, описываемой какой-то заранее определенной моделью.

На заре развития базы данных были ориентированы в основном на поддержку приложений в сфере бизнеса; это положило начало математически

точно определенной реляционной алгебре и реляционным системам баз данных. Кроме того, во многих отраслях бизнеса, а равно в научных приложениях используются массивы, изображения и пространственные данные, поэтому поддерживаются также новые модели, включая растровые изображения, карты и алгебраические операции с массивами. Графовые базы данных поддерживают запросы к графам, например о поиске кратчайшего пути между двумя вершинами. Они также обеспечивают простой обход графа.

С появлением веб-приложений, в частности социальных порталов, возникла необходимость поддерживать огромное количество запросов распределенным образом. Это привело к новой парадигме в области баз данных, получившей название **NoSQL** (**Not Only SQL**), с другими требованиями, например: примат производительности над отказоустойчивостью и возможность горизонтального масштабирования. В целом на эволюцию баз данных оказывали влияние разнообразные факторы, в том числе:

- **функциональные требования:** сама природа приложений, в которых используются базы данных, потребовала разработки расширений реляционных баз, таких как PostGIS (пространственные данные), или даже специализированных СУБД типа SciDB (для анализа научных данных);
- **нефункциональные требования:** успех языков объектно-ориентированного программирования породил новые тенденции, в частности объектно-ориентированные базы данных. Появились объектно-реляционные системы управления базами данных, перебрасывающие мост между реляционными базами и объектно-ориентированными языками. Взрывной рост объема данных и необходимость обрабатывать терабайты данных на стандартном оборудовании привели к появлению столбцовых баз данных, которые легко масштабируются по горизонтали.

Категории баз данных

Рождались и уходили в небытие различные модели баз данных, включая сетевую и иерархическую модель. Сейчас на рынке преобладают реляционные, объектно-реляционные и NoSQL базы данных. Не следует считать базы данных NoSQL и SQL соперниками – они дополняют друг друга. Применяя ту или иную систему баз данных, можно преодолеть ограничения технологий и выбрать оптимальный вариант.

Базы данных NoSQL

На базы данных NoSQL оказывает влияние теорема CAP, известная также как теорема Брюера. В 2002 году С. Джильберт и Н. Линч опубликовали ее формальное доказательство в статье «Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services». В 2009 году зародилось движение NoSQL. Сегодня насчитывается свыше 150 баз NoSQL (nosql-database.org).

Теорема CAP

Теорема CAP утверждает, что распределенная вычислительная система не может одновременно обеспечить все три следующих свойства:

- **согласованность** – все клиенты сразу же видят последние данные после каждого обновления;
- **доступность** – любой клиент может найти копию нужных данных даже в случае отказа узла. Иными словами, даже после выхода из строя части системы клиенты все равно могут обратиться к данным;
- **устойчивость к разделению** – система продолжает работать даже в случае утраты любого сообщения или отказа части системы.

Характер системы определяется тем, от какого свойства мы отказываемся. Например, пожертвовав согласованностью, мы получим простую, масштабируемую, высокопроизводительную систему управления базами данных. Зачастую основное различие между реляционной и NoSQL базой данных как раз и состоит в согласованности. В реляционной базе гарантируется выполнение свойств **ACID** (**атомарность**, **согласованность**, **изолированность**, **долговечность**). Напротив, во многих базах данных NoSQL принята модель **BASE** (basically available soft-state, eventual consistency – **базовая доступность, неустойчивое состояние, согласованность в конечном счете**).

Мотивация NoSQL

База данных NoSQL предоставляет средства для хранения нереляционных данных, их выборки и манипулирования ими. Такие базы являются распределенными, горизонтально масштабируемыми и, как правило, поставляются с открытым исходным кодом. NoSQL часто основана на модели BASE, отдающей предпочтение доступности над согласованностью. Эта модель дает неформальные гарантии того, что в отсутствие новых обновлений обращение к любому элементу данных рано или поздно вернет его последнюю версию. У такого подхода имеются следующие преимущества:

- простота проектирования;
- горизонтальная масштабируемость и простота репликации;
- отсутствие схемы;
- поддержка гигантских объемов данных.

Теперь рассмотрим некоторые типы баз данных NoSQL.

Хранилища ключей и значений

Хранилище ключей и значений – самый простой тип базы данных. Как следует из названия, хранилище основано на хеш-таблицах. Некоторые хранилища допускают хранение составных типов данных – списков и словарей. В некоторых ситуациях хранилища ключей и значений обеспечивают исключительную скорость, но им недостает поддержки сложных запросов и агрегирования. К числу наиболее известных баз данных такого типа относятся Riak, Redis, Memebase и MemcacheDB.

Столбцовые базы данных

Столбцовые базы данных организованы в виде совокупности столбцов, а не строк. Данные, принадлежащие одному столбцу, хранятся вместе.

В отличие от реляционных баз данных, добавление столбца обходится дешево и выполняется для каждой строки в отдельности. Количество столбцов в разных строках может различаться. Такая структура дает возможность исключить накладные расходы на хранение значений null. Эта модель ориентирована прежде всего на распределенные базы данных.

Одной из самых известных столбцовых баз данных является HBase, основанная на системе хранения Google Bigtable. Столбцовые базы данных проектировались в расчете на очень большие объемы данных, поэтому легко масштабируются. Если набор данных невелик, то использовать HBase не имеет смысла. Во-первых, в рекомендуемой топологии HBase должно быть не менее пяти узлов, а во-вторых, эта база сложна для изучения и администрирования.

Документные базы данных

Документная база данных предназначена для хранения слабоструктурированных данных. Основной единицей хранения в ней является документ. Данные в документах представлены в стандартных форматах, например: XML, JSON и BSON. Для документов не определена единая схема, они не обязаны иметь одинаковую структуру, поэтому обладают высокой степенью гибкости. В отличие от реляционных баз данных, изменить структуру документа легко, и эта процедура не мешает клиентам обращаться к данным.

Графовые базы данных

В основе графовых баз данных лежит теория графов. База данных содержит информацию о вершинах и ребрах графа. С вершинами и ребрами могут быть ассоциированы данные. Графовая база позволяет обходить вершины, следуя вдоль ребер. Поскольку граф – очень общая структура данных, такие базы можно использовать для представления самых разных данных. Самая известная реализация графовой базы данных с открытым исходным кодом и коммерческой поддержкой – Neo4j.

Реляционные и объектно-реляционные базы данных

Реляционные системы управления базами данных относятся к числу наиболее распространенных СУБД. Крайне маловероятно, что в наши дни найдется организация или персональный компьютер, где нет ни одной программы, так или иначе опирающейся на РСУБД.

Приложение может обращаться к реляционной базе, расположенной на выделенном сервере (или нескольких серверах) либо на том же компьютере, где оно работает; в последнем случае упрощенный движок РСУБД встроен в приложение в виде разделяемой библиотеки. Функциональность РСУБД зависит от

поставщика, но большинство совместимо со стандартами ANSI SQL. Формально реляционная база данных описывается реляционной алгеброй и основана на реляционной модели. **Объектно-реляционная база данных** похожа на реляционную и дополнительно поддерживает следующие объектно-ориентированные концепции:

- определенные пользователем и сложные типы данных;
- наследование.

Свойства ACID

В реляционной базе данных логическая операция называется транзакцией. Технически транзакция может состоять из нескольких операций с базой данных: **создания**, **чтения**, **обновления** и **удаления** (create, read, update, delete – **CRUD**). В качестве примера транзакции можно привести распределение фиксированного бюджета по нескольким проектам. Если увеличить сумму, выделенную одному проекту, то необходимо отобрать ту же сумму у какого-то другого проекта. В этом контексте свойства ACID описываются следующим образом:

- **атомарность:** всё или ничего, т. е. если какая-то часть транзакции завершается неудачно, то неудачно завершается и вся транзакция;
- **согласованность:** транзакция переводит базу данных из одного непротиворечивого состояния в другое, также непротиворечивое. Непротиворечивость базы данных обычно описывается с помощью ограничений и связей между данными. Представьте, к примеру, что вы захотели удалить свою учетную запись в интернет-магазине. Чтобы сохранить согласованность, нужно было бы удалить и детальную информацию, ассоциированную с учетной записью, например список адресов. Для этого служат ограничения внешнего ключа, о которых мы подробнее расскажем в следующей главе;
- **изолированность:** параллельное выполнение транзакций переводит систему в такое же состояние, как если бы они выполнялись последовательно;
- **долговечность:** результат зафиксированных, т. е. успешно выполненных, транзакций сохраняется даже в случае пропадания электропитания или аварии сервера. В PostgreSQL это обеспечивается **журналом предзаписи** (write-ahead log – WAL). В других базах данных, например в Oracle, аналогичный механизм называется журналом транзакций.

Язык SQL

Реляционные базы данных обычно дополняются **структурированным языком запросов** (structured query language – SQL). SQL – декларативный язык программирования баз данных, стандартизованный **Американским национальным институтом стандартов** (**ANSI**) и **Международной организацией по стандартизации** (**ISO**). Первый стандарт SQL был опубликован в 1986 го-

ду, за ним последовали стандарты SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011 и SQL:2016.

Язык SQL состоит из нескольких частей:

- **язык определения данных** (**DDL**): позволяет определить и изменить структуру реляционной базы данных;
- **язык манипулирования данными** (**DML**): служит для выборки данных из отношений;
- **язык управления данными** (**DCL**): служит для управления правами доступа.

Понятия реляционной модели

Реляционная модель – это логика первого порядка, или исчисление предикатов, разработанная Эдгаром Ф. Коддом в 1970 году и изложенная в статье «A relational model of data for large shared data banks»[1]. База данных в ней представлена в виде набора отношений (relation). Состояние базы данных в целом определяется состоянием всех имеющихся в ней отношений. Из отношений можно извлекать различную информацию, применяя операции соединения, агрегирования и фильтрации. В этом разделе мы дадим обзор основных понятий реляционной модели и начнем с описания отношения, кортежа, атрибута и домена. Эти термины, используемые в формальной реляционной модели, соответствуют таблице, строке, столбцу и типу данных в языке SQL.

Отношение

Отношение можно представлять себе как таблицу с заголовком, столбцами и строками. Имя таблицы и ее заголовок необходимы для интерпретации данных, хранящихся в строках. Каждая строка представляет группу взаимосвязанных полей, описывающих некий объект.

Отношение состоит из множества кортежей. Каждый кортеж – это упорядоченное множество атрибутов, причем атрибуты и порядок их следования во всех кортежах одинаковы. Атрибут характеризуется доменом, т. е. типом и именем.

	customer_id	first_name	last_name	email
Кортеж →	1	thomas	sieh	thomas@example.com
Кортеж →	2	wang	kim	kim@example.com
	Атрибут ↑	Атрибут ↑	Атрибут ↑	Атрибут ↑

Схема отношения описывается его именем и атрибутами. Например, `customer (customer_id, first_name, last_name, email)` – схема отношения customer. **Состояние отношения** определяется множеством составляющих его кортежей, поэтому добавление, удаление и изменение кортежа переводят отношение в другое состояние.

---

[1] Реляционная модель данных для больших совместно используемых банков данных. http://citforum.ru/database/classics/codd/.

Место кортежа в отношении и их относительный порядок не играют роли. Кортежи отношения можно упорядочить по одному или по нескольким атрибутам. В отношении не может быть кортежей-дубликатов.

Отношение может представлять сущности реального мира, например клиента, или ассоциации между отношениями. Так, клиент может пользоваться несколькими службами, а каждая служба может быть предложена нескольким клиентам. Эту ситуацию можно смоделировать тремя отношениями: `customer`, `service` и `customer_service`, причем `customer_service` ассоциирует отношения `customer` и `service`. Разделение данных по отношениям – ключевая идея реляционного моделирования, у которой есть специальное название – нормализация. В процессе нормализации отношения и их столбцы организуются так, чтобы уменьшить избыточность. Предположим, к примеру, что служба хранится прямо в отношении `customer`. Если одной службой пользуется несколько клиентов, то это приведет к дублированию данных о клиентах. К тому же для обновления службы пришлось бы обновлять все ее копии в таблице клиентов.

Кортеж

Кортеж – это упорядоченный набор атрибутов. Атрибуты перечисляются в круглых скобках через запятую, например (john, smith, 1971). Элементы кортежа идентифицируются именем атрибута. Кортежи обладают следующими свойствами:

- $(a_1, a_2, a_3,\ldots, a_n) = (b_1, b_2, b_3,\ldots, b_n)$ тогда и только тогда, когда $a_1 = b_1$, $a_2 = b_2$, …, $a_n = b_n$;
- кортеж не является множеством, порядок элементов имеет значение, и дубликаты разрешены:
  - $(a_1, a_2) \neq (a_2, a_1)$;
  - $(a_1, a_1) \neq (a_1)$;
- количество атрибутов в кортеже конечно.

В строгой реляционной модели не допускаются многозначные и составные атрибуты. Это важно с точки зрения устранения избыточности и повышения степени согласованности данных. Но в современных реляционных базах данных это ограничение частично снято, чтобы можно было хранить данные таких составных типов, как JSON.

По поводу того, как применять нормализацию, ведутся ожесточенные споры. Мы рекомендуем нормализовать данные, если нет основательных причин этого не делать.

Значение NULL

Для исчисления предикатов в реляционных базах данных применяется **трехзначная логика** (**3VL**), в которой значений истинности три: истина (true), ложь (false) и неизвестно (unknown), или NULL. В реляционной базе данных третье значение, NULL, можно интерпретировать разными способами, например: данные неизвестны, данные отсутствуют, неприменимо или данные бу-

дут загружены позже. Трехзначная логика устраняет неоднозначность. Например, два значения NULL не равны друг другу.

Ниже приведены таблицы истинности для логических операторов OR и AND; отметим, что оба оператора коммутативны, т. е. *A AND B = B AND A*:

A	B	A AND B	A OR B
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	TRUE
NULL	TRUE	NULL	TRUE
FALSE	FALSE	FALSE	FALSE
NULL	FALSE	FALSE	NULL
NULL	NULL	NULL	NULL

А вот таблица истинности для оператора NOT:

A	B
TRUE	FALSE
FALSE	TRUE
NULL	NULL

Атрибут

У каждого атрибута есть имя и домен, причем никакие два атрибута одного отношения не могут иметь одинаковые имена. Домен определяет множество допустимых значений атрибута. Один из способов задать домен – указать тип данных и ограничения на этот тип. Например, почасовая оплата должна быть положительным вещественным числом, большим пяти (если предположить, что минимальная почасовая оплата равна пяти долларам). Домен может быть непрерывным, как в случае зарплаты (положительное вещественное число), или дискретным, как в случае пола.

В строгой реляционной модели на домен налагается ограничение: значение должно быть атомарным, т. е. неделимым. Например, домен атрибута name (полное имя) не является атомарным, потому что имя можно разделить на две части: имя и фамилию. Приведем несколько примеров доменов:

- **телефон:** текст определенной длины, состоящий из цифр;
- **код страны:** двузначные и трехзначные коды стран определены в стандарте ISO 3166 (соответственно ISO alpha-2 и ISO alpha-3). Так, коды Германии равны DE и DEU.

В реальных приложениях лучше использовать международные стандарты в справочных таблицах, в т. ч. стран и валют. Это сделает ваши данные гораздо более понятными внешним программам и повысит качество данных.

Ограничения

В реляционной модели определено много ограничений для контроля целостности данных, избыточности и допустимости:

- **избыточность:** кортежи-дубликаты запрещены в отношении;
- **допустимость:** доменные ограничения контролируют допустимость данных;
- **целостность:** отношения в базе данных связаны друг с другом. Операция над одним отношением, например обновление или удаление кортежа, может оставить другое отношение в некорректном состоянии.

Ограничения в реляционной базе данных можно отнести к двум категориям:

- унаследованные от реляционной модели: ограничения доменной целостности, целостности сущностей и ссылочной целостности;
- семантические ограничения, бизнес-правила и ограничения, свойственные приложению, которые нельзя явно выразить средствами реляционной модели. Однако появление процедурных языков на основе SQL, в частности PL/pgsql в PostgreSQL, позволило моделировать такие ограничения и в реляционной базе данных.

Ограничение доменной целостности

Ограничение доменной целостности гарантирует допустимость данных. Прежде всего необходимо определиться с типом данных. Типом данных домена может быть integer, real, boolean, character, text, inet и т. д. Например, имя и адрес электронной почты имеют тип text. Зная тип данных, можно уже определить проверочное ограничение, например шаблон почтового адреса.

- **Проверочное ограничение:** такое ограничение применяется к одному атрибуту или к группе атрибутов. Рассмотрим отношение `customer_service`, в котором определены атрибуты `customer_id`, `service_id`, `start_date`, `end_date`, `order_date`. Мы можем определить для него ограничение `start_date<end_date`, проверяющее, что дата начала меньше даты конца.
- **Ограничение умолчания:** у атрибута может быть значение по умолчанию. Это может быть либо фиксированное значение, например стандартная почасовая оплата труда, скажем, 10 долларов. Или же динамически вычисляемое значение, например значение функции random, current time или date. Например, в отношении `customer_service` атрибут `order_date` может по умолчанию принимать текущую дату.
- **Ограничение уникальности:** гарантирует, что значение атрибута уникально среди всех кортежей отношения. Допускается также значение null. Пусть определено отношение `player` с атрибутами (`player_id`, `player_nickname`). У игрока есть уникальный идентификатор, и он может выбрать себе ник, который также должен идентифицировать его однозначно.
- **Ограничение not null:** по умолчанию атрибут может принимать значение null. Ограничение not null запрещает это. Например, у каждого человека в книге записей о рождении должно быть имя.

Ограничение сущностной целостности

В реляционной модели отношение определяется как множество кортежей. Это означает, что все кортежи в нем должны быть различны. Ограничение сущ-

ностной целостности означает наличие первичного ключа, представляющего собой один или несколько атрибутов со следующими характеристиками:

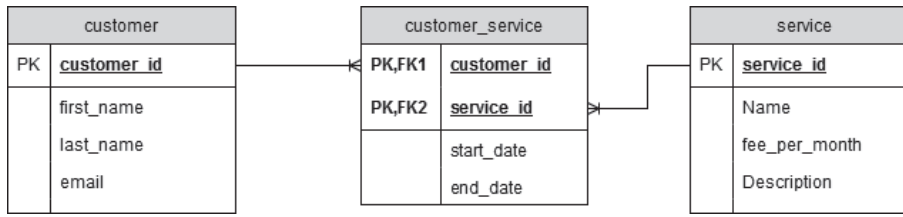
- ключ должен быть уникален;
- ни один атрибут, входящий в ключ, не должен принимать значение null.

В каждом отношении может быть только один первичный ключ, но много уникальных ключей. Ключ-кандидат – это минимальный набор атрибутов, с помощью которых можно однозначно идентифицировать кортеж. Любой уникальный, не принимающий значения null атрибут может быть ключом-кандидатом. Набор, состоящий из всех атрибутов, образует суперключ. На практике часто выбирают в качестве первичного ключа один атрибут, а не составной ключ (состоящий из двух или более атрибутов), чтобы было проще соединять отношения между собой.

Первичный ключ, генерируемый СУБД, называют **суррогатным**, или **синтетическим**. В противном случае ключ называется **натуральным**. В роли суррогатных ключей-кандидатов могут выступать порядковые номера или **универсальные уникальные идентификаторы** (**UUID**). У суррогатных ключей много преимуществ, в т. ч. высокая производительность, устойчивость к изменению требований, адаптируемость и совместимость с объектно-реляционными отображениями. Главный недостаток суррогатных ключей состоит в том, что они открывают возможность для появления кортежей-дубликатов, отличающихся только значением ключа.

Ограничения ссылочной целостности

Отношения связаны между собой общими атрибутами. Ограничения ссылочной целостности контролируют связи между двумя отношениями и гарантируют, что данные в кортежах согласованы. Если кортеж одного отношения ссылается на кортеж другого отношения, то этот второй кортеж должен существовать. Так, если с клиентом связана служба, то должны существовать и клиент, и служба. Например, в отношении `customer_service` не может быть кортежа со значениями (`5, 1,01-01-2014, NULL`), поскольку не существует клиента, для которого `customer_id` равно `5`.



Отсутствие ограничений ссылочной целостности может стать причиной многих проблем:

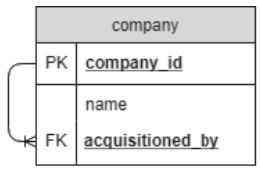
- некорректные данные в общих атрибутах;
- некорректная информация при соединении данных из разных отношений;

- снижение производительности из-за неоптимального плана выполнения, построенного планировщиком PostgreSQL или сторонней программой.

Наличие внешних ключей может повысить производительность чтения данных из нескольких таблиц. Планировщик выполнения запросов будет иметь более точную оценку количества подлежащих обработке строк. Отключение внешних ключей при выполнении массовой вставки резко ускоряет операцию.

Ограничения ссылочной целостности реализуются с помощью внешних ключей. **Внешним ключом** называется атрибут или группа атрибутов, однозначно идентифицирующих кортеж в отношении, на которое производится ссылка (внешнем отношении). В силу самой своей природы внешний ключ часто является первичным во внешнем отношении. Но, в отличие от первичного ключа, внешний может принимать значение null. Он также может ссылаться на уникальный атрибут внешнего отношения. Допуская значения null во внешнем ключе, мы получаем возможность моделировать различные ограничения кардинальности, т. е. ограничения на количество элементов по обе стороны связи. Например, если родитель может иметь более одного потомка, то говорят о связи один-ко-многим, поскольку с одним кортежем первичного (ссылающегося) отношения может быть ассоциировано несколько кортежей внешнего. Отношение может также ссылаться само на себя. Тогда внешний ключ называется **автореферентным**, или **рекурсивным**.

Например, одна компания может быть приобретена другой.



company_id	name	acquisitioned_by
1	Facebook	
2	WhatsApp	1

Для гарантии целостности данных можно определить несколько типов поведения в случае, когда кортеж во внешнем отношении обновляется или удаляется:

- **каскадом:** если кортеж во внешнем отношении удаляется или его внешний атрибут обновляется, то ссылающиеся на него кортежи первичного отношения также удаляются или обновляются;
- **запрет:** кортеж, на который ведет ссылка, не может быть удален, а его внешний атрибут не может быть изменен;
- **ничего не делать:** то же, что запрет, но решение откладывается до конца транзакции;
- **присваивание значения по умолчанию:** если кортеж во внешнем отношении удаляется или внешний атрибут обновляется, то внешнему ключу в первичном отношении присваивается значение по умолчанию;

- **присваивание null:** если кортеж во внешнем отношении удаляется, то внешнему ключу в первичном отношении присваивается значение null.

Семантические ограничения

Ограничения целостности или ограничения бизнес-логики касаются приложения в целом. Например, в любой момент у клиента может быть не более одной активной службы. Такие ограничения проверяются либо на уровне бизнес-логики прикладной программы, либо с помощью процедурных языков на основе SQL. Для этой цели можно также использовать триггеры и правила. Какой подход выбрать – процедурный язык на основе SQL или высокоуровневый язык программирования, – зависит от приложения; иногда комбинируют оба.

У процедурного языка на основе SQL имеются следующие преимущества:

- **производительность:** в РСУБД часто имеются сложные оптимизаторы для построения эффективных планов выполнения. А в некоторых случаях, например в приложениях добычи данных, объем обрабатываемых данных очень велик. Обработка на процедурном языке позволяет исключить передачу данных по сети. Кроме того, в некоторых процедурных языках применяются хитроумные алгоритмы кеширования;
- **срочные изменения:** процедурный язык на основе SQL позволяет исправлять ошибки, не останавливая службу.

У реализации бизнес-логики на уровне базы данных есть много плюсов и минусов, эта тема постоянно рождает споры. Отметим лишь некоторые недостатки: видимость внутреннего устройства базы данных, неэффективная работа программистов, вынужденных писать код без поддержки со стороны инструментов и IDE, трудности повторного использования кода.

Реляционная алгебра

Реляционная алгебра – это формальный язык реляционной модели. Она определяет набор замкнутых операций над отношениями, т. е. результатом любой операции является новое отношение. Реляционная алгебра наследует многие операторы от теории множеств. Операции реляционной алгебры можно отнести к двум группам:

- в первую группу входят теоретико-множественные операции: объединение, пересечение, разность множеств и декартово (или перекрестное) произведение;
- во вторую группу входят операции, специфичные для реляционной модели, например выборка и проекция. Операции реляционной алгебры бывают также унарными и бинарными.

Перечислим примитивные операторы:

- `select` (σ): унарная операция (выборка), записываемая в виде $\sigma\varphi_R$, где φ – предикат. Этот оператор выбирает такие кортежи *R*, для которых φ принимает значение true;

- `project` (π): унарная операция (проекция), которая отбирает часть атрибутов отношения и записывается в виде $\pi_{a1,a2,...,an}R()$, где $a1$, $a2$, ..., an – имена атрибутов;
- `cartesian product` (×): бинарная операция, которая порождает более сложное отношение, комбинируя кортежи двух операндов. Пусть R и S – два отношения, тогда $R \times S = (r_1, r_2, ..., r_n, s_1, s_2, ..., s_n)$, где $(r_1, r_2, ..., r_n) \in R$ и $(s_1, s_2, ..., s_n) \in S$;
- `union` (∪): теоретико-множественное объединение отношений. Отметим, что отношения должны быть совместимы по объединению, т. е. состоять из одинаковых атрибутов в одном и том же порядке. Формально $R \cup S = (r_1, r_2, ..., r_n) \cup (s_1, s_2, ..., s_n)$, где $(r_1, r_2, ..., r_n) \in R$ и $(s_1, s_2, ..., s_n) \in S$;
- `difference` (-): бинарная операция (разность), операнды которой должны быть совместимы по объединению. Создается новое отношение, содержащее те кортежи первого операнда, которые отсутствуют во втором. Формально $R - S = (r_1, r_2, ..., r_n)$, где $(r_1, r_2, ..., r_n) \in R$ и $(r_1, r_2, ..., r_n) \notin S$;
- `rename` (ρ): унарная операция (переименование), которая воздействует только на атрибуты. В основном этот оператор применяется, чтобы различить одноименные атрибуты в соединяемых отношениях или чтобы дать атрибуту более вразумительное имя для целей презентации. Записывается в виде $\rho_{a/b}R$, где a и b – имена атрибутов, причем b – один из атрибутов R.

Помимо примитивных операторов, имеются агрегатные функции, например: `sum`, `count`, `min`, `max` и `avg`. Примитивные операторы можно использовать для определения других реляционных операторов, в т. ч. левого соединения, правого соединения, эквисоединения и пересечения. Реляционная алгебра очень важна в силу своей выразительной мощи, помогающей оптимизировать и переписывать запросы. Например, операция выборки коммутативна, т. е. $\sigma_a\sigma_b R = \sigma_b\sigma_a R$. Другой пример – две последовательные выборки можно заменить одной с конъюнкцией предикатов: $\sigma_a\sigma_b R = \sigma_{a\ AND\ b} R$.

Операции выборки и проекции

Операция `SELECT` применяется, чтобы выбрать часть кортежей отношения. Она всегда возвращает множество кортежей без повторений, что следует из ограничения сущностной целостности. Например, запрос «*дай мне информацию о клиенте, для которого customer_id равно 2*» записывается следующим образом:

$\sigma_{customer\_id\ =2}$ customer.

Как уже было сказано, операция выборки коммутативна, т. е. запрос «*дай мне всех клиентов с именем kim, для которых известен адрес электронной почты*» можно записать тремя способами:

$\sigma_{email\ is\ not\ null}(\sigma_{first\_name=kim}$ customer);
$\sigma_{first\_name=kim}(\sigma_{email\ is\ not\ null}$ customer);
$\sigma_{first\_name=kim\ and\ email\ is\ not\ null}$ (customer).

Конечно же, предикаты выборки зависят от типа данных. Для числовых типов в качестве оператора сравнения можно указывать ≠, =, <, >, ⩾, ⩽. Предикативное выражение может также содержать скобки и функции. В SQL эквивалентом операции выборки является команда `SELECT *`, а предикат определяется во фразе `WHERE`.

Символ * означает «все атрибуты отношения». Отметим, что в производственных системах использовать * не рекомендуется, лучше перечислять необходимые атрибуты явно.

Следующее предложение `SELECT` эквивалентно выражению реляционной алгебры $\sigma_{\text{customer\_id}=2}$ customer:

```
SELECT * FROM customer WHERE customer_id = 2;
```

Оператор проекции можно наглядно представлять себе как вертикальный срез таблицы. Запрос «*дай мне имена клиентов*» в реляционной алгебре записывается так:

$\pi_{\text{first\_name, last\_name}}$ customer.

Ниже представлен результат проецирования:

first_name	last_name
thomas	sieh
wang	kim

В строгой реляционной модели кортежи-дубликаты не допускаются; количество кортежей, возвращенных оператором `PROJECT`, всегда меньше или равно количеству кортежей во всем отношении. Если в список атрибутов в операторе `PROJECT` входит первичный ключ, то в результирующем отношении будет столько же строк, сколько в исходном.

Оператор проекции тоже поддается оптимизации. Например, справедливо следующее тождество для каскадных проекций:

$\pi_a(\pi_a,\pi_b(R)) = \pi_a(R)$.

В SQL эквивалентом оператора `PROJECT` является команда `SELECT DISTINCT`. Ключевое слово `DISTINCT` означает, что нужно устранить дубликаты. Чтобы получить показанный выше результат, следует выполнить такую команду SQL:

```
SELECT DISTINCT first_name, last_name FROM customers;
```

В некоторых случаях операции `PROJECT` и `SELECT` можно выполнять в любом порядке. Запрос «*дай мне имя клиента с идентификатором customer_id = 2*» можно записать двумя способами:

$\sigma_{\text{customer\_id}=2}(\pi_{\text{first\_name, last\_name}}$ customer);
$\pi_{\text{first\_name, last\_name}}(\sigma_{\text{customer\_id}=2}$ customer).

Но бывает и так, что порядок операторов PROJECT и SELECT важен, иначе получается некорректное выражение. Запрос «*дай мне фамилии всех клиентов с именем kim*» можно записать только так:

$\pi_{(last\_name}(\sigma_{first\_name=kim}$ customer).

Операция переименования

Операция rename применяется, чтобы изменить имя атрибута результирующего отношения или присвоить конкретное имя результирующему отношению. Она позволяет:

- устранить неоднозначность в случае, когда в двух или более отношениях встречаются одноименные атрибуты;
- присвоить вразумительные имена атрибутам, особенно при формировании отчетов;
- изменить определение отношения, сохранив обратную совместимость.

Эквивалентом оператора rename в SQL является ключевое слово AS. Ниже создается отношение, содержащее один кортеж с одним атрибутом, которому присвоено имя PI:

```
SELECT 3.14::real AS PI;
```

Теоретико-множественные операции

К теоретико-множественным относятся операции union (объединение), intersection (пересечение) и minus (разность). Intersection не является примитивным оператором реляционной алгебры, потому что его можно выразить через объединение и разность:

$A \cap B = ((A \cup B) - (A - B)) - (B - A)$.

Операторы пересечения и объединения коммутативны:

$A \cap B = B \cap A$;
$A \cup B = B \cup A$.

Запрос «*дай мне идентификаторы всех клиентов, с которыми не ассоциирована ни одна служба*» можно записать так:

$\pi_{customer\_id}$ customer – $\pi_{customer\_id}$ customer_service.

Операция декартова произведения

Операция cartesian product позволяет построить отношение, содержащее комбинации кортежей двух других отношений. Пусть A и B – два отношения, и $C = A \times B$. Тогда:

количество атрибутов C равно количество атрибутов A + количество атрибутов B;

*количество кортежей C равно количество атрибутов A * количество кортежей B.*

На следующем рисунке показано декартово произведение отношений `customer` и `customer_service`.

customer_id	first_name	las_name
1	thomas	sieh
2	wang	kim

X

customer_service_id	customer_id	start_date	end_date
1	1	2017-11-01	2018-11-01
2	1	2017-11-01	2018-11-01

=

customer_id	first_name	las_name	customer_service_id	customer_id	start_date	end_date
1	thomas	sieh	1	1	2017-11-01	2018-11-01
2	wang	kim	1	1	2017-11-01	2018-11-01
1	thomas	sieh	2	1	2017-11-01	2018-11-01
2	wang	kim	2	1	2017-11-01	2018-11-01

В SQL эквивалентом декартова произведения является оператор перекрестного соединения `CROSS JOIN`. Запрос «*для клиента с идентификатором 1 выбрать идентификатор и имя клиента, а также идентификаторы ассоциированных с ним служб*» записывается так:

```
SELECT DISTINCT customer_id, first_name, last_name, service_id
FROM customer AS c CROSS JOIN customer_service AS cs
WHERE c.customer_id=cs.customer_id AND c.customer_id = 1;
```

На этом примере наглядно видна связь между реляционной алгеброй и языком SQL. Мы использовали операции `select`, `rename`, `project` и `cartesian product`. Покажем, как можно с помощью реляционной алгебры оптимизировать выполнение этого запроса. Его можно было бы выполнить несколькими способами.

План выполнения 1:

1. Выбрать клиента, для которого `customer_id = 1`.
2. Выбрать службу для клиента с идентификатором `customer_id = 1`.
3. Выполнить перекрестное соединение отношений, полученных на шагах 1 и 2.
4. Спроецировать отношение, полученное на шаге 3, на атрибуты `customer_id`, `first_name`, `last_name` и `service_id`.

План выполнения 2:

1. Выполнить перекрестное соединение отношений `customer` и `customer_service`.
2. Выбрать все кортежи, для которых

   ```
   customer_service.customer_id=customer.customer_id and
   customer.customer_id = 1.
   ```

3. Спроецировать отношение, полученное на шаге 2, на атрибуты `customer_id`, `first_name`, `last_name` и `service_id`.

Запрос SELECT написан именно таким образом, чтобы показать, как операции реляционной алгебры транслируются на язык SQL. Современный SQL позволяет проецировать на атрибуты без использования DISTINCT. Кроме того, следовало бы использовать внутреннее соединение, а не перекрестное.

У любого плана выполнения есть стоимость, выражаемая в терминах процессорного времени и количества операций доступа к памяти (запоминающему устройству с произвольной выборкой – ЗУПВ) и диску. РСУБД выбирает план с наименьшей стоимостью. В показанных выше планах операция `rename`, как и оператор `distinct`, для простоты была опущена.

Моделирование данных

Модель данных описывает сущности реального мира, например клиента, службу и продукты, а также связи между ними. Модель данных – это абстракция отношений в базе данных. С помощью модели разработчик представляет бизнес-требования в виде отношений. Модели также служат средством обмена информацией между разработчиками и заказчиками.

На предприятии модели данных играют очень важную роль – обеспечение согласованности данных между взаимодействующими системами. Так, если некоторая сущность не определена или определена неправильно, то это повлечет за собой несогласованность данных и разночтения. Например, если семантика сущности «клиент» определена расплывчато и разные подразделения называют одну и ту же сущность по-разному, скажем, «customer» и «client», то в отделе эксплуатации может возникнуть путаница.

Виды моделей данных

В стандарте ANSI определены следующие виды моделей данных:

- **концептуальная модель данных** – описывает семантику доменов и служит для перечисления основных бизнес-правил, действующих лиц и концепций. Содержит высокоуровневое описание бизнес-требований и часто называется моделью данных верхнего уровня;
- **логическая модель данных** – описывает семантику в контексте определенной технологии. Например, для объектно-ориентированных языков это могут быть UML-диаграммы классов;
- **физическая модель данных** – описывает, как хранятся данные на уровне оборудования. Оперирует такими понятиями, как сеть хранения данных, табличное пространство и т. д.

В соответствии со стандартом ANSI эта абстракция позволяет изменять некоторые из трех видов, не затрагивая остальных. Можно изменить логическую и физическую модель, оставив концептуальную неизменной. Например, какой алгоритм сортировки использовать – пузырьковый или быстрый, – для концептуальной модели данных несущественно. Как и структура отношений. Мы могли бы разделить одно отношение на несколько, применив правила

нормализации или воспользовавшись типом данных `enum` для моделирования справочных таблиц.

Модель сущность-связь

Модель **сущность-связь** (entity-relation – **ER**) относится к категории концептуальных моделей данных. В ней данные представлены в форме, понятной как заказчикам, так и разработчикам. Существуют специальные методы преобразования ER-модели в реляционную.

Концептуальное моделирование – часть **жизненного цикла разработки программного обеспечения**. Обычно этим занимаются после завершения сбора требований – к функциональности и к данным. В этот момент разработчик может нарисовать черновой вариант ER-диаграммы и описать функциональные требования с помощью диаграмм потоков данных, диаграмм последовательностей, пользовательских историй и многих других технических приемов.

На этапе проектирования разработчик базы данных должен уделять особое внимание качеству проекта, прогонять эталонные тесты для обеспечения требуемой производительности и проверять корректность пользовательских требований. Если моделируется простая система, то можно сразу приступать к кодированию. Но подходить к проектированию нужно тщательно, поскольку модель данных оказывает влияние не только на алгоритмы, но и на сами данные. Изменение модели в будущем может повлечь за собой серьезные трудности при организации переноса данных из одной структуры в другую.

При проектировании схемы базы часто бывает несколько альтернативных решений, и нужно выбирать. Перечислим некоторые распространенные ошибки.

- **Избыточные данные.** Неудачный проект базы данных грешит избыточностью. Наличие избыточных данных может стать причиной других проблем, в т. ч. несогласованности и снижения производительности. При обновлении кортежа, содержащего избыточные данные, необходимо обновить все кортежи, содержащие те же данные.
- **Переизбыток null.** В некоторых приложениях, например медицинских, данные по необходимости разрежены. Так, количество атрибутов в отношении `diagnostics` может исчисляться сотнями: жар, головная боль, насморк и т. д. Большинство из них для конкретного диагноза несущественно, но в общем случае все они нужны. Такую ситуацию можно смоделировать с помощью сложных типов данных, например JSON.
- **Сильная связанность.** В некоторых случаях сильная связанность ведет к запутанным структурам данных, которые трудно изменять. Бизнес-требования со временем меняются, и некоторые из них могут устареть. Моделирование обобщения и специализации (например, студент, обучающийся неполное время, является студентом) сильно связанным образом может стать причиной проблем.

Пример приложения

Для иллюстрации принципов модели сущность-связь рассмотрим веб-сайт для торговли автомобилями. Ниже приведены требования к этому приложению.

1. Сайт позволяет пользователю зарегистрироваться и предоставляет различные услуги в зависимости от категории пользователя.
2. Существуют продавцы и обычные пользователи. Продавец может создавать новые объявления о продаже автомобилей, прочие пользователи могут только производить поиск и просмотр.
3. В процессе регистрации любой пользователь должен указать свое полное имя и действительный адрес электронной почты. Этот адрес будет использоваться для входа в систему.
4. Продавец должен указать адрес.
5. Пользователь может оценить объявление и качество услуг продавца.
6. История поисков пользователя должна сохраняться для последующего использования.
7. Каждый продавец имеет ранг, который влияет на поиск его объявлений. Ранг определяется количеством опубликованных объявлений и рейтингами, проставленными пользователями.
8. У объявления о продаже имеется дата, а у автомобиля много атрибутов, в т. ч. цвет, количество дверей, количество предыдущих владельцев, регистрационный номер, фотографии и т. д.

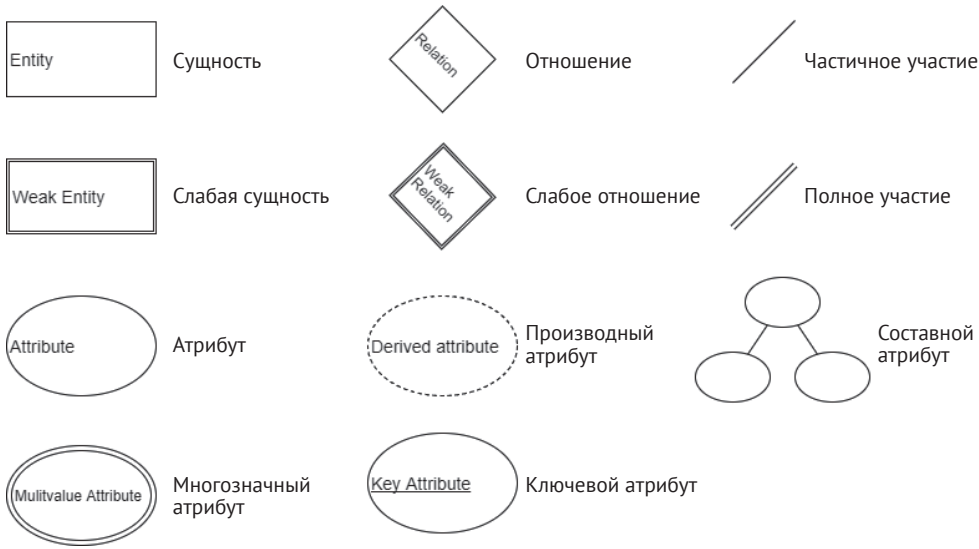
Сущности, атрибуты и ключи

На диаграмме сущность-связь представлены сущности, атрибуты и связи. Сущность соответствует объекту реального мира, например автомобилю или пользователю. Атрибут – это свойство объекта. Связь описывает ассоциацию между двумя или более сущностями.

Атрибуты могут быть простыми (атомарными) или составными. В составном атрибуте можно выделить отдельные части. Часть составного атрибута дает неполную информацию, которая в отрыве от всего остального бессмысленна. Например, адрес состоит из названия улицы, номера дома и почтового индекса. Любая его часть бесполезна без знания остальных.

Атрибуты бывают однозначными и многозначными. Цвет птицы – пример многозначного атрибута. Она может быть, например, черно-красной. У многозначного атрибута могут быть ограничения снизу и сверху на допустимое количество значений. Кроме того, одни атрибуты могут быть выведены из других, например возраст можно вывести из даты рождения. В нашем примере полный ранг продавца определяется количеством объявлений и рейтингами пользователей.

Для идентификации сущности служат ключевые атрибуты. Ключевой атрибут должен быть уникальным, но ему не обязательно соответствует первичный ключ в физической модели отношения.

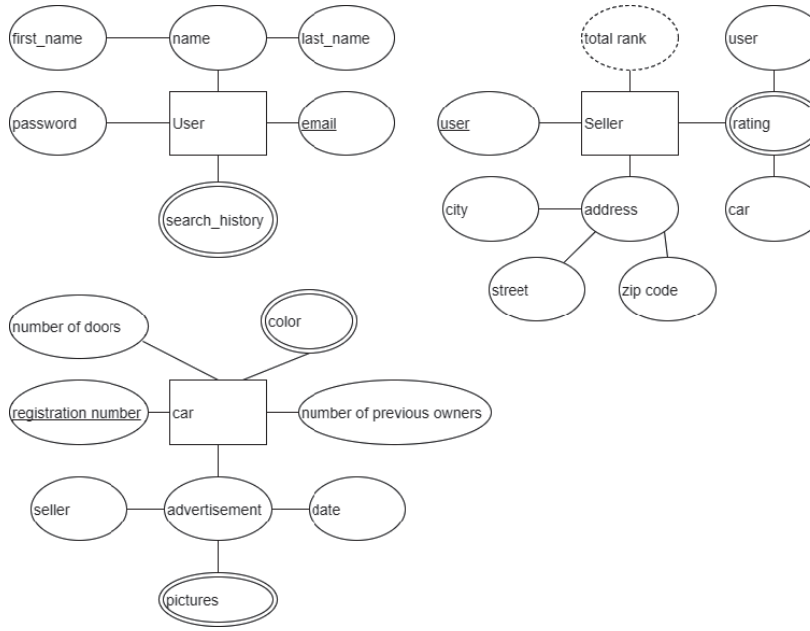


У сущности должно быть имя и набор атрибутов. Сущности бывают двух видов:

- **слабая сущность:** не имеет собственных ключевых атрибутов;
- **сильная**, или **регулярная**, **сущность:** имеет ключевой атрибут.

Слабая сущность обычно связана с какой-то сильной сущностью. Эта сильная сущность называется **идентифицирующей**. У слабых сущностей имеется частичный ключ, или дискриминатор, – атрибут, который в совокупности с идентифицирующей сущностью однозначно идентифицирует данную сущность. В нашем примере, если предположить, что при каждом поиске автомобиля пользователь задает новый ключ, ключ поиска будет частичным ключом. Символ слабой сущности – прямоугольник, ограниченный двойной линией.

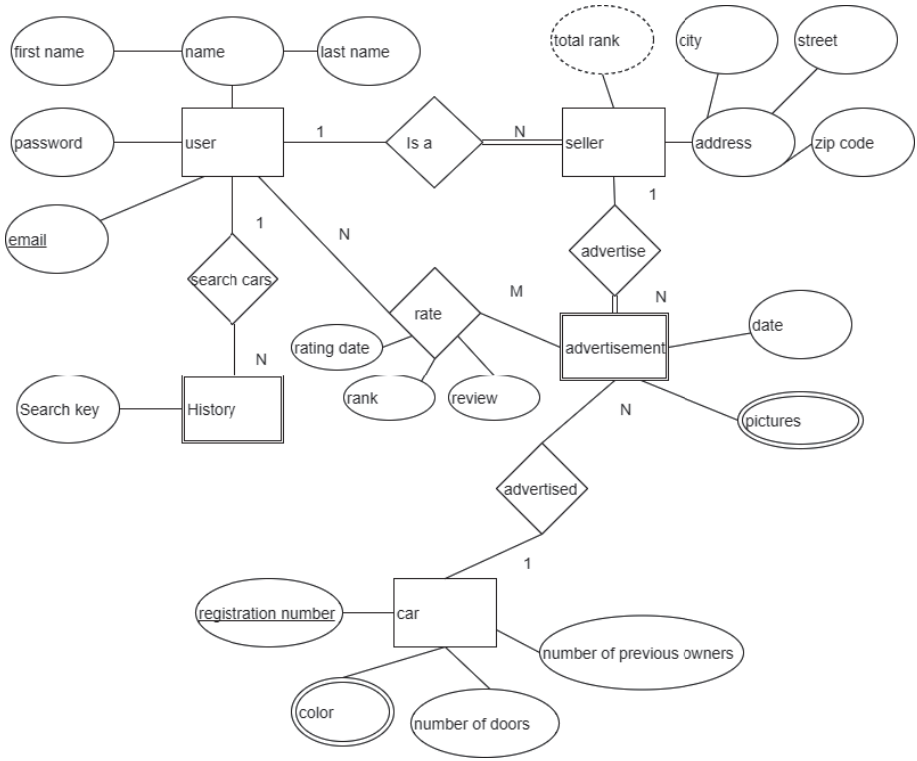
На следующем рисунке показан предварительный проект приложения. У сущности `user` несколько атрибутов. Атрибут `name` составной, а атрибут `email` ключевой. Сущность `seller` (продавец) – специализация сущности `user`. Атрибут `total rank` производный, он вычисляется путем объединения рейтингов, поставленных пользователями, и количества объявлений. Атрибут автомобиля `color` (цвет) многозначный. Пользователи могут выставлять рейтинги некоторым объявлениям продавца; это тернарное отношение, в котором участвуют три сущности: автомобиль, продавец и пользователь. Фотография автомобиля – многозначный атрибут сущности `advertisement` (объявление). Из рисунка видно, что объявления об одном автомобиле может дать несколько продавцов. Это реальная ситуация, потому что владелец может обратиться к нескольким посредникам с просьбой продать его автомобиль.



Если атрибут одной сущности ссылается на другую сущность, то образуется связь. В модели сущность-связь ссылки должны моделироваться не как атрибуты, а как связи или слабые сущности. Как и в случае сущностей, существует два вида связей: слабые и сильные. Слабая связь ассоциирует слабую сущность с другими сущностями. Связи, как и сущности, могут иметь атрибуты. В нашем примере объявление о продаже автомобиля дает продавец; дата объявления – свойство этой связи.

У связей имеется ограничение кардинальности, которое определяет, какие возможны комбинации участвующих в связи сущностей. Кардинальность связи объявления и продавца равна 1:N, т. е. каждое объявление дается одним продавцом, а каждый продавец может давать объявления о нескольких автомобилях. Связь между продавцом и пользователем называется полным участием (total participation) и обозначается двойной линией. Это означает, что продавец не может существовать сам по себе, он должен быть пользователем.

Ограничение кардинальности многие-ко-многим обозначается N:M, чтобы подчеркнуть, что количество участвующих сущностей по разные стороны связи может быть различным.



До сих пор мы обсуждали только базовые концепции диаграмм сущность-связь. Такие концепции, как нотация (min, max) в ограничениях кардинальности, тернарные (и n-арные) связи, обобщение, специализация и **расширенные диаграммы сущность-связь** (enhanced entity relation diagrams – **EER**), не обсуждались.

Отображение диаграмм сущность-связь на отношения

Правила отображения диаграммы сущность-связь на множество отношений (т. е. схему базы данных) почти прямолинейные, но не жесткие. Можно начать с моделирования сущности как атрибута, а затем уточнить связи. Атрибут, принадлежащий нескольким сущностям, можно выделить в независимую сущность. Ниже перечислены самые распространенные правила (список не полный).

- Отобразить регулярные сущности на отношения. Если у сущностей имеются составные атрибуты, то включить все части атрибутов. Выбрать один из ключевых атрибутов на роль первичного ключа.

- Отобразить слабые сущности на отношения, включить простые атрибуты и части составных атрибутов. Добавить внешний ключ для ссылки на идентифицирующую сущность. Первичный ключ обычно является комбинацией частичного и внешнего ключей.
- Если в отношении есть атрибут, связь с которым имеет кардинальность 1:1, то этот атрибут можно перенести в одну из участвующих сущностей.
- Если в отношении есть атрибут, связь с которым имеет кардинальность 1:N, то этот атрибут можно перенести в участвующую сущность со стороны, соответствующей N.
- Связям многие-ко-многим (N:M) поставить в соответствие новое отношение. Добавить внешние ключи для ссылки на участвующие сущности. Первичным ключом будет композиция внешних ключей.
- Многозначному атрибуту поставить в соответствие отношение. Добавить внешний ключ для ссылки на сущность, владеющую многозначным атрибутом. Первичным ключом будет композиция внешнего ключа и многозначного атрибута.

UML-диаграммы классов

Унифицированный язык моделирования (**UML**) – стандарт, разработанный организацией Object Management Group (OMG). UML-диаграммы широко применяются в программах моделирования. Разным видам моделирования соответствуют разные диаграммы, в том числе диаграммы классов, прецедентов, деятельности и реализации.

Диаграмма классов может описывать несколько типов ассоциаций, т. е. связей между классами. На ней можно изображать как атрибуты, так и методы. Диаграмму сущность-связь легко транслировать в UML-диаграмму классов. У диаграмм классов есть ряд достоинств:

- **обратное конструирование кода:** схему базы данных легко обратить и сгенерировать диаграмму классов;
- **моделирование объектов расширенной реляционной базы данных:** в современных реляционных базах есть ряд дополнительных типов объектов: последовательности, представления, индексы, функции и хранимые процедуры. UML-диаграммы классов способны представить такие типы.

Резюме

На проектирование системы управления базами данных оказывают влияние CAP-теоремы. Реляционные базы данных и базы данных NoSQL не соперничают, а дополняют друг друга. В одном приложении можно использовать базы данных разных типов. При этом хранилище типа «Ключ-значение» может быть использовано как механизм кеширования для повышения производительности реляционной базы данных.

На рынке преобладают реляционные и объектно-реляционные базы данных. В основе реляционных баз данных лежат понятие отношения и строгая математическая модель. Объектно-реляционные базы данных, в частности PostgreSQL, преодолевают ограничения реляционных баз данных за счет введения сложных типов данных, наследования и дополнительных расширений.

Фундаментальными для реляционных баз данных являются понятия отношения, кортежа и атрибута. Подобные базы гарантируют корректность и согласованность данных, применяя такие методы, как сущностная целостность, ограничения, ссылочная целостность и нормализация данных.

В следующей главе мы расскажем, как установить сервер и клиентские инструменты на различных платформах. Мы также познакомимся с такими средствами PostgreSQL, как встроенная поддержка репликации и развитая система типов.