Лекция 4. ДОКУМЕНТО-ОРИЕНТИРОВАННЫЕ БАЗЫ ДАННЫХ

Введение. Понятие документа. Операции над документами. Гибкость схемы данных. Проблемы обновления хранимых документов. Примеры документо-ориентированных СУБД. CouchDB. Сближение реляционных и документо-ориентированных баз данных.

MongoDB

ВВЕДЕНИЕ

За последние два десятилетия резко возросла роль реальных задач, предполагающих обработку объектов, отличающихся большим набором описываемых данных, высокой структурированностью и неоднородностью. При хранении таких данных в реляционных таблицах необходим неуклюжий промежуточный слой между объектами приложения и моделью таблиц, строк и столбцов реляционной БД. Эту расстыковку моделей иногда называют рассогласованием (impedance mismatch).

На рисунке 4.1 показана возможность выразить резюме (профиль в социальной сети LinkedIn) на языке реляционной схемы. Профиль специалиста в целом определяется уникальным идентификатором, user_id. Такие поля, как first_name и last_name, встречаются один раз для одного пользователя, так что их можно сделать столбцами в таблице users.

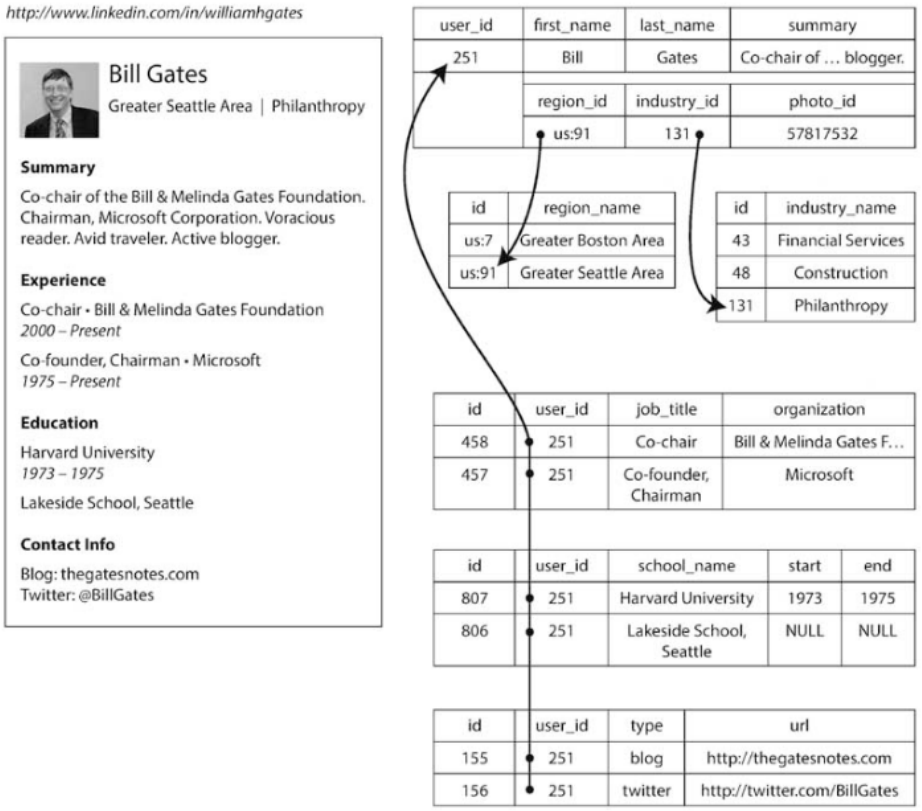

Рисунок 4.1. Представление профиля LinkedIn с помощью таблиц реляционной БД

Однако у большинства людей за время карьеры бывает больше одной работы (должности), могут иметь место различные периоды обучения и число элементов контактной информации. Между пользователем и этими элементами существует связь «один-ко-многим», которая требует специального разрешения через вспомогательные таблицы. Пример этих же данных в формате JSON показывает насколько локальнее, проще и понятнее можно представить структуру данных типа «резюме» в документо-ориентированной СУБД с поддержкой этого языка (рисунок 4.2).

```
{
"user_id": 251,
"first_name": "Bill",
"last_name": "Gates",
"summary": "Co-chair of the Bill & Melinda Gates... Active blogger.",
"region_id": "us:91",
"industry_id": 131,
"photo_url": "/p/7/000/253/05b/308dd6e.jpg",
"positions": [
  {"job_title": "Co-chair", "organization": "Bill & Melinda Gates Foundation"},
  {"job_title": "Co-founder, Chairman", "organization": "Microsoft"}
],
"education": [
  {"school_name": "Harvard University", "start": 1973, "end": 1975},
  {"school_name": "Lakeside School, Seattle", "start": null, "end": null}
],
"contact_info": {
  "blog": "http://thegatesnotes.com",
  "twitter": "http://twitter.com/BillGates"
 }
}
```

Рисунок 4.2. Представление документа «Резюме» в формате JSON

Для извлечения профиля в реляционном примере необходимо выполнить несколько запросов (по запросу к каждой таблице по user_id) или запутанное многостороннее соединение таблицы users с подчиненными ей таблицами. В JSON-представлении же вся нужная информация находится в одном месте, и одного запроса вполне достаточно. Использование XML, JSON и других подобных языков снижает рассогласование между кодом приложения и слоем хранения. Основные доводы в пользу документной модели данных – гибкость схемы, лучшая производительность вследствие локальности и большая близость к применяемым структурам данных (для некоторых приложений).

Пожалуй, наиболее важным фактором выбора базы данных является то, для чего планируется ее использовать – для хранения данных или документов. Если необходимо хранить данные, нужно выбирать базу данных, настроенную на хранение данных, т. е. реляционную или объектно-ориентированную. С другой стороны, потребность в хранении документов, заставляет выбирать

систему управления информационным обеспечением, которая специально ориентирована на хранение документов.

Поэтому следующим шагом в поддержке процессов сложно структурированных и неоднородных объектов стало появление документо-ориентированных баз данных и документно-ориентированных СУБД.

Документо-ориентированная система управления базами данных – это система, специально предназначенная для хранения иерархических структур данных, а именно документов. В основе таких систем лежат документные хранилища, имеющие структуру дерева. Структура дерева начинается с корневого узла и может содержать несколько внутренних и листовых узлов. Листовые узлы содержат данные, которые при добавлении документа заносятся в индексы, что позволяет даже при достаточно сложной структуре находить путь искомых данных.

ПОНЯТИЕ ДОКУМЕНТА

Центральным понятием документно-ориентированной базы данных является понятие *документа*. Хотя каждая реализация базы данных, ориентированной на документы, отличается деталями интерпретации этого понятия, в целом все они предполагают, что документы инкапсулируют и кодируют информацию о некотором объекте в каком-либо стандартном формате. Наиболее часто используемые кодировки: XML, YAML, JSON, а также двоичные формы, такие как BSON.

Документ задается как набор в общем случае необязательных полей, для каждого из которых определены имя и тип. Допустимо большинство стандартных типов. Это так называемые «форматные» поля, задающие числовые, символьные и другие величины, а также текстовые поля. Текстовые поля имеют переменную длину и композиционную структуру, не имеющую прямых аналогов среди стандартных типов языков программирования: текстовое поле состоит из параграфов; параграф – из предложений; предложение – из слов. При этом идентифицируемым или адресуемым элементом данных с точки зрения хранения будет поле, а с точки зрения поиска (атомарным семантически значимым) – слово.

Документы в хранилище документов примерно эквивалентны программной концепции объекта. Они не должны соответствовать жёстко заданной стандартной схеме, не обязаны иметь одинаковые разделы, слоты, или части. Как правило, ООБД имеют много разных типов объектов, и эти объекты часто имеют много необязательных полей. Каждый объект, даже принадлежащий к одному классу, может выглядеть совершенно по-разному. Документо-ориентированные хранилища похожи на ОО-хранилища в том, что они позволяют хранить разные типы документов, допускают, чтобы поля в них были необязательными, и часто позволяют их кодировать с использованием разных систем кодирования. Некоторые специалисты считают, что модели данных документных и объектно-ориентированных БД аналогичны. В чём-то

они правы, хотя разница все же есть: документо-ориентированные БД не хранят поведение объектов, а только их состояние.

Базы данных документов сильно отличаются от традиционных реляционных баз данных. Реляционные базы данных обычно хранят данные в отдельных таблицах, определенных программистом, и один объект может быть распределен по нескольким таблицам. Документальные базы данных хранят всю информацию для каждого объекта в форме единственной записи, а структура каждого хранимого объекта в принципе может отличаться от любого другого. Это устраняет необходимость объектно-реляционного сопоставления при загрузке данных в базу данных. Документы могут измениться по типу и форме в любое время.

Структура, текст и другие данные внутри документа обычно называются *содержимым документа*, и на них можно ссылаться с помощью методов поиска или редактирования. В отличие от реляционной базы данных, где каждая запись содержит одни и те же поля, а неиспользуемые поля остаются пустыми; в приведенном выше примере нет пустых «полей» ни в одном из вышеприведенных примеров документов. Такой подход позволяет добавлять новую информацию в некоторые записи, не требуя, чтобы все остальные записи в базе данных имели одинаковую структуру.

Документы адресуются в базе данных с помощью *уникального ключа*, который однозначно представляет этот документ. Этот ключ представляет собой простой идентификатор (или ID), обычную строку, URI или путь для однозначного определения местоположения в структуре БД. Ключ можно использовать для извлечения документа из базы данных. Обычно база данных сохраняет *индекс ключа* для ускорения поиска документа, а в некоторых случаях ключ требуется для создания или вставки документа в базу данных.

Key	Document
1001	{ "CustomerID": 99, "OrderItems": [{ "ProductID": 2010, "Quantity": 2, "Cost": 520 }, { "ProductID": 4365, "Quantity": 1, "Cost": 18 }], "OrderDate": "04/01/2017" }
1002	{ "CustomerID": 220, "OrderItems": [{ "ProductID": 1285, "Quantity": 1, "Cost": 120 }], "OrderDate": "05/08/2017" }

Рисунок 4.3. Пример пары «ключ – документ»

Документы могут быть организованы (сгруппированы) в *коллекции*. Их можно считать отдалённым аналогом таблиц реляционных СУБД, но коллекции могут содержать другие коллекции. Если провести аналогию с реляционными СУБД, то коллекциям соответствуют таблицы, а документам – строки в них. Хотя документы коллекции могут быть произвольными, для более эффективного индексирования лучше объединять в коллекцию документы с похожей структурой.

В следующем примере (рисунок 4.4) можно увидеть набор документов, объединённых в коллекцию «меню». Здесь breakfast, lunch и dinner выступают в качестве ключей. Обратите внимание, что, хотя все три примера содержат список блюд, состав этих блюд отображается его по-разному. Как уже отмечалось выше, в документной базе нет четкого регламента на схемы документов – документы могут иметь одинаковую или разную структуру. А группировка документов в коллекции направлена на структурирование категорий различных документов для более быстрого поиска.

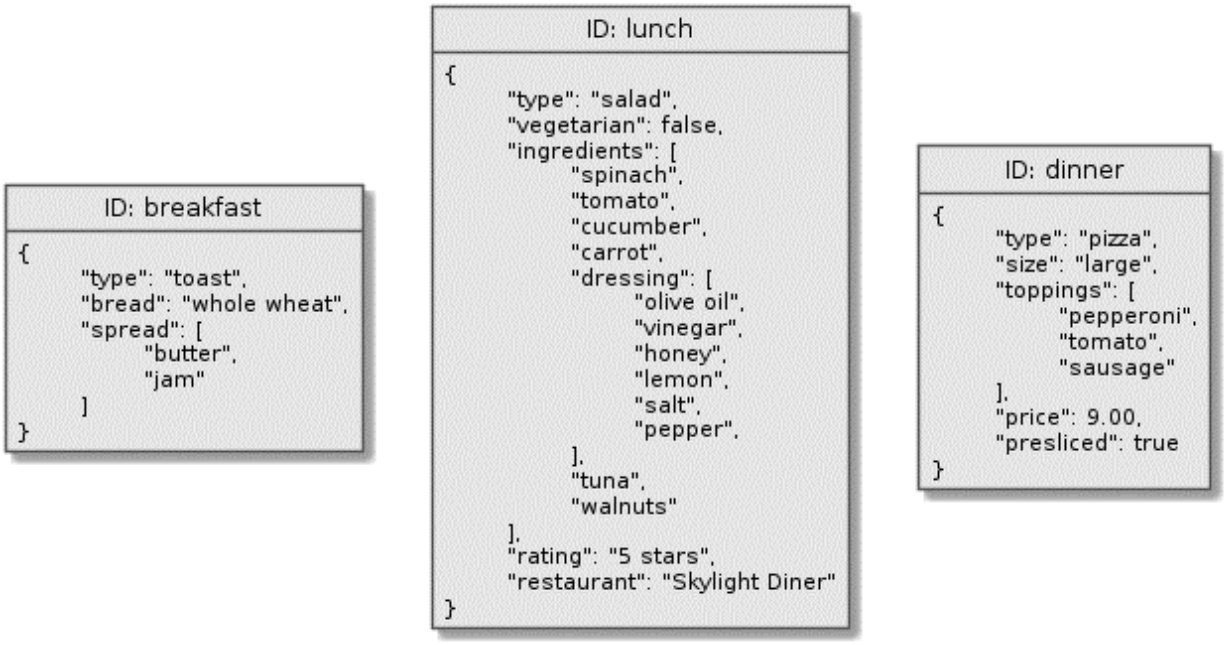

Рисунок 4.4. Представление документов коллекции «Меню»

Операции над документами

Основные операции, которые документно-ориентированная база данных поддерживает для документов, аналогичны операциям в других типах баз данных. Терминология в этой части не полностью стандартизирована, но в большинстве случаев распознают их как CRUD-операции (Create, Read, Update, Delete):

– создание (или вставка);
– извлечение (или запрос, или поиск);
– обновление (или редактирование);
– удаление.

Помимо простого поиска по ключу к документу, который можно использовать для извлечения документа, документо-ориентированные базы данных предлагают API или язык запросов, который позволяет пользователю извлекать документы на основе содержимого (или метаданные). Например, запрос, который извлекает все документы, включающие определенное поле, для которого задано определенное значение. Набор доступных API-запросов, а также ожидаемая производительность запросов значительно различаются от одной реализации СУБД к другой.

Хранилища документов используют метаданные в документе для классификации содержимого. Метаданные позволяют, например, понять, что одна серия цифр – это номер телефона, а другая – почтовый индекс. Можно выполнять поиск по этим типам данных. Например, выполнить поиск документов, содержащих в телефонном номере комбинацию цифр 222. При этом почтовый индекс 222222 будет игнорироваться.

Документо-ориентированные СУБД обычно предоставляют некоторый механизм для обновления или редактирования как содержимого документа, так и метаданных документа. Это позволяет заменять отдельные данные или отдельные структурные части документа.

Поисковые запросы

Информационный поиск в документо-ориентированных системах представляет собой поиск отдельной информации из документов или поиск целых документов, содержащих ответ на пользовательский запрос. При поиске может использоваться сходство внутренней структуры отдельных документов, представленной, например, средствами формального языка типа SGML (Standard Generalized Markup Language). Помимо текста, для поиска могут использоваться содержащиеся в статьях уравнения, таблицы и графики, хотя пока такие средства поиска разрабатываются только для узких специализированных применений (поиск по структурным химическим формулам, элементам электронных схем и т. д.). В более широком смысле, объектами текста могут являться заголовки или абзацы, примечания, ссылки, названия таблиц и т. п. В некоторых системах применяются свернутые (сигнатурные) образы документов.

Информационный запрос пользователя представляет собой конкретное значение одного или нескольких полей документа, выраженное на естественном языке. Формальное представление информационного содержания запроса называется *поисковым предписанием*. Формальное представление информационного содержания документа – это *поисковый образ документа*. Набор правил, определяющий степень смысловой близости поискового образа документа и поискового предписания называется *критерием смыслового соответствия*. На рисунке 4.4 изображена схема обработки запроса в документо-ориентированной базе данных. После отправки запроса на получение искомого документа (или набора документов) начинается поиск

этого документа в базе. Если поисковое предписание запроса совпадает с поисковым образом документа, то найденная информация передается отправителю запроса, иначе выводится соответствующее сообщение.

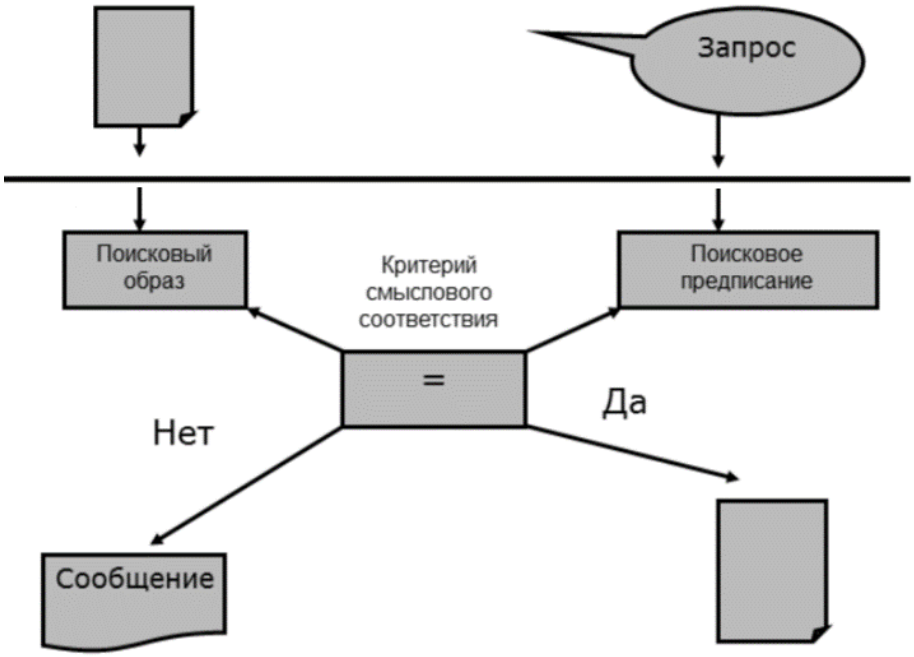

Рисунок 4.4. Схема обработки запроса в документо-ориентированной БД

Гибкость схемы данных в документе

Главное преимущество документо-ориентированной базы –это отсутствие единой схемы всех элементов. В противоположность реляционным БД, документо-ориентированные БД обычно называют *бессхемными* (schemaless) или *неструктурированными*. Но такое название может ввести в заблуждение, поскольку программный код, читающий данные, предполагает наличие у этих данных определенной структуры. Значит какая-то неявная схема все равно есть, но она не навязывается базой. Более точным было бы название *схема, определяемая при чтении* (schema-on-read): структура данных неявна и её интерпретация происходит при чтении.

В традиционном подходе реляционных БД, напротив, имеет место *схема при записи* (schema-on-write). Имеется явная и строго заданная при проектировании схема данных, и база гарантирует, что все записываемые и извлекаемые данные соответствуют этой схеме.

С другой стороны, схема, определяемая при чтении, аналогична в каком-то смысле динамической проверке типов данных во время выполнения в языках программирования, в то время как схема при записи аналогична статической проверке типов во время компиляции.

Различие между этими подходами особенно заметно в ситуациях, когда приложению необходимо изменить формат его данных. Например, допустим, что в настоящий момент мы храним адрес пользователя в одном поле, а нам потребовалось хранить страну и город отдельно. В документо-ориентированной

БД можно было бы просто начать записывать новые документы с новыми полями. А в приложении добавить код, который обрабатывает чтение документов в старом формате.

В традиционной системе с БД необходимо вносить изменение в структуру таблицы, а потом написать и выполнить код, осуществляющий разделение поля адреса и разнесение его компонентов в разные столбцы (команда UPDATE). И хотя сейчас команды типа ALTER TABLE (изменение структуры таблицы) поддерживается практически всеми СУБД и выполняется за несколько миллисекунд, в случае большой таблицы изменение её схемы могут обернуться минутами и даже часами (например, для MySQL). Оператор UPDATE на большой таблице будет выполняться медленно в любой СУБД, поскольку придется перезаписывать все строки.

Подход выявления схемы данных при чтении, а значит и использование документо-ориентированной БД предпочтителен, если структура хранимых данных разнородна, например, в таких ситуациях:

- существует множество различных типов хранимых объектов, и нет смысла помещать каждый из них в отдельную таблицу;
- структура данных определяется внешними неподконтрольными системами, способными изменяться с течением времени.

Неструктурированные документы окажутся намного более естественной моделью данных.

Проблемы обновления хранимых документов

Документы обычно хранятся в виде единых непрерывных строк, закодированных в определенных форматах. Если приложению часто требуется доступ ко всему документу, например, для визуализации веб-страницы, то локальность хранилища дает определенные преимущества. В реляционной БД данные обычно рассредоточены по нескольким таблицам, что потребует нескольких поисков по индексу для полного их извлечения, а значит, потребует больше операций дискового поиска и займет больше времени.

Локальность дает преимущество только тогда, когда требуется получить большие части документа за один раз. СУБД обычно отыскивает и загружает весь документ целиком, даже если нужен только его маленький фрагмент. Значит, в случае просмотра значительного числа больших документов память, выделенная программной системе, будет расходоваться неэкономно и затрачиваться лишнее время.

При обновлении документа, как правило, необходимо тоже переписать весь документ целиком. Однако, если изменения не меняют размер документа, то их легко выполнить «на месте». При этом физически документ в хранилище не перемещается. Если же изменения затрагивают размер хранимого объекта, документо-ориентированной СУБД придется потратить немало усилий.

Поэтому в большинстве случаев рекомендуется минимизировать размер документов и избегать увеличивающих этот размер операций записи.

ПРИМЕРЫ ДОКУМЕНТО-ОРИЕНТИРОВАННЫХ СУБД

Варианты применений

Документо-ориентированные базы данных обычно используются для аналитической обработки большого объема данных, например, на сайтах с очень высокой нагрузкой. Часто такие базы данных используются совместно с традиционными решениями на реляционных базах данных. Успешное применение документальных хранилищ отмечается в издательском деле, документальном поиске. Документная модель хорошо работает в таких примерах использования как каталоги, пользовательские профили и системы управления контентом, где каждый документ уникален и изменяется со временем. Документные базы данных обеспечивают гибкость индексации, производительность выполнения стандартных запросов и аналитику наборов документов.

Управление контентом. Документная база данных – отличный выбор для приложений управления контентом, таких как платформы для блогов и размещения видео. При использовании документной базы данных каждая сущность, отслеживаемая приложением, может храниться как отдельный документ. Документная база данных позволяет разработчику с удобством обновлять приложение при изменении требований. Кроме того, если необходимо изменить модель данных, то требуется обновление только затронутых этим изменением документов. Для внесения изменений нет необходимости обновлять схему и прерывать работу базы данных.

Каталоги. Документные базы данных эффективны для хранения каталожной информации. Например, в приложениях для интернет-коммерции разные товары обычно имеют различное количество атрибутов. Управление тысячами атрибутов в реляционных базах данных неэффективно. Кроме того, количество атрибутов влияет на производительность чтения. При использовании документной базы данных атрибуты каждого товара можно описать в одном документе, что упрощает управление и повышает скорость чтения. Изменение атрибутов одного товара не повлияет на другие товары.

MongoDB

Пожалуй, самая популярная документо-ориентированная СУБД. Данные хранятся в формате JSON/BSON. Обладает хорошим горизонтальным масштабированием, поддерживает репликации, индексы, параллельную обработку данных по технологии Map-Reduce. Написана на языке C++.

Система поддерживает ad-hoc-запросы: они могут возвращать конкретные поля документов или наборы отдельных документов. Поддерживается поиск по регулярным выражениям. Также можно настроить запрос на возвращение случайного набора результатов. Имеется поддержка

индексов.Система может работать с набором реплик, то есть содержать две или более копии данных на различных узлах вычислительной сети.

Есть официальные драйверы для основных языков программирования: Си, C++, C#, Go, Java, Node.js, Perl, PHP, Python, Ruby, Rust, Scala, Swift. Существует также большое количество неофициальных или поддерживаемых сообществом драйверов для других языков программирования и фреймворков.

Поддерживаются коллекции с фиксированным размером. Такие коллекции сохраняют порядок вставки и по достижении заданного размера ведут себя как кольцевой буфер.

Применяется в веб-разработке как хранилище операционных данных веб-страниц, например, хранение комментариев, рейтингов, профилей пользователей, сеансы пользователей. Подходит для регистрации и хранения информации о событиях; систем управления документами и контентом; электронной коммерции; игр; хранения данных мониторинга.

Лицензия: GNU AGPL – open source, использование бесплатно. Кроме того, компания MongoDB выпускает коммерческую версию СУБД, включающую дополнительные функции (например, интеграцию с SASL, LDAP, Kerberos, SNMP), инструменты управления, мониторинг и резервное копирование, а также поддержку.

CouchDB

Разработка компании Apache. Документо-ориентированная система управления базами данных с открытым исходным кодом, не требующая описания схемы данных, распространяется свободно, написана на языке Erlang. Во многом похожа на MongoDB.

Для хранения данных используется JSON, для реализации Map-Reduce-запросов – JavaScript. Возможно написание любой логики на Erlang. Одной из особенностей СУБД является поддержка репликации с несколькими ведущими узлами.

Внешний интерфейс (API) к данной СУБД построен на основе архитектуры REST, то есть сама база данных, отдельные записи, отображения и запросы – суть ресурсы, которые имеют уникальный адрес (URL) и поддерживают операции GET, PUT, POST, DELETE. Поэтому для взаимодействия с базой данных было написано много клиентских библиотек, в том числе на таких языках: JavaScript, PHP, Ruby, Python и Erlang.

CouchDB используется во многих программных продуктах и на множестве веб-сайтов для синхронизации адресов, заметок и закладок; в качестве агрегатора сообщений электронной почты, социальных сетей, систем обмена мгновенными сообщениями; для хранения динамического контента; для внутреннего использования в отделе товаров и др.

Amazon DocumentDB

Amazon DocumentDB (совместима с MongoDB) — это быстрая, масштабируемая, высокодоступная и полностью управляемая документная база данных, которая поддерживает рабочие нагрузки MongoDB. Разработчики могут использовать в Amazon DocumentDB такой же код приложения, драйверы и инструменты для запуска, управления и масштабирования рабочей нагрузки, что и в MongoDB, при этом получая высокопроизводительную, масштабируемую и готовую к работе базу данных и не тратя время на управление базовой инфраструктурой.

Сближение реляционных и документо-ориентированных баз данных

В более поздних версиях SQL-стандарта были добавлены средства поддержки структурированных типов данных и данных в формате XML. Таким образом, многоэлементные данные можно сохранять в отдельной строке с возможностью отправлять к ним запросы и совершать по ним различные виды поиска и индексации. Эти возможности стали поддерживать в разной степени БД проектировщики СУБД Oracle, IBM DB2, MS SQL Server и PostgreSQL. Тип данных JSON также стал поддерживаться в разной степени несколькими СУБД, включая IBM DB2, MySQL и PostgreSQL.

Для случая, рассмотренного в начале лекции (рис. 4.1 и 4.2) третий возможный вариант – кодирование должностей, образования и контактной информации в виде JSON- или XML-документа, сохранение его в текстовом столбце в базе данных с интерпретацией приложением его структуры и содержимого. Однако в этом случае отсутствует возможность выполнения запросов к содержимому этих кодированных столбцов. Столбец необходимо скачать целиком и осуществлять разбор и поиск средствами приложения.

КРАТКИЕ ИТОГИ

Документо-ориентированные системы управления базами данных очень быстро захватили рынок. Они имеют достаточно много преимуществ по сравнению с их предшественниками, поскольку допускают гораздо большую вложенность и сложность структуры данных.

В заключении подытожим и сгруппируем достоинства и недостатки рассмотренного типа баз данных и его СУБД.

Преимущества:

- в сравнении с реляционными базами данных лучшая производительность при индексировании больших объемов данных и большом количестве запросов на чтение;
- имеют гибкий подход к форматам хранимых данных, легко работают с неструктурированными данными;
- легче масштабируются в сравнении с SQL-решениями;
- децентрализированы, что предполагает возможность разбиения информационной базы на несколько физически распределенных баз данных;

- в отличие от SQL, при использовании отношений 1:1 и 1:M, отображаются без дополнительных таблиц и полей для связи;
- позволяют легко менять схему данных: не нужно выполнять никаких операций обновления для добавления новых полей;
- нет проблем с хранением неструктурированных данных;
- единое место хранения всей информации об объекте.

Недостатки:

- отсутствие транзакционной логики и контроля целостности в большинстве реализаций: необходимо реализовывать ее в логике приложения;
- когда используются двунаправленные связи «многие-ко-многим», то для установления связи между двумя уже существующими сущностями надо производить две операции – добавление к первой ссылку на вторую, а ко второй на первую;
- для обработки данных необходимо использование дополнительного языка программирования.