МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Воронежский государственный технический университет»

Кафедра систем автоматизированного проектирования и
информационных систем

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ «Технология разработки NoSQL решений»

для обучающихся по направлению 09.03.02 «Информационных системы и технологии», профиль «Разработка web-ориентированных информационных систем» всех форм обучения

Воронеж 2022

УДК 658.512:621(07)
ББК 30.18:85.1:34.5я7

Составители: Э. Р. Саргсян, Н. А. Рындин

Методические рекомендации по выполнению лабораторных работ по дисциплине «Технология разработки NoSQL решений» для обучающихся по направлению 09.03.02 «Информационных системы и технологии», профиль «Разработка web-ориентированных информационных систем» всех форм обучения / ФГБО ВО «Воронежский государственный технический университет»; сост.: Э. Р. Саргсян, Н. А. Рындин – Воронеж: Изд-во ВГТУ, 2021. – 33 с.

Приводится описание выполнения лабораторных работ по курсу «Технология разработки NoSQL решений» для студентов обучающихся по направлению 09.03.02 «Информационных системы и технологии», профиль «Разработка web-ориентированных информационных систем» всех форм обучения

УДК 658.512:621(07)
ББК 30.18:85.1:34.5я7

Рецензент -

Рекомендовано методическим семинаром кафедры САПРИС и методической комиссией ФИТКБ Воронежского государственного технического университета в качестве методических материалов

ВВЕДЕНИЕ

Современные технологии обработки и хранения данных не ограничиваются только реляционными данными. Непрерывное увеличение объемов обрабатываемых данных приводит к поиску новых программных и аппаратных решений для обеспечения высоких характеристик хранилищ информации. Одним из таких подходов является применение специализированных систем управления базами данных (СУБД), существенно отличающихся от ранее используемых – это нереляционные базы данных. СУБД, разрабатываемые в рамках данного подхода, получили название NoSQL СУБД. Несмотря на принципиальные отличия от ранее применяемых, представители технологии NoSQL позволяют, при грамотном применении, обеспечивать существенные преимущества по сравнению с реляционными СУБД. Подход NoSQL затронул не только программную часть информационных систем; существенно пересмотрены принципы построения центров обработки данных для использования в рамках NoSQL.

В рамках развития высокопроизводительных систем обработки данных появились сложные технологии, которые не относятся к СУБД, например технология Hadoop. Данный фреймворк представляет собой целый стек технологий, обеспечивающих слой виртуализации и сервисов для высокопроизводительных программных комплексов. Hadoop рассчитан на применение в распределенной гетерогенной среде, данная технология не предполагает применения какого-либо одного языка программирования или СУБД – это именно уровень в многослойной информационной системе.

Потребность в высокопроизводительных, распределенных и масштабируемых СУБД, вызванная объективными причинами техногенного характера, приводит к неуклонному росту востребованности специалистов по работе с данными, аналитиков в области Big Data на рынках труда практически всех стран. Большинство современных университетов предоставляют возможность подготовки специалистов по направлениям Big Data и NoSQL.

Таким образом, при проектировании высокопроизводительных информационных систем на основе баз данных необходимо учитывать тенденции и современное состояние программных и аппаратных средств обработки данных.

В рамках данного пособия рассматривается комплекс вопросов, направленных на всестороннее изучение процессов, протекающих в области Data Science. Изложение материала пособия построено по принципу «от теории к практике». Первые разделы пособия отражают исторические аспекты NoSQL-подхода; теоретические подходы, лежащие в основе науки о данных. Последние разделы посвящены изложению методов работы с высокопроизводительными системами обработки данных уровня предприятия.

Пособие позволяет учащемуся получить исчерпывающие теоретические представления о таких технологиях как Hadoop, NoSQL, а также обеспечивает магистра достаточной информацией для самостоятельных исследований в области Big Data.

РАЗДЕЛ 1. ОСНОВЫ BIG DATA

На современном этапе развития корпоративных информационных систем (ИС) продолжаются поиски гибких методологий и процессов разработки и проектирования. Подобные технологии сами непрерывно эволюционируют и имеют огромное значение для бизнеса, государственных учреждений. Важнейшим этапом проектирования ИС является создание своеобразной дорожной карты, которая определяет приоритетные этапы реализации проекта, имеющие критическое значение для проекта в целом. Организации, успешно разрабатывающие и проектирующие решения нового поколения в области IT, следуют передовым методологиям проектирования, основываясь на собственном опыте внедрения и развертывания ИС. Отдельные методологии проектирования ИС будут рассмотрены в данном разделе. С другой стороны, предприятия, использующие передовые технологии из области Big Data или Internet of Things (IoT) в рамках инициативных проектов, экспериментов или тестовых заданий, очень часто затрудняются обнаружить реальную выгоду от подобных проектов или от непосредственного внедрения новых решений. Многие пользователи не могут выявить устойчивых связей между разрозненными требованиями к эффективности внутри предприятия. В таком случае передовые IT-проекты рано или поздно достигают стадии, когда необходимо принять решение о прекращении их финансирования или о неприменимости подобных решений в рамках конкретного предприятия. К сожалению, подобные ситуации встречается на практике, но и они лишены своих положительных моментов.

В данном разделе учебного пособия мы рассмотрим передовые технологии Big Data, которые, в общепринятом случае, включают технологии традиционных хранилищ, основанных на реляционных системах управления базами данных (РСУБД), кластерах Hadoop, базах данных NoSQL и прочих решениях для управления данными.

1.1 Развитие вычислительных технологий и стратегии их реализации

В начале данного раздела рассмотрим каким образом технологии больших данных (Big Data) влились в исторический процесс развития информационных технологий.

Начнем изложение со времени, когда подобные технологии даже не были реализованы в виде проектов. На рис. 1.1 демонстрируется схема, отражающая эволюцию технологий обработки данных.

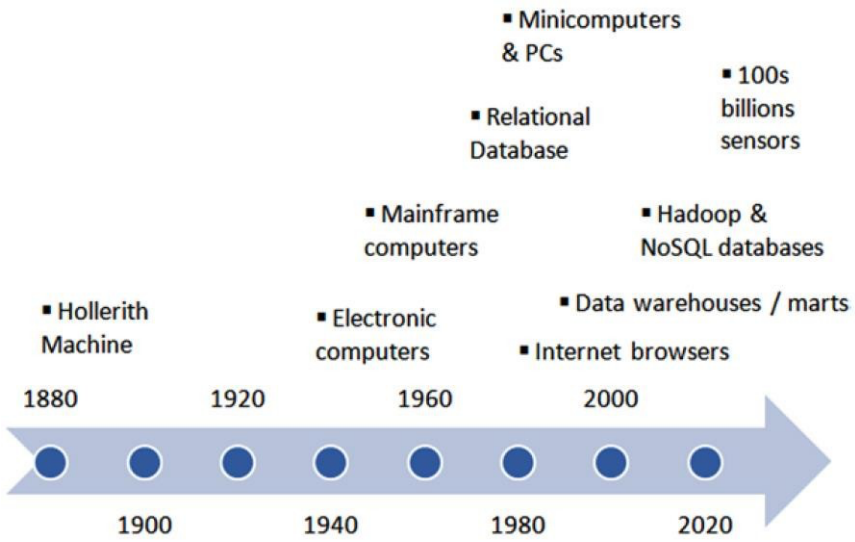


Рисунок 1.1 – Эволюция технологий обработки информации

1.1.1 Краткая история

Существует много мнений о том, каким образом описывать развитие вычислительных технологий. Данное историческое описание начинается в то время, когда вычислительной уже вышли за рамки механических вычислителей. Начнем с рассмотрения решений для обработки данных, ориентированная на предоставление информации в определенном формате. Многие считают, что основной предпосылкой к появлению табличного формата данных, стало широкое распространение оборудования на перфокартах, предложенного Германом Холлери.

Первое применение подобной технологии было реализовано при автоматизации переписи населения в США. Принципы организации данного

процесса были хорошо изучены, когда в 1880 г. Холлери предложил его решение. Многие века, государственные структуры выполняли накопление данных о различных аспектах жизни людей. В процессе такого накопления, стала проявляться структура массива данных, требуемая для хранения и обработки информации и содержащая следующие поля: имя гражданина, пол, адрес, возраст, собственность, место рождения, уровень образования и т.д. Постепенное увеличение ключевых показателей (KPI) с одновременным увеличение численности населения привели к необходимости внедрения автоматизированного подхода к хранению и обработке данных. Перфокарты Холлери обеспечивали подобную автоматизацию.

В 1930 г. данная технология стала популярной по всему миру и широко применялась для автоматизации различных процессов во всех областях.

В 1940 г., с началом Второй Мировой Войны, перед учеными встали сложные проблемы военной направленности, связанные с ускорением вычислений. К подобным проблемам относились шифрование, расчет баллистических траекторий. Необходимость автоматизации подобных решений привела к появлению электронно-вычислительных устройств, состоящих из переключателей, вакуумных трубок и проводников, заключенных в шкафы, которые заполняли комнату или здание. После войны, в продолжении исследований в военной сфере, стали появляться компьютеры для автоматизации коммерческих предприятий в области финансов и др.

Следующие десятилетия были связаны с развитием и распространением современных операционных систем, программного обеспечения, языков программирования (для упрощения и ускорения разработки приложений) и баз данных (для быстрого и простого хранения данных). Баз данных эволюционировали от иерархических к более гибким реляционным, хранящим данные в табличном виде. Таблицы связаны внешним ключом посредством введения дублирующихся столбцов. Структурированный язык запросов (SQL), вскоре стал стандартным средством доступа к реляционной базе данных.

В начале 1970-х годов разработка приложений была сфокусирована на обработке и построении отчетов по часто изменяющимся данным; подобные технологии получили название – оперативная обработка транзакций (OLTP). Разработка программного обеспечения была основана на предположении о необходимости достижения ключевых бизнес-показателей (показателей эффективности) или удовлетворении текущих экономических потребностей предприятия. Несмотря на то, что транзисторы и интегральные микросхемы существенно увеличили эффективность таких систем, стоимость вычислительных систем, мэйнфреймов и программного обеспечения оставалась слишком высокой и являлась основным сдерживающим фактором развития IT.

Ситуация изменилась с появлением дешевых мини-ЭВМ, а затем персональных компьютеров, в конце 1970-х начале 1980-х годов. Электронные таблицы и реляционные базы данных сделали доступным более гибкий анализ данных, что привело к появлению систем поддержки принятия решений. С течением времени данные стали более распределенными, появлялось понимание того, что различные подходы к сбору данных привели к противоречивым результатам в области анализа данных. На современном этапе развития технологий обработки данных возникла объективная необходимость в новых подходах к архитектуре информационных систем.

1.1.2 Хранилища данных

Билла Инмона часто называют человеком, который первым предложил технологию «хранилище данных». Он описывал хранилище данных как «предметно-ориентированная, интегрированная, энергонезависимая, и изменяемая во времени коллекция данных, предназначенная для обеспечения поддержки принятия решений». В начале 1990-х, он разработал концепцию корпоративного хранилища данных (КХД). Корпоративное хранилище рассматривалось как централизованное решение для всех исторических данных для компании. КХД представлено моделью данных в третьей

нормальной форме (3НФ), где все атрибуты являются атомарными и содержат уникальные значения. Аналогичные схемы использовались в базах данных OLTP.

Рисунок 1.2 демонстрирует небольшой фрагмент модели в третьей нормальной форме для информационного хранилища компании по продаже авиабилетов. Очевидно, что такая схема позволяет анализировать независимо транзакции отдельного покупателя, статистику мест (seat), статистику сегментов, цен, скидки для постоянных клиентов.

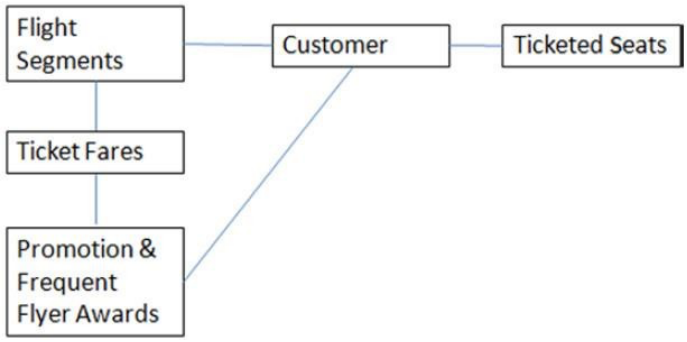


Рисунок 1.2 – Схема в третьей нормальной форме

В КХД загружаются данные из таблиц OLTP-системы через предварительные подсистемы преобразования. Преобразование используются для обеспечения согласованности данных при их извлечении из источников различного типа, а также для обеспечения поддержки бизнес-правил качества данных и стандартов. В первых хранилищах процессы извлечения, преобразования и загрузки данных между источниками и целевым хранилищем, часто выполнялась на регулярной основе (еженедельно, ежемесячно) в пакетном режиме. Но из-за потребности бизнеса видеть изменяющиеся данные в режиме реального времени, наблюдалась тенденция к все более частой загрузке данных в хранилища. Современные технологии обеспечивают загрузку данных в непрерывном режиме, без задержек (наличие задержки может быть связано со сложными алгоритмами преобразования данных). Многие организации обнаружили, что единственный способ уменьшения задержки, вызванной преобразованием данных, – это применение жестких правил к данным в первичных OLTP-системах. Этом обеспечивается

качество и согласованность данных источника, а также уменьшается сложность процедур преобразования.

Изначально большинство практических решений были ориентированы на сбор всех данных, которые могли быть размещены в хранилище, полагая, что бизнес-аналитики могут определить, что с ней делать впоследствии. Такой подход часто приводит к негативным последствиям, когда бизнес-аналитики не могли легко манипулировать необходимыми данными. Многие бизнес-аналитики просто выгружали данные из корпоративных хранилищ в электронные таблицы и уже после обрабатывали их. При этом часто эти выгрузки дополнялись данными из других источников. То етьс единого стандарта на обработку данных в хранилищах не существовало. Подобная ситуация привела к тому, что технология корпоративных хранилищ данных во многих источниках характеризовалась как неудачная и требующая переосмысления.

В рамках продолжающихся дебатов об эффективности КХД, Ральф Кимбалл представил подход, который обеспечивал бизнес-аналитикам возможность выполнения специальных запросов в более естественной понятной манере. Предложенная им схема «звезда» содержала таблицу фактов, окруженную таблицами-измерениями. Эта схема широко использовалась в витринах данных.

Рисунок 1.3 демонстрирует эффективность схемы «звезда» при реализации витрины данных на примере авиакомпании. Допустим, требуется определить всех покупателей авиабилетов из США в Мексику в июле 2014 г. Как видно из схемы, транзакции с клиентами задействуют таблицу фактов. Таблицы измерений (место отправления (origination) и назначения(destination)) содержат географическую детализацию информации (континент, страну, штат или провинцию, город, и идентификатор аэропорта). Таблица измерения «Time Dimension» позволяют определить периоды времени (год, месяц, неделя, день, час дня).

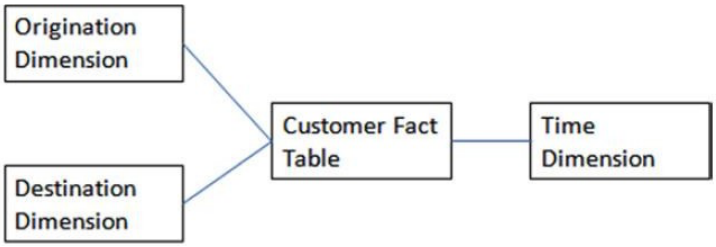


Рисунок 1.3 – Simple star schema

Не все реляционные базы данных со схемой «звезда» изначально обеспечивали оптимальную производительность. Подобные проблемы производительности привели к созданию многомерной аналитической системы обработки транзакций (MOLAP), специально предназначенных для обработки данных, организованных в виде схемы звезда. MOLAP-системы демонстрировали высокую производительность, так как были организованы в виде кубов данных. На рис. 1.4 схематично представлен трехмерный куб данных.

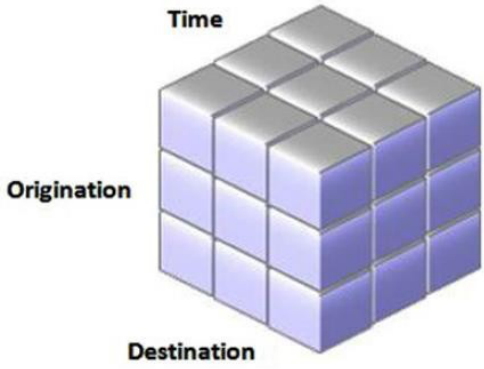


Рисунок 1.4 – Three-dimensional cube representation

Со временем, когда механизмы реляционных БД были существенно оптимизированы, были достигнуты высокие показатели производительности запросов на схемах типа «звезда». Подобные системы получили название реляционной системы аналитической обработки данных (ROLAP).

1.1.3 Зависимые и независимые витрины данных

В середине 90-х годов было множество дебатов об эффективности КХД по сравнению с витринами данных. Аналитики отметили, что в схеме «звезда» легче манипулировать данными (а также возможно разворачивать на ее основе

витрины данных); программисты стали создавать специализированные представления для КХД. Но разработка надстроек над КХД для преодоления противоречий не являлась достаточно гибким и быстрым процессом и подобные технологии не могли удовлетворить растущие потребности рынка.

Также часто возникает вторая проблема. Когда отдельные витрины данных определены и развернуты независимо друг от друга и не следуют правилам определения данных в рамках КХД, может возникнуть вопрос о месте расположения дублирующихся данных. Рисунок 1.5 иллюстрирует проблемы, которые могут возникнуть, когда различные направления бизнеса используют свои собственные независимые витрины данных и извлекают данные непосредственно из базы данных OLTP-источников. На практике, сложность реализации такой системы существенно выше, чем то, что показано на рис. 1.5, так как данные могут передаваться еще и между витринами. Также могут быть добавлены электронные таблицы, выступающие в качестве средства бизнес-аналитики. Организации, которые используют подобные системы, тратят большое количество времени на накладные сопутствующие согласования, на выявление достоверных и своевременных данных.

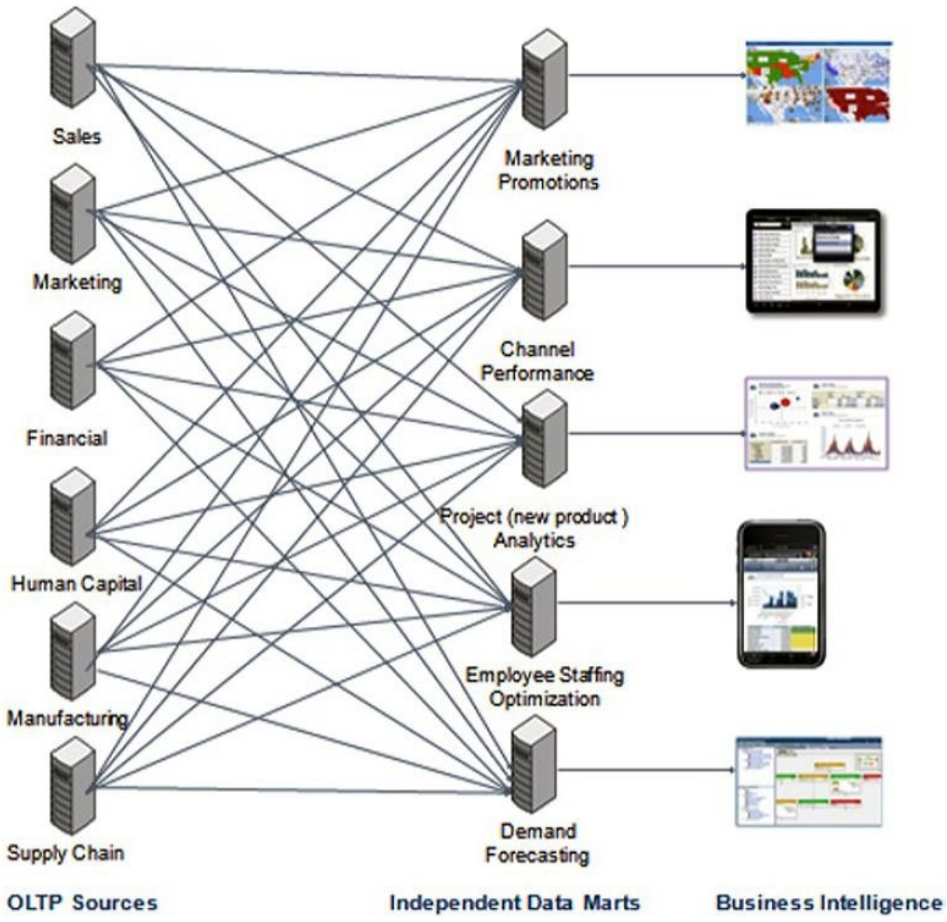


Рисунок 1.5 – Независимые витрины данных с собственными подсистемами загрузки

Существовали веские аргументы, указывающие на эффективность как витрин, так и OLAP-систем. С ростом объемов хранимых данных, аналитики сходились во мнении о необходимости применения комбинированного подхода. КХД реализовывались и расширялись постепенно по мере появления новых источников данных, также постепенно реализовывались витрины данных. Витрины данных реализовывались как системы, зависимые от КХД. КХД оставались хранилищами данных во временном измерении, тогда как данные, отображаемые в витринах извлекается из EDW. Существовало исключение, когда КХД не использовались в качестве источника – использование источника данных третьей стороны, когда они задействованы в ограниченном сегменте деятельности компании. Эти уникальные данные проецировались только на одну витрину данных. Рисунок 1.6 иллюстрирует схему взаимодействия КХД и зависимых витрин данных.

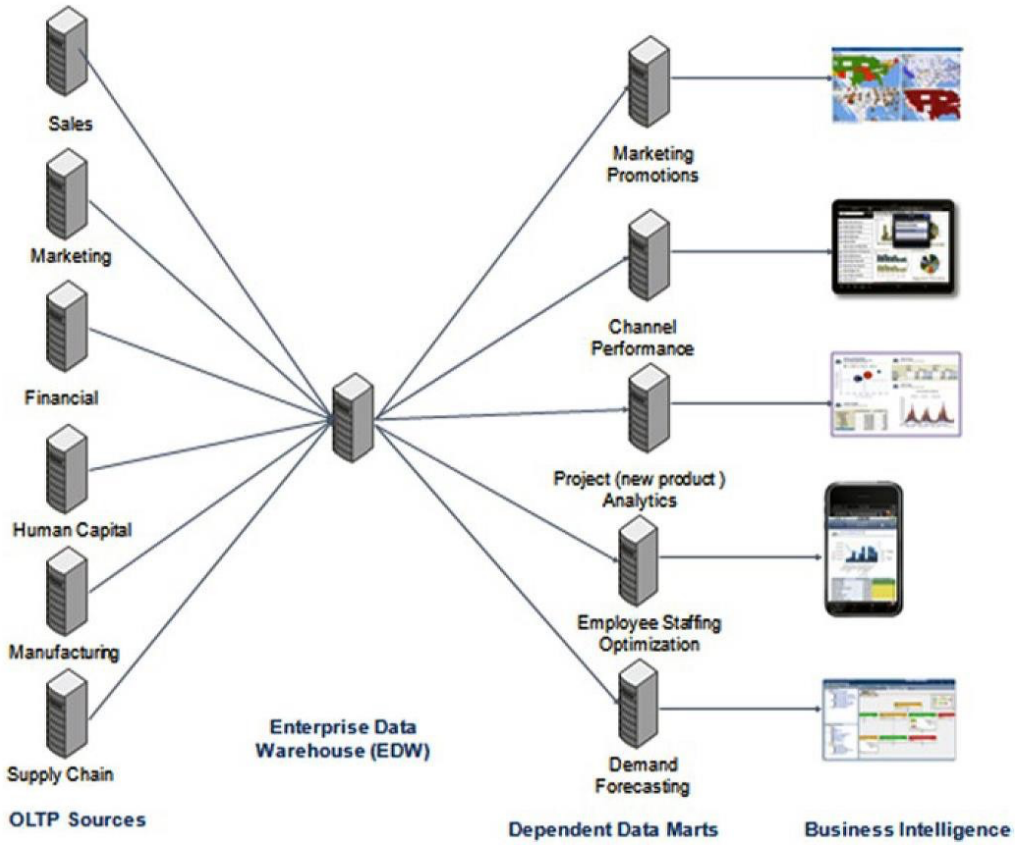


Рисунок 1.6 – Зависимые витрины данных

Такой подход часто требует определения согласованных измерений для установления согласованности между витрины данных. В таком случае согласованными измерениями, можно представить один запрос, который получает данные из нескольких витрин данных (так как измерения представляют одни и те же иерархии в различных витринах данных).

При первом использовании корпоративных хранилищ данных и витрин перед архитекторами ИС стали вопросы обеспечения доступности, отказоустойчивости и безопасности. Удовлетворение растущих требований к централизованным хранилищам приводит к расту новых возможностей и инструментов в дистрибутивах. Об этом свидетельствует растущий интерес к инструментам, направленным на повышение доступности, восстанавливаемости, безопасности в последних версиях программного обеспечения с открытым исходным кодом от различных производителей.

1.1.4 Инкрементальный подход

Первые опыты внедрения хранилищ данных укладывались в концепцию «paralysis by over analysis» (примерно «больше простоя, чем анализа»). Это связано с тем, что разработчики большое внимание уделяли проектированию системы, а не удовлетворению бизнес-требований. Проектирование КХД часто длилось более 12 месяцев и этот процесс происходил без учета требований бизнеса. При проектировании КХД применялся подход, который называется «водопад»; он предполагает фиксацию внимания разработчика на перечне работ (процедур), в то время как время разработки и затрачиваемые ресурсы являются вторичными и изменяемыми.

Схематично, концепция подхода «водопад» представлена на рис. 1.7.

Затянутые во времени планы проекта, задержки и отсутствие внимания к бизнесу часто приводило к отказу от централизованной разработки, проектированию и развертыванию независимых витрин данных, а также созданию витрин, основанных на электронных таблицах. Такое состояние было вызвано необходимостью решения текущих проблем компании.

В свете этих проблем, многие разработчики отказались от подхода «водопад» и перешли к гибкому поэтапному (инкрементальному) подходу. Инкрементальный подход учитывает тесные связи ИТ-проектировщиками и специалистами бизнеса.

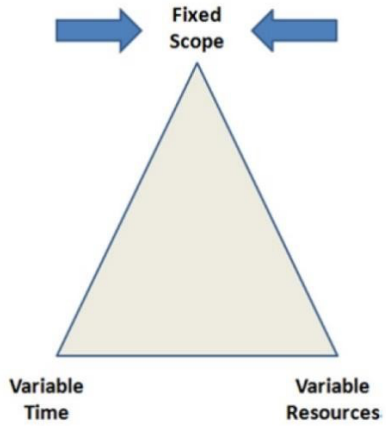


Рисунок 1.7 – Подход «водопад»

Временные рамки 120 дней или менее для реализации решения и оценки его эффективности в достижении бизнес-целей стали обычным явлением во многих организациях. На рис. 1.8 показана схема инкрементального подхода с фиксированными временными рамками и ограниченными доступными ресурсами.

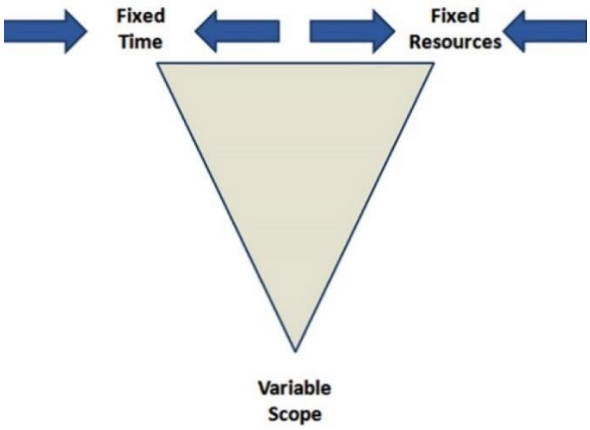


Рисунок 1.8 – Инкрементальный подход

Хотя на рис. 1.8 демонстрируется переменный объем работ, реальный инкрементальный подход предполагает наличие фиксированного показателя, коррелирующего с бизнес-целями компании на каждой итерации процесс проектирования. Таким образом, на практике, применяемая методика представляет собой сбалансированное единство пошагового подхода и подхода «водопад».

В рамках данного подхода, эффективность решения повторно оценивается на каждом шаге жизненного цикла. Определение и оценка КХД и зависимых витрин данных с все более уменьшающимся шагом означает, что ИТ-проект и разработка должны быть скорректированы, так как результаты использования смещаются от бизнес-ожиданий. Рентабельность инвестиций может быть вычислена через регулярные промежутки времени и отражать любые количественные целевые показатели.

Некоторые компании выбирают стратегию при которой бизнес-аналитики присутствуют в штате компании и непрерывно осуществляют мониторинг бизнес-требований. Другие компании обращаются к опыту

создания центров аналитической обработки, объединяющих работу бзнес-аналитиков и IT-архитекторов. Гибкость взаимодействия между этими категориями специалистов имеет более важное значение для успеха бизнеса, чем выбранный подход к проектированию КХД.

Успешные проекты Big Data, обеспечивающие новые решения в области бизнеса, часто разрабатываются с применением инкрементального подхода. В современной IT-индустрии продолжаются обсуждения передовых методик и факторов, влияющих на эффективность бизнеса.

1.1.5 Стратегии реализации

Ранее, реализация отдельного хранилища представляла собой индивидуальный проект, адаптированный к конкретной организации. Так как теоретические аспекты моделей данных не были в достаточной степени проработаны, то, фактически, модель разрабатывалась с нуля. Непредсказуемые характеристики и график нагрузки приводили к неопределенным требованиям к серверному оборудованию. Но со временем были получены результаты всестороннего изучения практического опыта использования КХД и были применены технологии, разработанные с учетом пожеланий разработчиков.

Появились технологии использования предопределенных моделей данных, разработанных на основе общих потребностей аналитиков. Эти модели были доступны для промышленного применения (производство, торговля, банковский сектор) и представлены в виде горизонтальных аналитических решений (расчет финансовых показателей, анализ продаж, маркетинговые исследования и планирование поставок). Такие модели сегодня доступны от поставщиков программного обеспечения и консалтинговых компаний, и отличаются наличием логических связей, а иногда включают в себя физическую схему. Они охватывают ключевые области бизнеса и содержат таблицы и другие элементы данных, необходимые для начала развертывания хранилища данных. Некоторые из них также

содержат с ETL-скрипты, используемые для извлечения данных из популярных ERP- и CRM-источников. Конечно, большинство организаций вынуждены настраивать модели, основываясь на их собственных уникальных бизнес-требованиях. Тем не менее, модели выступают базой для сложных проектов на основе хранилищ данных и развертываются наиболее успешно при использовании поэтапного (инкрементального) подхода.

Ранее уже было отмечено, что конфигурирование серверов и хранилища данных также как и его разработка и проектирование представляет собой самостоятельную сложную задачу.

Учитывая, что рост объемов данных происходит гораздо более быстрыми темпами, чем увеличение объемов доступного дискового пространства, множество систем обработки данных имеют существенное ограничение, если не было уделено достаточное внимание архитектуре используемого решения. В последние годы развертывание платформ, легко встраиваемых в бизнес процессы предприятия, для хранилищ данных и киосков данных стало довольно распространенным явлением. Есть несколько таких решений от производителей реляционных СУБД, которые также предоставляют серверы и системы хранения. Появление Flash-памяти в системах хранения привело к увеличению скоростных характеристик; СУБД были оптимизированы для работы с новыми типами носителей. Позднее, такие факторы как снижение стоимости памяти, появление новых процессоров, способных работать с огромными массивами оперативной памяти, и дальнейшее развитие баз данных в области совершенствования технологий хранения и извлечения данных, привело к увеличению производительности запросов по выборке данных и аналитических запросов. Все эти позволило упростить механизмы оптимизации баз данных и снизить сложность задачи проектирования.

Некоторые из ограничений, присущих серверам и системам хранения были преодолены, но неизбежно возникают другие естественные ограничения, вызванные естественными физическими рамками аппаратных систем. При

этом специалисты в области анализа данных непрерывно внедряют новые технологии и, таким образом, раздвигают рамки ограничений.

В настоящее время, количество внедрений Hadoop-кластеров и решений на базе NoSQL не так велико, но непрерывно увеличивается. Также наблюдается рост серверов высокой доступности на базе этих технологий.

Так как ценность подобных решений для бизнеса неоспорима, то все большее значение приобретают такие показатели как время проектирования и реализации, простота обслуживания. Таким образом, ожидается, что востребованность систем NoSQL и Big Data будет неуклонно расти, как в свое время наблюдался рост технологий хранилищ данных.

1.1.6 Инструментарий интеллектуального анализа данных

Каким образом осуществляется доступ к данным и как они используются целиком определяется потребностями и навыками специалистов отдельных направлений бизнеса. Для тех, кому необходимо принимать решения на основе анализа данных, рынок предлагает инструменты, которые они могут использовать могут варьироваться от простых инструментов готовой отчетности до чрезвычайно сложных инструментов анализа данных. Иногда современная инфраструктура также предоставляет «двигатели» (engines) для автоматического принятия решений и выполнения действий, а также инструменты исследования информации. Рисунок 1.9 иллюстрирует зависимость размера сообщества для определенной группы инструментов и методик от сложности этих инструментов.

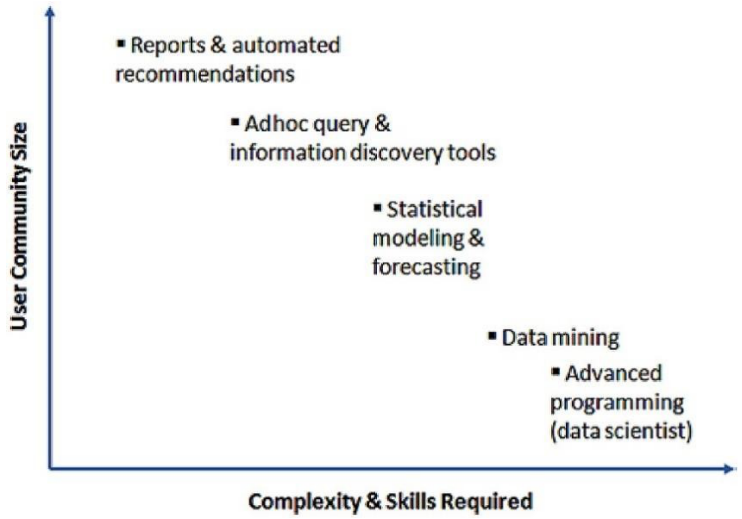


Рисунок 1.9 – Размер сообщества разработчиков и пользователей инструментария анализа данных и сложность инструментария

Самый простой способ донести информацию до бизнес-аналитика – использование предварительно определенных форм отчетов, в которых отображаются ключевые показатели эффективности. Отчеты имеют ограниченную гибкость в информации, которую можно просматривать, но они обеспечивают возможность стать потребителями информации широкому сообществу пользователей. Инструментарий отчетности, используемый разработчиками, генерирует SQL-запросы для запроса данных. Разработчики часто судят о качестве инструментальных средств построения отчетности по ясности представления показателей эффективности, а также по легкости и гибкости процедур генерации, публикации и распространения отчетов. Например, большинство шаблонов отчетов поддерживает публикацию в форматах PDF, DOC, XML и XLS.

Специальные запросы и инструменты анализа обеспечивают большую степень гибкости, так как бизнес-аналитики могут создавать свои собственные «что, если» вопросы путем самостоятельного обращения к таблицам. Разработчики могут создавать метаданные, чтобы перевести имена таблиц в значимые бизнес-ориентированные описания данных. Легкость, с которой пользователи ИС могут обращаться к данным также зависит от лежащей в

основе базы данных схемы. Как было рассмотрено ранее, схема «звезда» с измерениями и иерархическая схема являются удобными для навигации.

Типичные специализированные запросы, инструменты анализа и отчетности можно увидеть при использовании таких решений как Oracle Business Intelligence Foundation Suite, SAP Business Objects, IBM Cognos, MicroStrategy, Tableau, QlikView, Pentaho и Tibco Jaspersoft. Безусловно, многие могут сказать, что Microsoft Excel является наиболее популярным инструментом для выполнения такого рода работ, но подавляющее большинство организаций применяет широкий спектр инструментов анализа данных от различных производителей.

Некоторые специалисты бизнес-анализа имеют дело с огромными объемами данных, при этом их задача – выявить скрытые закономерности и/или спрогнозировать будущие значения целевых показателей по этим данным. Число таких специалистов неуклонно растет. Методы анализа варьируются от простых статистических расчетов, изучаемых в университете (среднее значение, стандартное отклонение, и т.д.) до сложных математических моделей, основанных алгоритмах интеллектуального анализа данных.

Статистические функции, с которыми работают бизнес-аналитики, можно разделить на следующие виды:

– простые статистические функции, такие как суммирование, сортировка, упорядочивание, построение гистограмм;

– определение плотности вероятности, дискретной вероятности;

– специальные функции (такие как гамма-функция);

– тестовые функции (такие как хи-квадрат, простая и взвешенная каппа-функция, функция корреляции).

Алгоритмы интеллектуального анализа данных используются в случае, когда необходимо понять, какие переменные имеют решающее значение для точного прогнозирования результатов и в определении прогнозных моделей, которые впоследствии будут использованы для прогнозирования результатов.

Такие модели часто применяются там, где присутствуют сотни переменных, но только десяток или меньше, которые действительно влияют на результат. Алгоритмы интеллектуального анализа данных можно классифицировать следующим образом:

– алгоритмы кластеризации: применяются для того, чтобы определить какому именно выходному подмножеству принадлежат исходные элементы;

– логические модели: применяются для определения причинно-следственных связей (и часто связаны в выведением целых деревьев решений);

– искусственные нейронные сети: обучаемые абстрактные модели, способные выполнять широкий спектр операций анализа после процедуры обучения;

– алгоритмы обнаружения аномалий: используются для выявления исключений из правил, аномальных событий.

Поставщики, осуществляющие поставки ПО для статистического анализа и интеллектуального анализа данных: SAS Institute, IBM SPSS, R (открытый проект для статистических вычислений), and Oracle Advanced Analytics.

В начале этого века, было установлено, что присутствует растущая потребность в изучении больших наборов данных, которые могут включать структурированные, полуструктурированные и потоковые данные. Рассмотренные средства анализа информации позволяют проводить анализ данных, которые не имеют предопределенной схемы. Такие инструменты обращаются, как правило, к своим хранилищам информации, например Oracle Endeca Information Discovery, они также могут быть основаны на Hadoop.

Анализируемые данные, как правило, загружаются из OLTP-источников, корпоративных хранилищ данных, NoSQL или Hadoop. Такие решения как Tibco Spotfire, Oracle Big Data Discovery и некоторые другие инструменты могут непосредственно обращаться к Hadoop и использовать данную технологию для информационного поиска. Существует ограниченный круг задач, которые требуют использования систем реального времени.

Например, могут возникнуть задачи информирования посетителя сайта о интересующих его товарах в момент посещения им сайта, или информирование о необходимости замены оборудования из-за наличия прогноза о его скором выходе из строя.

Данные о функционировании веб-сайта обычно анализируются с использованием моделей прогнозирования. Результаты предсказания обычно формируются в виде обновлений системы управления сайтом в реальном времени. Система, затем, предоставляет определенную страницу или сообщение. Чем больше функционирует сайт, тем более качественные предсказания выдает модель. Часто совместно с системой автоматического управления сайтом применяется подсистема отчетов для мониторинга состояния системы. Другие задачи, такие как предсказание отказов оборудования, могут потребовать немедленного реагирования без глубокого анализа.

Системы управления, основанные на правилах, иди системы, управляемые событиями, могут быть запрограммированы на автоматическое принятие мер в случае возникновения условного события. Они часто развертываются в IoT-решениях, чтобы выполнить действие, основываясь на показаниях датчиков.

1.2 Разработка стратегий управления данными

Изменения требований бизнеса к рабочей нагрузке провоцируют появление новых технических требований, которые влекут развитие реляционных баз данных и внедрение новых возможностей, нацеленных на удовлетворение этих требований. Несмотря на постоянное совершенствование, были выявлены недостатки реляционной архитектуры, вытекающие из модели данных, характеризуемые высокими накладными расходами при извлечении данных из таблиц. Это в значительной степени способствовало появлению баз данных в рамках подхода NoSQL и Hadoop.

Так совпало, что подобные технологии появились в то время, когда набирают популярность решения с открытым исходным кодом, что, в свою очередь, способствует развитию нереляционных подходов. В модели с открытым исходным кодом, производители и отдельные лица имеют доступ к исходному коду и эти разработчики предоставляют обновления и утилиты, доступные всем. Исходный код для баз данных NoSQL может быть лицензирован Apache Software Foundation и GNU. Лицензии Hadoop могут быть получены от Apache. Поскольку новые функции встроены в новых версиях открытым исходным кодом, производители программного обеспечения могут выбирать, что включать в их собственные дистрибутивы. Хотя дистрибутивы можно скачать бесплатно, производители считают, что они могут в конечном счете стать прибыльным и успешным компаниями, получая доходы за счет подписки (включая поддержку), а также путем предоставления услуг за отдельную плату

1.2.1 Базы данных NoSQL

Появление термины «NoSQL» относится к концу 1990-х годов, он предназначался для описания широкого класса нереляционных СУБД, предназначенных для обеспечения быстрого обновления данных и манипулирования огромными объемами данных, обеспечивая при этом горизонтальную масштабируемость. Отсутствие такого поведения в реляционных СУБД стало проблемой для некоторых интернет-приложений (таких как торговые платформы и социальные сети), где высокая производительность обновления являлась критической.

На ранних стадиях развития базы данных NoSQL не поддерживали язык SQL, оправдывая название этого класса. Но позднее поддержка SQL была в определенной степени реализована во многих NoSQL-базах данных. На ранних стадиях NoSQL-базах отсутствовали свойства, характерные для реляционных СУБД: атомарность, согласованность, изолированность и

долговечность (свойства ACID). Но поддержка подобного поведения не преследуется в NoSQL-решениях, так как влечет высокие накладные расходы и сводит на нет все преимущества. На сегодняшний день во многих NoSQL-базах данных заявлена поддержка некоторых из ACID-свойств. Тем не менее, очевидно, что новые системы не предназначены для использования в качестве замены для реляционных баз данных, OLTP или систем, предполагающих соединения многих типов данных по нескольким измерениям.

Существуют различные типы баз данных NoSQL. К ним относятся следующие:

1. Пары ключ/значение: базы данных, которые оперируют парами «ключ-значение» или «ключ-набор значений», а также характеризующиеся большим количеством легковесных операций, изменяемым числом значений, привязанных к одному ключу.

2. Ориентированные на использование столбцов (семейства столбцов): Базы данных, основанные на парах «ключ-значение» часто представляются в виде двумерного массива, поведение которого близко к классической таблице.

3. Документо-ориентированные: технология близкая к столбцовой архитектуре, но направлена на хранение данных в формате отдельных документов, при этом допускающих произвольную вложенность, произвольную структурную сложность.

4. Графовые: базы данных использующие терминологию узлов и связей.

Свойство горизонтальной масштабируемости баз данных NoSQL обеспечивается за счет механизма шардинга (sharding). Шардинг – это механизм распределения данных по нескольким независимым серверам (или узлам) в кластере. Производительность такого решения зависит от вычислительной мощности отдельного узла, но также определяется и тем, насколько эффективно произведено распределение данных между узлами. Например, если все часто используемые данные помещены на один узел, и вся активность кластера связана с обращением к часто используемым данным, то такое решение является плохо масштабированным. Многие производители баз

данных NoSQL уделяют большое внимание механизмам автоматического шардинга для обеспечения лучшей балансировки нагрузки и упрощения добавления/удаления узлов кластера.

Несмотря на то, что NoSQL-базы не так надежны в части поддержки высокой доступности, они предполагают наличие эффективных механизмов репликации, обеспечивающих функционирование БД в случае отказа узлов кластера. Копии данных, обычно, распределяются по узлам кластера, отличным от узла, на котором содержится первичный блок данных.

Существуют десятки различных баз данных NoSQL различных типов. Наиболее известные: Apache Cassandra, MongoDB, Amazon DynamoDB, Oracle NoSQL Database, IBM Cloudant, Couchbase, Inc., и MarkLogic. По мере того как список возможностей для этих баз данных может быстро измениться, понимание возможности, предоставляемые версией рассматривается для развертывания очень важно. В качестве примера, некоторые добавляют в памяти возможности только в их более поздних версиях.

Так как список особенностей подобных СУБД очень быстро меняется, понимание возможностей, предоставляемых конкретной версией, является важным моментом при развертывании системы. Например, некоторые СУБД предоставляют возможность работы с данным непосредственно в оперативной памяти (in-memory), но только в последних версиях.

1.2.2 Hadoop. История развития, особенности и инструменты

Различные потоковые данные с веб-сайтов все чаще вызывает проблемы для компаний и организаций, стремящихся проанализировать эффективность своих поисковых систем. Такие потоки данных включают в себя встроенные идентификаторы и данные значения, другие вспомогательные символы. Очевидно, что этот тип данных не вписывается в реляционную архитектуру.

Над этой проблемой работал Дугласс Каттинг (Douglass Cutting); его работы привели к разработке новой системы еще в 2002 г, которую он назвал Nutch. В 2003 и 2004 годах, компания Google опубликовала две важные статьи, описывающие файловой системы Google (GFS) и MapReduce. Понятие распределенной файловой системы не было новым в то время, но в документах Google изложено видение того, как решить проблему поиска.

Каттинг понимал важность исследований Google и реализовал изменения своей системы в соответствии со статьями компании. Технология MapReduce позволяла выполнять разделение потоков данных и уменьшать объемы обрабатываемых данных в каждом отдельном потоке. Файловая система GFS обеспечивала масштабирование, и это свойство являлось очень важным, учитывая взрывной рост количества сайтов в сети Интернет.

В 2006 г. к проекту присоединилась компания Yahoo!, технология была переименована, также был выбран символ технологии (слон). Так появился Hadoop. С этого года проект Hadoop стал проектом Apache Software Foundation.

Распределенная файловая система обеспечивает распараллеливание рабочей нагрузки для систем обработки огромных объемов данных. MapReduce обеспечивает необходимую масштабируемость. Когда комбинация этих технологий была представлена, ее описывали как решение проблемы больших объемов данных (Big Data), то есть проблемы, возникающей при обработке данных, характеризуемых большим объемом, высокой скоростью обработки и разнообразными форматами данных. Со временем, понятие Big Data приобрело более широкий смысл – так производители обозначают широкий класс средств для решения различных похожих задач.

На сегодняшний день, кластеры Hadoop считаются идеальным решением для различных типов высоконагруженных систем. Некоторые из этих кластеров в настоящее время используются в высокоскоростных ETL-

системах обработки для обеспечения параллельного преобразования между исходными системами и хранилищами данных. Другие кластеры Hadoop используются в аналитических системах прогнозирования, где используются такие технологии как R, SAS или собственная подсистема машинного обучения на базе Hadoop и библиотека интеллектуального анализа данных Mahout. Специалисты в области анализа данных иногда пишут код (используя Java, Python, Ruby on Rails и другие языки) и встраивают поддержку MapReduce или Mahout в этот код, чтобы выявить закономерности в данных. Многие разработчики организуют доступ к данным в кластере посредством различных интерфейсов SQL, таких как Hive, Impala, или других похожих инструментов.

Apache Software Foundation обеспечивает площадку для развития и создания инструментов Hadoop, которые реализуются в виде отдельных проектов. Как только появляется новый релиз, результаты нового проекта могут появиться в дистрибутивах других производителей, таких как Cloudera, Hortonworks, IBM, MapR и Pivotal. Состояние проекта Apache Hadoop отображается на сайте apache.org.

Некоторые базовые технологии из области управения данными:

1. HDFS – the Hadoop Distributed File System. Распределенная файловая система Hadoop.

2. Parquet – сжатый формат хранения данных, основанный на столбца, для системы Hadoop.

3. Sentry – система управления ролями, правами доступа, авторизацией, доступом к данным и метаданным, хранящимся в кластере Hadoop.

4. Spark – система in-memory обработки данных.

5. YARN – фреймворк для планирования и управления вычислительными задачами и ресурсами кластера Hadoop.

6. Zookeeper – сервис для управления взаимодействием распределенных приложений.

Важные инструменты для передачи и хранения данных в Hadoop:

1. Flume – сервис для сбора и агрегирования потоковых данных, включая логи и данные о событиях HDFS.

2. Kafka – платформа для обеспечения механизма публикации/подписки (Pub/sub) для обработки событий передачи данных в режиме реального времени.

3. Sqoop – инструментарий передачи данных между Hadoop и базами данных.

Hadoop включает следующие программные инструменты, API и утилиты:

1. Hive – инструментарий для предоставления возможности выборки данных в SQL-стиле из хранилища Hadoop.

2. MapReduce – это парадигма программирования, применяемая в ранних версиях Hadoop. Предполагает выполнение этапа «map» (фильтрация и сортировка) и, затем, «reduce» (агрегирующие операции) над данными, распределенными между узлами кластера.

3. Oozie – планировщик заданий, используемый при управлении задачами Hadoop.

4. Pig – фреймворк для параллельной обработки данных и язык управления потоками данных.

5. Spark GraphX – API для обеспечения просмотра данных в виде графов и коллекций, обеспечивающий преобразование и слияние графов.

6. Spark MLib – библиотека машинного обучения, реализованная для Spark.

7. Spark SQL – API для обеспечения выполнения запросов в Spark в контексте Hive.

8. Spark Streaming – специализированный API для создания процессов в Spark.

9. Solr – платформа полнотекстового индексирования и поиска.

Разработчики приложений Hadoop иногда используют возможности, предоставляемые базами данных NoSQL, которые являются частью проектируемого ими решения. HBase представляет собой базу данных NoSQL на основе столбцов, разворачиваемую на HDFS и предоставляющую доступ на чтение и запись. Подобная БД полезна при обработке больших объемов разряженных данных. В дополнение к поддержке специализированных запросов (ad hoc), HBase часто используется для получения обобщающих сводок данных.

При распределении кластера Hadoop на серверах и системах хранения требуется назначить имена узлов: узлов данных и узлов, которые будут предоставлять различные сервисы. Корректное развертывание служб кластера позволяет избежать многих проблем, например, отказа кластера Hadoop. Данные, как правило, защищены тройной репликацией, чтобы гарантировать, их доступность в случае сбоя узла.

До сих пор ведутся дебаты о целесообразности применения кластера Hadoop в качестве компонента информационной системы вместо хранилища данных на основе реляционной СУБД. В таблице 1.1 обобщена информация о достоинствах и недостатках каждого вида систем.

Таблица 1.1 –Различия между Hadoop и хранилищами данных (реляционными базами данных)

Характеристика	Hadoop	Хранилище данных
Значимость данных	Данные, как правило, смешанного качества и значимости – наиболее важный объем	Данные только высокого качества – этот параметр имеет решающее значение
Схема	Чаще всего используется распределённая файловая система	Чаще всего 3НФ и гибридная схема «звезда»
Типичная загрузка	Анализ информации, прогнозирование, ETL-обработка	Отчетность, специализированные запросы, OLAP

Источник данных	Различные типы данных от структурированных до потоковых	Структурированные типы данных
Доступность	Репликация данных по узлам кластера	Гарантированная целостность
Защищенность	Авторизация, шифрование, списки контроля доступа	Также как в Hadoop, но с более детализированным контролем доступа
Масштабируемость	Распределение по сотням узлов кластера, сотни Петабайт данных	Распределение по нескольким узлам, обычно сотни Терабайт данных или несколько Петабайт

1.2.3 Интернет вещей

В последнее десятилетие растет востребованность систем отчетности и систем анализа для данных, собранных с датчиков и управляющих устройств различного назначения. Дебаты о пользе таких решений и, одновременно, тестирование подобных систем, начались с 1980-х годов в рамках экспериментальных проектов по разработке автоматизированных устройств, подключенных к Интернет. В начале XXI века появилось множество теорий об особом виде межмашинного взаимодействия (machine-to-machine, M2M), а также о новых возможностях, ассоциированных с новым видом технологий. Появилось множество шуток типа «заговора кухонной техники». Тем не менее, вскоре стало очевидно, что последствия развития новой технологии гораздо более значимы, чем просто появление множества новых устройств; появился новый термин для описания сложившегося положения – «Интернет вещей» (Internet of Things, IoT).

Следует отметить, что еще до появления общепринятой терминологии, многие производители компьютерных устройств закладывали возможность получения данных об окружающей среде в своих продуктах, предусмотрев в их конструкции специализированные датчики и контроллеры.

Производителями также предлагались программные системы для сбора данных с датчиков. Полезность подобных устройств в бизнесе, в прошлом десятилетии, не была оценена по достоинству, а программное обеспечение для анализа данных IoT не было достаточно эффективным. Таким образом, даже в случае массового получения данных через специализированные датчики, они не анализировались, и очень часто просто игнорировались.

Но так как проектирование IT-продукции требует больших временных затрат, а сами высокотехнологичные товары обладают коротким сроком жизни («моральное старение»), то инженеры закладывали в устройства датчики и дополнительные возможности на перспективу. То есть разработчики предполагали, что рано или поздно IoT-технологии будут оценены бизнесом. Было очевидно, что данные могут помочь при анализе качества продуктов и услуг, предсказании отказов, автоматическом подключении услуг, при анализе параметров окружающей среды и автоматическом определении оптимальных параметров энергосбережения, и еще сотни применений.

Очень часто спорят о необходимости перехода на автоматические системы управления большими данными: просят привести аргументы необходимости перехода к Hadoop и технологиям Big Data. Области, которые предполагают использование датчиков и интеллектуальных контроллеров в реальных сферах бизнеса и производства, не могут обойтись без подобных систем. Наличие большого потока данных от датчиков приводит к необходимости высокопроизводительного анализа данных, получаемых с IoT-устройств.

Растущий спрос на датчики привел к интенсификации исследований в данной области и, в результате, к миниатюризации и снижению стоимости устройств. Миниатюризация связана со снижением энергопотребления. В результате этого процесса, миллиарды датчиков функционируют сегодня по всему миру, и это число, как ожидается, в ближайшее время вырастет до сотни

миллиардов устройств к 2020 г. Такая популярность подстегивает исследования в области IoT.

Параллельно с технологическим развитием, наблюдается процесс развития программного обеспечения для обработки данных в сфере IoT. Базы данных NoSQL и системы Hadoop уже повсеместно использовались в качестве ядра веб-сайтов, социальных сетей и пр., но оказалось, что эти технологии максимально эффективно удовлетворяют требованиям обработки данных от датчиков и контроллеров.

Сфера применения IoT: анализ работы двигателей, мобильные устройства, мониторинг здоровья, отслеживание почтовых перевозок и сотни других. На рис. 1.10 схематично показаны ключевые компоненты IoT.

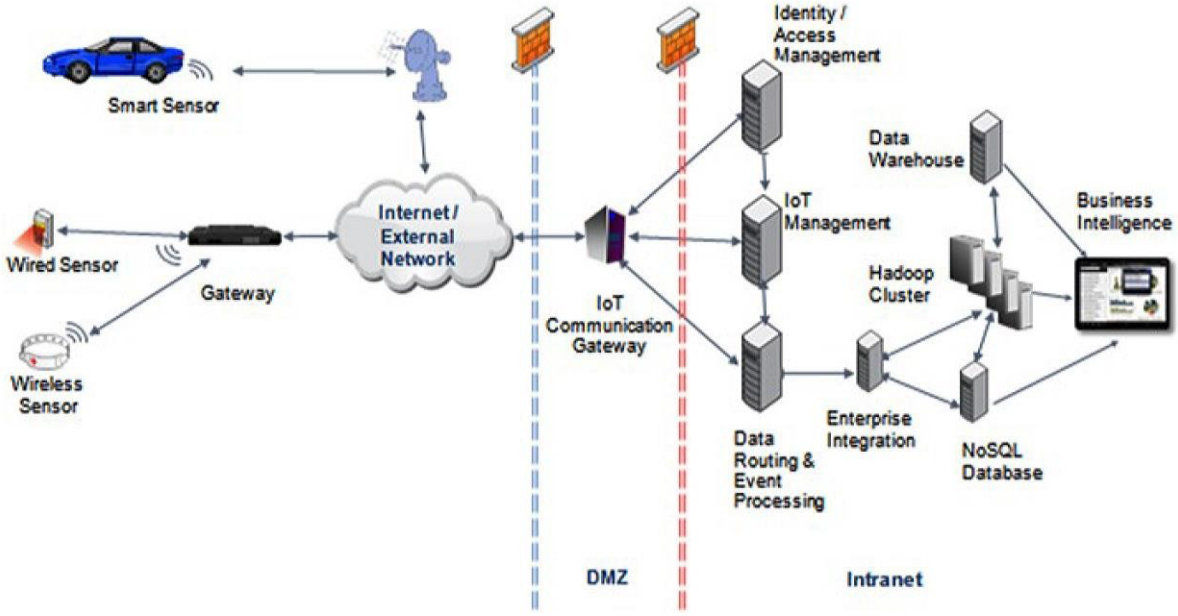


Рисунок 1.10 – Компоненты IoT

До настоящего времени в рамках пособия рассматривалось использование систем управления данными в качестве компонента для интеллектуального анализа для бизнеса. Интернет вещей вводит дополнительные требования для обеспечения маршрутизации данных, обработки событий, управления ресурсами и управления программным обеспечением для датчиков. Требуются новые технологии и средства идентификации, а также специализированные коммуникационные каналы для связи датчиков и устройств корпоративной и глобальной сетей. Несмотря на обилие проблем, самой сложной задачей является отсутствие единых стандартов в области IoT и компаний их разрабатывающих и поддерживающих. Среди существующих стандартов, применяемых в сфере IoT, можно выделить: Open Internet Consortium (архитектура взаимодействия end-to-end для IoT), IETF (описывает взаимодействие и форматы передачи), AllSeen Alliance (для маршрутизации передачи данных между устройствами), IPSO Alliance (для представления данных), Open Mobile Alliance (для управления устройствами и подключения датчиков), и Thread Group (для сетей и систем «умный дом»). На сегодняшний день Industrial Internet Consortium предпринимает попытки стандартизировать решения IoT для промышленного применения.

Компании, занимающиеся анализом данных в области IoT, предлагают свои собственные решения. Некоторые из них, проектируют и разрабатывают комплексные решения. Другие – проектируют и публикуют программное обеспечение для интеллектуального взаимодействия с сенсорными устройствами (обычно используется Java), выполняют поддержку ПО, а в итоге фокусируют свои усилия на сборе и анализе данных. Но существуют участники рынка, которые работают только в сфере анализа данных, получая их от компаний, которые занимаются передачей и непосредственным получением данных с датчиков. Материал пособия охватывает все три описанных сценария использования баз данных NoSQL и систем Hadoop для Интернета вещей. Но в пособии не рассматриваются технологии,

свойственные только для сферы IoT, затрагивающие такие аспекты как: передача сообщений, безопасность, взаимодействие и питание датчиков.

1.3 NoSQL и Big Data

Термин «NoSQL» представляет собой комбинацию слов («No» и «SQL»), что указывает на противопоставление данного подхода технологии SQL. Но на самом деле более точным термином был бы NoRDMS или NoRelational, но из-за простого произношения был выбран именно такой вариант.

Со временем, возникали предложения об использовании NonRel в качестве альтернативы NoSQL, или версии о расшифровке аббревиатуры NoSQL как «не только SQL». Каким бы ни был буквальный смысл, NoSQL используется сегодня в качестве обобщающего термина для всех баз данных и хранилищ данных, которые не реализуют популярные и хорошо разработанные принципы построения РСУБД; они часто связаны с большими наборами данных, а также с веб-технологиями. Все вышесказанное означает, что NoSQL – это не единственный продукт и не единственная технология. Это целый класс продуктов и, одновременно, семейство разнообразных, а иногда и связанных между собой, концепций хранения и манипулирования данными.

Big Data (дословно «Большие данные») – это термин, используемый для описания нашей способности анализировать постоянно растущие объемы данных в мире. Как называть данную сферу деятельности – большие данные, анализ данных, бизнес-аналитика, – не имеет существенного значения. Что действительно имеет значение, так это, что в настоящее время существуют технологии сбора и анализа данных такого уровня, который был не доступен еще несколько лет назад. Big Data привносит инновационные концепции в бизнес, промышленность, научные исследования и многие другие сферф жизни.

1.3.1 История возникновения нереляционных подходов

Нереляционные базы данных не являются новой технологией. Подобные системы хранения появились практически одновременно с первыми вычислительными машинами.

Нереляционные базы данных получили широкое распространение во времена мэйнфреймов и существовали в рамках специализированных проектов для узких практических областей, например в системах хранения исторических данных о авторизации и аутентификации. Тем не менне, нереляционные хранилища, появившиеся в рамках NoSQL-подхода, представляют собой реинкарнацию, которая учитывает изменившиеся условия IT-индустрии и широкое распространение масштабируемых Интернет-приложений. Эти хранилища, большей частью, ориентируются на распределенные и параллельные вычисления.

Начиная с Inktomi, которая может рассматриваться как первоя настоящая поисковая система, и заканчивая Google, становится очевидным, что широко распространенные РСУБД имеют существенные недостатки при обработке больших объемов данных. Эти проблемы касаются производительности, эффективности распараллеливания, масштабирования и стоимости вычислений.

Сложности применения РСУБД в масштабируемых веб-системах обработки данных не являются уникальными для данного класса программ. РСУБД предполагает наличие четкой структуры данных. Предполагается, что данные плотные и однотипные по своей природе.

РСУБД разрабатываются из расчета, что свойства данных известны заранее и взаимосвязи между элементами данных заранее известны и устойчивы. Также предполагается, что индексы могут быть последовательно определены на наборах данных, а также могут быть эффективно использованы для быстрой обработки запросов. К сожалению, эффективность применения РСУБД резко снижается, как только эти предположения оказываются не верными. При попытке адаптировать РСУБД и устранить указанные

противоречия, фактически, получаются СУБД, построенные по принципам NoSQL-подхода.

Гибкость NoSQL-решений имеет свою цену. NoSQL частично устраняет проблемы, присущие РСУБД, делает возможной работу с большими объемами разряженных данных, но, одновременно, исключает возможность использование транзакций, механизмов индексирования. Большинством производителей NoSQL-решений не были реализованы средства поддержки языка SQL, и попытки реализовать такую поддержку предпринимаются до сих пор.

Компания Google, в последние несколько лет, существенно продвинулась в разработке поискового сервиса и прочих приложений, включая Google Maps, Google Earth, Gmail, Google Finance и Google Apps. Подход Google заключается в раздельном решении проблем на различных уровня стека семейства приложений. Google нацелена на построение масштабируемой архитектуры для параллельной обработки огромного объема данных. При этом компания разработала целый набор технологий, включая распределенную файловую систему, хранилище на основе семейства столбцов, распределенную систему координации и среду параллельного выполнения программ на основе парадигмы MapReduce. Компанией Google были опубликованы работы, разъясняющие принципы функционирования ключевых технологий. Наиболее важными из них являются [6-9].

Данные публикации вызвали большой интерес в сообществе open-source-разработчиков. Так разработчики первой свободно распространяемой поисковой системы Lucene, развили некоторые идеи Google в своей версии сервиса. Впоследствии команда Lucene работала в Yahoo!, где с помощью других разработчиков, они создали «параллельную вселенную», которая копировала все элементы стека распределенных вычислений от Google. Эта свободно распространяемая технология – Hadoop, вместе со своими сопутствующими продуктами, сервисами, фреймворками и библиотеками [10].

Не будем подробно рассматривать историю развития Hadoop, но следует отметить, что первая версия продукта реализовывала принципы NoSQL. Не важно кто придумал NoSQL и когда, но важно, что появление Hadoop стало причиной взрывного развития данного направления. Одновременно, успешные проекты Google способствовали повсеместному внедрению нового поколения систем распределенных вычислений, Hadoop-систем и NoSQL-решений.

Через год после публикаций Google, явившихся катализатором развития в области параллельной масштабируемой обработки данных, компания Amazon представила результат свои успешные наработки. В 2007 году компанией Amazon была представлена идея построения распределенной системы высокой доступности Dynamo.

После успешной апробации NoSQL-технологии гигантами IT-индустрии – Google и Amazon, – стали появляться новые продукты в данном сегменте. Многие разработчики стали использовать новые походы в различных областях, на различных предприятиях: от стартапов до крупных корпораций. Технологии NoSQL стали востребованными. Менее чем за 5 лет, NoSQL и связанные с ними концепции управления большими объемами данных стали популярными и успешные варианты использования были продемонстрированы многими известными компаниями, включая Facebook, Netflix, Yahoo, EBay, Hulu, IBM и др. Многие из этих компаний также способствуют распространению их собственных расширений и свободно распространяемых продуктов в мире.

1.3.2 Базовые принципы Big Data

Рассмотрим важный вопрос: когда можно говорить, что данные являются «большими данными»? Ответ на этот вопрос зависит от того, кого вы спрашиваете и когда вы задаете этот вопрос. В настоящее время, любой набор данных размером несколько терабайт классифицируется как «большие данные». Обычно это размер, при котором набор данных достаточно большой,

чтобы возникла необходимость распределить его по нескольким узлам. Это также размер, при котором традиционные методы РСУБД демонстрируют снижение эффективности обработки.

Байт – это 8 бит. В системе СИ увеличение порядка на 3 (в 1000 раз) добавляет к единице измерения определенную приставку:

Килобайт (КБ), Kilobyte (kB) – 10^3 байт;

Мегабайт (МБ), Megabyte (MB) – 10^6 байт;

Гигабайт (ГБ), Gigabyte (GB) – 10^9 байт;

Терабайт (ТБ), Terabyte (TB) – 10^{12} байт;

Петабайт (ПБ), Petabyte (PB) – 10^{15} байт;

Эксабайт (ЭБ), Exabyte (EB) – 10^{18} байт;

Зеттабайт (ЗБ), Zettabyte (ZB) – 10^{21} байт;

Иотабайт (ИБ), Yottabyte (YB) – 10^{24} байт.

В традиционной двоичной интерпретации увеличение разрядности происходит в число раз, кратное 2, то есть умножение осуществляется на 2^{10} (или 1 024), а не на 10^3 (или 1 000). Для предотвращения двусмысленности, используется альтернативная система единиц измерения:

Кибибайт (КиБ), Kibibyte (KiB) – 2^{10} байт;

Мебибайт (МиБ), Mebibyte (MiB) – 2^{20} байт;

Гибибайт (ГиБ), Gibibyte (GiB) – 2^{30} байт;

Тебибайт (ТиБ), Tebibyte (TiB) – 2^{40} байт;

Пебибайт (ПиБ), Pebibyte (PiB) – 2^{50} байт;

Эксбибайт (ЭиБ), Exbibyte (EiB) – 2^{60} байт;

Зебибайт (ЗиБ), Zebibyte (ZiB) – 2^{70} байт;

Йобибайт (ЙиБ), Yobibyte (YiB) – 2^{80} байт.

Несколько лет назад, Терабайт данных для персонального компьютера являлся достаточно большим объемом. Сегодня, подобный объем жесткого диска для хранения данных и данных для восстановления общедоступен. Ожидается, что в ближайшие несколько лет объем жесткого диска персонального компьютера увеличится до нескольких терабайт. Сейчас мы

живем в эпоху взрывного роста объемов обрабатываемых данных. Цифровые камеры, блоги, социальные сети, твиты, электронные документы, музыка и видео – объем различных типов данных растет быстрыми темпами. Каждый человек потребляет и способствует генерации данных одновременно.

Трудно оценить истинные размеры накопленных или размер данных в сети Интернет, но некоторые исследования, приблизительные оценки показывают, что это очень большой объем, который превышает зетабайт.

Так как объем данных увеличивается, а также появляются все новые разнообразные источники данных, то вскоре человечеству предстоит столкнуться с более серьезными проблемами в данной сфере:

1. Сложная реализация эффективного хранения и организации доступа к большим объемам данных. Дополнительные требования отказоустойчивости и резервного копирования еще больше усложняет ситуацию.

2. Манипулирование большими наборами данных требует выполнения множества параллельных процессов. Реализация восстановления в случае возникновения сбоев в процессе такого параллельного выполнения в достаточно короткий период времени, является сложной задачей.

3. Одной из сложнейших проблем в области управления данных является управление данными в условиях постоянно меняющейся схемы и метаданных, работа со слабоструктурированными и неструктурированными данными, генерируемыми различными источниками.

Therefore, the ways and means of storing and retrieving large amounts of data need newer approaches beyond our current methods. NoSQL and related big-data solutions are a first step forward in that direction.

Одновременно с ростом объемов данных, происходит развитие аппаратных технологий информационных систем в сфере хранения данных. Обычные характеристики современного жесткого диска: объем 1 ТБ, скорость доступа к данным 300 Мбит/с, скорость вращения 7200 оборотов в минуту. С такими параметрами для считывания всего содержимого диска (1 ТБ)

занимает около 55 минут (очевидно, что при увеличении объема общее время считывания только вырастет). Кроме того, следует учитывать, что скорость 300 Мбит/с является пиковой, текущая скорость считывания может уменьшаться в реальных устройствах и составлять до 65% от пиковой – это связано с конструктивными особенностями HDD. Твердотельные накопители (SSD) являются альтернативой для вращающихся носителей. В SSD используются микросхемы, в отличие от электромеханических вращающихся дисков HDD. Такие диски сохраняют данные в энергозависимой оперативной памяти.

Ожидается, что SSD-накопители будут обеспечивать более высокую скорость доступа и улучшенные характеристики для операций ввода-вывода (Input/Output operations Per Second, IOPS) по сравнению с традиционными HDD. В конце 2009 и начале 2010 года, компания Toshiba анонсировала SSD-накопители, которые могут обеспечить скорость доступа более Гбит/с. Но SSD обладают и недостатками, один из них – высокая стоимость. Учитывая, что скорость доступа к диску является ограничивающим фактором, становится очевидным направление развития систем хранения: добавление в систему узлов хранения с относительно небольшим объемом, а не хранение данных на едином огромном устройстве. Таким образом, параллельно с проблемой роста объемов хранимых данных, сосуществует проблема масштабирования.

1.3.3 Масштабируемость

Масштабируемость – это способность системы увеличивать пропускную способность в случае увеличения объемов ресурсов и увеличения нагрузки. Масштабируемость может быть достигнута либо путем предоставления больших и мощных ресурсов (прежде всего аппаратных) для удовлетворения дополнительных требований, или, в альтернативном варианте, подобное поведение может быть достигнуто за счет построения кластера на базе обычных компьютеров (кластер при этом функционирует как единая ЭВМ).

Первый вариант реализации масштабирования – использование дорогих, высокопроизводительных компьютеров, – известен как вертикальное масштабирование. Использование супер-ЭВМ с многоядерными процессорами и большими дисковыми массивами, подключенными напрямую, – типичный вариант вертикального масштабирования. Такие варианты являются дорогими. Альтернатива вертикальной масштабируемости – горизонтальная масштабируемость. Горизонтальная масштабируемость предполагает объединение недорогих компьютеров в кластер, который масштабируется по мере увеличения нагрузки за счет добавления новых ЭВМ.

С появлением Big Data и необходимости параллельной обработки при манипулировании большими объемами данных привело к повсеместному внедрению горизонтально масштабируемых систем. Некоторые из этих подобных систем в компаниях Google, Amazon, Facebook, eBay и Yahoo! задействуют большое количество серверов (число которых иногда переваливает за тысячу или сотни тысяч).

Обработка данных, распределенных по узлам кластера, осуществляется параллельно. При такой реализации, модель MapReduce, возможно, обеспечивает один из наилучших подходов для обработки больших данных в горизонтальных кластерных системах.

Контрольные вопросы

1. Назовите основные факторы появления и развития Big Data.
2. Опишите области в которых возникают проблемы обработки больших объемов данных.
3. Что такое зависимые и независимые витрины данных?
4. Охарактеризуйте ключевые принципы инкрементального подхода при разработке хранилищ данных.

5. Поясните содержание терминов: «NoSQL», «Hadoop», «IoT», «Big Data».

6. Что такое концепция (правило) пяти V?

7. Перечислите направления применения технологий Big Data.

РАЗДЕЛ 2. ОСНОВЫ NOSQL

Базы данных на основе семейств столбцов – наиболее популярное решение среди нереляционных баз данных. Такие базы данных стали популярны благодаря беспрецедентным усилиям команды разработчиков Google и росту популярности социальных сетей Facebook, LinkedIn и Twitter. Эти системы стали своего рода символом NoSQL-революции.

Множество публикаций обеспечили внимание разработчиков со всего мира к технологиям поиска Google и открыли технологии крупно масштабных систем и больших данных: Google Earth, Google Analytics и Google Maps. Было установлено, что кластер недорогих машин способен управлять огромными объёмами данных гораздо эффективнее чем одна дорогая супер-ЭВМ. Были выдвинуты три ключевые концепции:

1. Необходимо хранить данные в распределенной сетевой среде, которая должна объединять несколько компьютеров. Файлы могут быть достаточно большими и должны храниться на нескольких вычислительных узлах.

2. Данные необходимо хранить в структуре, которая обеспечивает большую гибкость по сравнению с традиционными нормализованными реляционными базами данных. Схема хранения должна обеспечивать эффективное хранение огромных объемов разряженных данных. Необходимо изменять схему базы данных без удаления лежащих в ее основе таблиц.

3. Процесс вычислений должен строиться таким образом, чтобы данные могли обрабатываться отдельными изолированными подмножествами. Такое требование реализуется эффективно если программный код размещается физически на той же машине, где и обрабатываемые данные. Данный подход позволяет избежать необходимости передавать огромные объемы данных через сеть, что также увеличивает производительность.

2.1 Архитектура хранилища данных

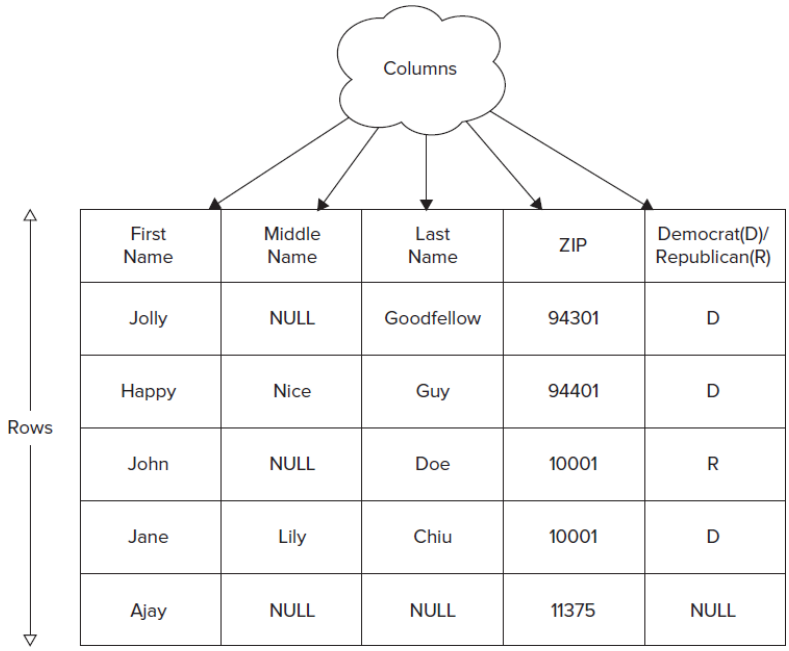
3.1.1 Столбцы в реляционных базах данных

Bigtable от Google и Apache HBase (часть фреймворка Hadoop) являются базами данных, основанными на семействе столбцов, как и Hypertable и Cloudata. Архитектура этих систем варьируется, но основные принципы построения у них общие.

Поколение разработчиков нашего времени живет в эпоху реляционных баз данных. Обучение в университете, использование на работе, публикации – все направлено на изучение, внедрение и развитие систем управления реляционными базами данных. В основе концепции РСУБД лежат отношения и связи между ними.

Современное поколение разработчиков широко применяет реляционные базы данных. Фундаментальные концепции реляционных бах данных: сущности и связи между ними, изучаются в университетах, внедряются на предприятиях и обсуждаются и исследуются. Поэтому, изложение принципов построения столбцовых баз данных начнем с рассмотрения примера на основе РСУБД. Такой подход позволит более эффективно освоить материал.

В РСУБД атрибуты отношений представляются столбцами таблиц. Столбцы определяются заранее и значения хранятся для кадой строки каждого столбца. На рис. 3.1 представлена схема реляционной таблицы, которая понятно любому разработчику.



First Name	Middle Name	Last Name	ZIP	Democrat(D)/ Republican(R)
Jolly	NULL	Goodfellow	94301	D
Happy	Nice	Guy	94401	D
John	NULL	Doe	10001	R
Jane	Lily	Chiu	10001	D
Ajay	NULL	NULL	11375	NULL

Рисунок 3.1 – Схема реляционной таблицы

В данном элементарном примере таблица содержит 5 столбцов. В РСУБД для каждого столбца определяется соответствующий тип данных. Например, для столбца «First Name» указывается тип varchar, для «ZIP» – int. В ячейках таблицы могут отсутствовать неопределенные значения (NULL-значения). Например, в примере на рис. 3.1 у Jolly Goodfellow отсутствует отчество (поле «Middle Name»).

Обычно, в таблице РСУБД определяется несколько столбцов, иногда – десятки. Внутри таблицы хранятся тысячи записей. В особых случаях таблицы могут хранить миллионы строк, но такое размещение может существенно повлиять на производительность и требуется применение специализированных механизмов оптимизации, таких как нормализация и индексирование.

В процессе использования таблицы может возникнуть ситуация, когда требуется добавить несколько столбцов, например «Street Address» и «Food Preferences». Как только добавлены новые столбцы, они не содержат значений. В ячейках будет множество NULL-значений. Такая ситуация часто встречается на практике. В этом случае, таблица будет иметь вид, представленный на рис. 3.2.

First Name	Middle Name	Last Name	Street Address	ZIP	D/R	Veg/ Non-Veg

Рисунок 3.2 – Разраженный набор данных

Предположим, что данные в таблице изменяются во времени и нам необходимо хранить различные версии данных. В таком случае можно использовать структуру напоминающую трехмерную таблиц Excel, где одним из измерений выступает время. Рисунок 3.3 демонстрирует 3-D-набор данных.

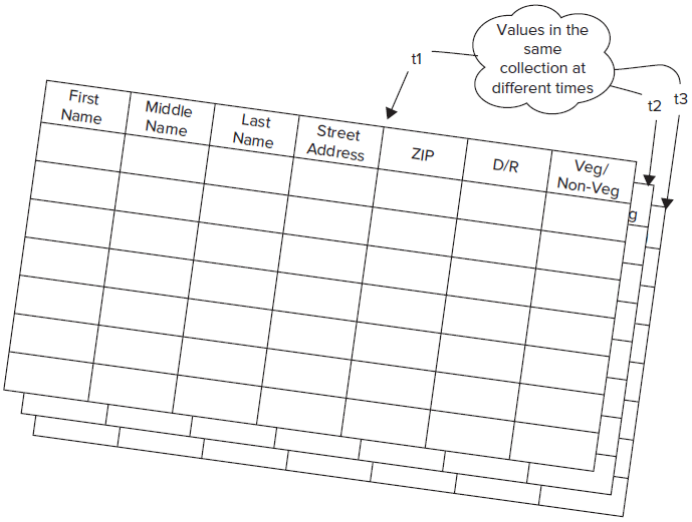


Рисунок 3.3 – 3-D-структура данных

Несмотря на то, что пример достаточно простой, вы может догадаться насколько сильно увеличиться количество пустых ячеек если в определенный момент времени все данные из таблицы будут удалены. Хранение большого количества пустых ячеек может быть спряжено с серьезными проблемаами. Точнее, проблемы связаны с особенностями РСУБД.

3.1.2 Столбцовые наборы данных и РСУБД

На основе предыдущего примера с таблицей рассмотрим столбцовые базы данных. Рассмотрение будет акцентировано на особенностях построения баз данных на основе столбцов.

Прежде всего, столбцовые базы данных отличаются отсутствием строгих требований к схеме базы данных; можно легко добавлять новые столбцы при появлении дополнительных данных. В обычном столбцовом хранилище заранее определяются семейства столбцов, а не столбцы. Семейство столбцов – это набор столбцов, сгруппированных в пакет. Столца в одном пакете логически связаны, хотя это не является обязательным требованием. Колонки семейства хранятся физически совместно и пользователь, как правило, объединяет в семейство несколько столбцов с одинаковыми характеристиками. Могут существовать различные ограничения на количество столбцов в семействе, но определение минимального количества колонок в пакете позволяет сделать схему более гибкой. В нашем примере достаточно определить три семейства: name, location и preferences.

В столбцовых базах данных семейство столбцов – это аналог столбца в РСУБД. Оба элемента определяются заранее, перед добавлением данных. Столбцы в РСУБД предполагают задание типа хранимых значений, что не является обязательным в столбцовых базах: они могут содержать любое количество столбцов, содержащих любые типы значений.

Каждая строка в столбцовых базах хранится только для тех столбцов, в которых для нее присутствуют корректные значения. NULL-значения не хранятся в базе. На рис. 3.4 показано как изменится схема данных при использовании семейств столбцов.

	name	location	preferences
	first name=>"...", last name=>"..."	zip=>"..."	d/r=>"...", veg/non-veg=>"..."

Рисунок 3.4 – Таблица в столбцовой базе данных

Кроме хорошего механизма хранения разряженных данных, столбцовые базы данных предоставляют возможность работы с данными в историческом разрезе. Поэтому постоянно развивающиеся данные будут храниться с использованием структуры, представленной на рис. 3.5.

	time	name	location	preferences
	t9	first name=>"...", last name=>"..."	zip=>"..."	d/r=>"...", veg/non-veg=>"..."
	t8			
	t7			
	t5			

Рисунок 3.5 – Хранение исторических данных

На физическом уровне данные не хранятся как единая таблица, но расположены как набор семейств столбцов. Набор столбцов спроектирован для масштабируемой архитектуры и легко может оперировать миллионами столбцов и миллиардами строк. Поэтому, одна таблица часто разделяется на множество машин. Строковый ключ идентифицирует запись в столбцовой базе. Строки упорядочены и объединены в пакеты, содержащие непрерывные значения, отражающие изменение данных во времени.

Рисунок 3.6 демонстрирует физическую схему хранения таблицы.

row-key	time	name
	t9	
	t8	
	t7	
	t5	

row-key	time	location
	t9	
	t8	

row-key	time	preferences
	t9	
	t7	

Рисунок 3.6 – Разделение таблицы на физическом уровне

Обычное решение при использовании столбцовой базы данных – применение кластера. Запуск подобного решения на единственной машине осуществляется для целей тестирования. Каждая база данных на основе семейства столбцов рассчитана на определенную типовую топологию кластера и обладает своим механизмом развертывания, но все подобные решения предполагают некоторый типовой сценарий.

3.1.3 Наборы столбцов, встроенные отображения, пары ключ/значение

Несмотря на то, что представление столбцовых баз данных в виде таблиц со специальными свойствами легко понять, это создает путаницу. Часто такие термины, как столбцы и таблицы сразу же вызывают в воображении реляционные базы данных и приводит к необходимости планирования схемы данных. Это может быть вредным и часто приводит к отказу разработчиков от использования столбцовых баз данных. Это, безусловно, одна из ловушек проектирования, которую каждый разработчик должен избегать. Всегда необходимо помнить, что корректная методика работы с инструментарием важнее самого инструментария. Если РСУБД является именно тем, что нужно в конкретном проекте, то следует использовать именно эту модель. Однако, если вы используете столбцовую базу данных, в целях масштабирования огромного хранилища данных, тогда используйте NoSQL-подход без каких-либо РСУБД.

Часто, легче представлять себе столбцовые базы как набор отображений или карт. Карты или хэш-карты, которые также являются ассоциативными массивами, представляют пары ключей и соответствующих значений.

Ключи должны быть уникальными, чтобы избежать коллизий, а значения часто являются любым массивом байтов. Некоторые карты могут содержать только строковые ключи и значения, но большинство баз данных на основе столбцов не имеют такого строгого ограничения.

Не удивительно, что Bigtable от Google, настоящий прообраз для всех столбцовых баз данных текущего поколения, официально определяется как разреженная, распределенная, устойчивая, многомерная и отсортированная карта отображения.

Рассматривая пример как многомерную вложенную карту, можно создать первые два уровня ключей в JSON-подобном представлении, например, так:

```
"row_key_1" : {
    "name" : {
    ...
    },
    "location" : {
    ...
    },
    "preferences" : {
    ...
    }
},
"row_key_2" : {
    "name" : {
    ...
    },
    "location" : {
    ...
    },
    "preferences" : {
    ...
    }
},
"row_key_3" : {
    ...
}
```

Рисунок 3.7 – JSON-подобное представление данных

Первый уровень ключа является строковым ключом, идентифицирующим запись в базе данных. Ключ второго уровня – это идентификатор уровня семейства столбцов. Три семейства столбцов были определены ранее: name, location и preferences. Следуя такому шаблону, можно определить ключ третьего уровня как идентификатор столбца. Каждая строка может охватывать различное количество столбцов внутри семейства, соответственно ключ третьего уровня может варьироваться между двумя точками многомерной карты. Добавление третьего уровня можно продемонстрировать так:

```
"row_key_1" : {
    "name" : {
        "first_name" : "Evgeny",
        "middle_name" : "Ivanovich",
        "last_name" : "Nikolaev"
    },
    "location" : {
        "zip": "355029"
    },
    "preferences" : {
        "color" : "Red"
    }
},
```

```
"row_key_2" : {
    "name" : {
        "first_name" : "Ivan",
        "middle_name" : "Sergeevich",
        "last_name" : "Novak"
    },
    "location" : {
        "zip": "355000"
    },
    "preferences" : {
        "color" : "Green"
    }
},
```

Рисунок 3.8 – Многомерная JSON-карта

В заключении, необходимо добавить версионирование элементов данных. Ключ третьего уровня может быть расширен путем добавления метки времени. В нашем примере для демонстрации такого поведения применяется целочисленная метка времени; показывается, что любимый цвет Николаева Евгения в момент времени 1 – красный, а в момент времени 2– зеленый.

Карта строки определяется следующим образом:

```
"row_key_1" : {
    "name" : {
        "first_name" : {
            1: "Evgeny"
        }
        "last_name" : {
            1: "Nikolaev"
        }
    },
    "location" : {
        "zip": {
            1: "355029"
        }
    },
    "preferences" : {
        "color" : {
            1: "Red",
            2: "Green"
        }
    }
}
```

Рисунок 3.9 – Многомерная JSON-карта с временной меткой

3.1.4 Webtable

Никакое обсуждение баз данных на основе столбцов не обходится без определяющего примера на основе Webtable – решения, которое хранит копии просмотренных веб-страниц. Такое табличное хранилище хранит содержимое

веб-страницы, а также атрибуты, которые относятся к странице. Такие атрибуты могут быть якорем, который ссылается на страницу или MIME-типы, которые имеют отношение к содержимому страницы. Google впервые ввел это решение в своей научно-исследовательской работе по Bigtable. Webtable использует инвертированные URL веб-страниц в качестве строки-ключа. Таким образом, URL www.example.com подразумевает строку ключ com.example.www. Строковые ключи ведут к упорядочиванию строк данных в столбцовой базе. Таким образом, те строки, которые относятся к двум поддоменам одного домена example.com, такие как www.example.com и news.example.com, сохраняются близко друг к другу, когда инвертированный адрес используется в качестве строки-ключа. Это позволяет проще реализовать запрос на все содержимое, относящейся к одному домену. Обычно, веб-содержимое, якоря и MIME-данные обрабатываются как семейства столбцов, что приводит к модели данных на основе таблиц и семейств столбцов (рис. 3.10).

row-key	time	contents	anchor	mime
com.cnn.www	t9		cnnsi.com	
	t8		my.look.ca	
	t6	"<html>..."	my.look.ca	"text/html"
	t5	"<html>..."		
	t3	"<html>..."		

Рисунок 3.10 – Концепция Webtable

Многие open-source реализации Bigtable включают документацию по Webtable.

2.2 Архитектура распределенного хранилища HBase

Надежная архитектура HBase заключает в себе несколько компонент. По крайней мере в основе распределенной системы лежит централизованный сервис для конфигурирования и синхронизации. Обзор архитектуры представлен на рис. 3.11.

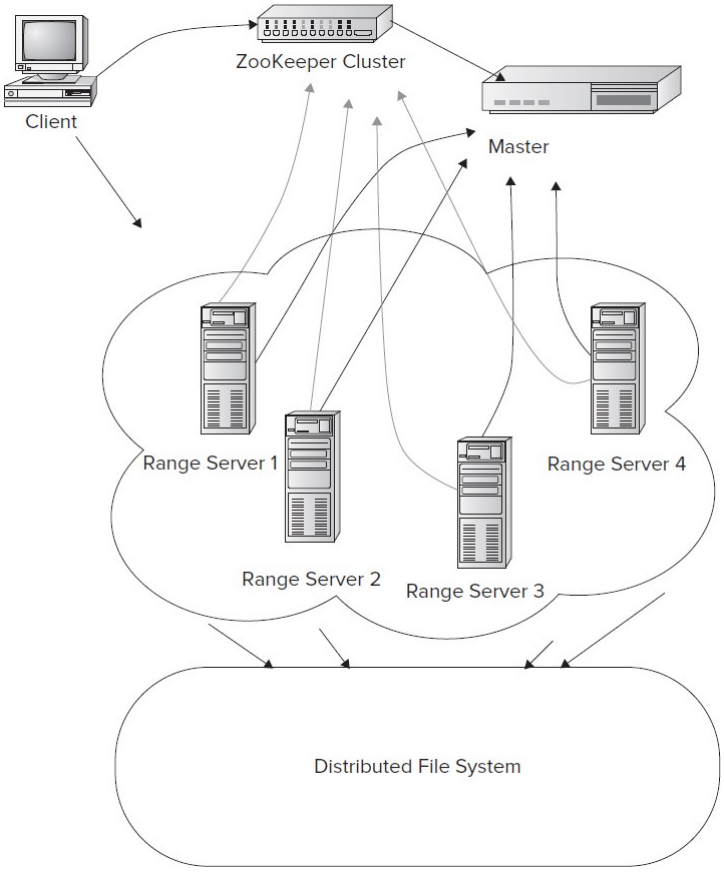


Рисунок 3.11 – Конфигурация HBase

Развертывание HBase придерживается шаблона мастер-рабочий. Поэтому, как правило, имеется мастер-процесс и набор процессов-рабочих, широко известный как сервера диапазонов. Когда HBase начинает выполняться, мастер-процесс размещает набор диапазонов на серверах диапазонов. Каждый диапазон хранит упорядоченное множество строк, где каждая строка идентифицируется уникальным строковым ключом. По мере того как число строк, хранимых в диапазоне увеличивается и превышает некоторое пороговое значение, диапазон разделяется на две части и строки делятся между двумя новыми диапазонами.

Как и большинство столбцовых баз данных, HBase хранит столбцы в семействах столбцов. Поэтому, каждая область использует отдельное

хранилище для каждого семейства столбцов таблицы. Каждое хранилище, в свою очередь, отображается на физический файл, который хранится в распределенной файловой системе. Для каждого хранилища, абстрактный уровень HBase предоставляет доступ к файловой системе с помощью тонкой прослойки сервисов, которая выступает в качестве посредника между хранилищем и физическим файлом.

Каждая область обладает своим хранилищем типа in-memory, или кэшем, и системой записи (write-ahead-log, WAL). WAL – это целое семейство технологий, обеспечивающих атомарность и долговечность (два из свойств ACID) в СУБД. WAL используется с различными СУБД, включая популярные РСУБД, такие как PostgreSQL и MySQL. В HBase разработчик может включить или отключить механизм WAL. Отключение увеличит производительность решения, но снизит надежность и возможности восстановления. Когда данные пишут в область, они сначала записываются в WAL, в случае соответствующей конфигурации. Затем, данные переписываются в хранилище in-memory для данной области. Если область памяти переполнена, то данные сбрасываются на диск и сохраняются в долговременное дисковое хранилище.

Рисунок 3.12 демонстрирует ключевые концепции использования серверов диапазонов.

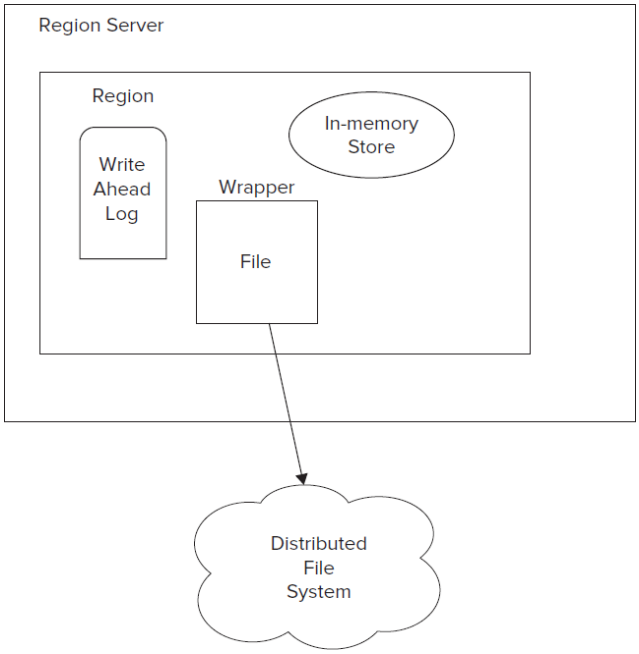


Рисунок 3.12 – Сервер диапазонов и диапазон

Если используется распределенная файловая система, как Hadoop HDFS, то шаблон мастер-работчий распространяется на базовую схему хранения. В HDFS NameNode и набор DataNodes образуют структуру, аналогичную конфигурации на основе мастер-процесса и серверов диапазонов в HBase. В такой конфигурации каждый физический файл хранилища для хранилища HBase заключается DataNode HDFS. HBase использует API файловой системы, чтобы избежать сильной связи с HDFS и таким образом API действует в качестве посредника для взаимодействия между хранилищем HBase и HDFS-файлом. API также позволяет HBase беспрепятственно работать с другими типами файловых систем. Например, HBase может использоваться с CloudStore (ранее известным как Kosmos FileSystem, KFS), вместо HDFS.

В дополнение к распределенной файловой системе для хранения, кластер HBase также использует утилиты конфигурации. В основополагающем документе по Bigtable, Google назвал эту программу конфигурации Chubby. Hadoop, будучи клоном инфраструктуры Google, использует его точный аналог Zookeeper. Zookeeper – это front-end-подсистема кластера HBase для клиентов и управления конфигурацией.

Чтобы получить доступ к HBase в первый раз, клиент получает доступ через два каталога Zookeeper. Эти каталоги называются -ROOT- и .meta. Каталоги поддерживают информацию о состоянии и местоположении всех регионов. Каталог -ROOT- хранит информацию о всех .meta.-таблиц и .meta.-файлов, хранит записи для пользовательского пространства таблицы, то есть для таблицы, которая содержит данные. Когда клиент хочет получить доступ к конкретной строке, он сначала опрашивает Zookeeper для доступа к каталогу -ROOT-. Каталог -ROOT- находит каталог .meta. для соответствующей строки, которая, в свою очередь определяет все детали для доступа к области конкретной строки. Трехступенчатый процесс доступа к строке не повторяется в следующий раз, когда клиент запрашивает данные строки повторно. Базы данных на основе столбцов в значительной степени полагаются на кэширование всей необходимой информации. Это означает, что клиенты могут обратиться непосредственно к серверу диапазонов в следующий раз, когда понадобятся данные строки. Длинный цикл поиска повторяется только если информация о области или в кэше является устаревшей или регион отключен и недоступен. Каждый диапазон часто идентифицируется наименьшим значением строкового ключа, таким образом, просматривая строку, можно определить, что строковый ключ больше или равен идентификатору диапазона.

К настоящему моменту были рассмотрены существенные концептуальные и физические модели хранения базы данных на основе столбцов. Также были рассмотрены механизмы чтения и записи данных в подобные хранилища. Расширенные функции и подробные нюансы функционирования баз данных на основе столбцов будут освещены в последующих главах.

2.3 Документо-ориентированное хранилище

MongoDB является хранилищем на основе документов, в котором документы сгруппированы в коллекции. Коллекции могут быть приближенно рассмотрены как реляционные таблицы. Тем не менее, коллекции не накладывают строгие ограничения схемы, которые приняты в отношении реляционных таблиц. Произвольные документы могут быть сгруппированы вместе в одной коллекции. Документы в коллекции должны быть структурно похожи для эффективной индексации. Коллекции могут быть разделены с помощью пространств имен, но внутренне представление не является иерархическим.

Каждый документ хранится в формате BSON. BSON является двоичным представлением формата документа JSON-типа, структура которого близка к набору пар ключ/значение. BSON является надстройкой JSON и поддерживает дополнительные типы, такие как регулярные выражения, двоичные данные и даты. Каждый документ имеет уникальный идентификатор, который MongoDB может генерировать, если он не указывается явно. Генерация происходит когда данные добавляются в коллекцию. Структура автоматически сгенерированного идентификатора показана на рис. 3.13.

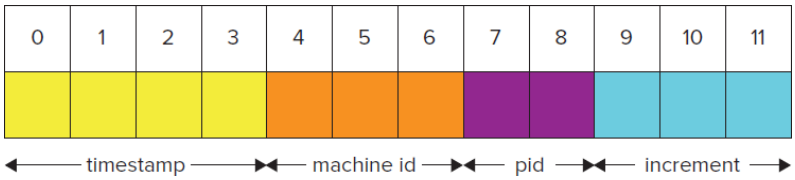


Рисунок 3.13 – Структура идентификатора

Дроайвера и клиенты MongoDB выполняют сериализацию и десериализацию BSON. Сервер MongoDB, с другой стороны, понимает формат BSON и не требует дополнительных накладных расходов на сериализацию. Двоичные представления читаются в том же формате, в каком они передаются по сети. Это обеспечивает увеличение производительности.

Высокая производительность является важной концепцией в философии MongoDB, которая построена на подобных концепциях. Одной из таких

концепций является использование mapped-memory файлов для реализации хранилища.

3.3.1 Memory-Mapped файлы

Memory-Mapped файл представляет собой сегмент виртуальной памяти, который отображен байт-в-байт на файл или файл-подобный ресурс, на который можно ссылаться через дескриптор файла. Из этого следует, что приложения могут взаимодействовать с такими файлами, как если бы они были частью первичной памяти. Это, очевидно, повышает производительность ввода/вывода по сравнению с обычным дисковым чтением и записью. Доступ и манипулирование данными в памяти гораздо быстрее, чем реализация аналогичных системных вызовов. Кроме того, во многих операционных системах, таких как Linux, отображаемые на файлы регионы памяти являются частью буфера страниц дисковой памяти в оперативной памяти. Этот буфер обычно называется кэшем страниц. Он реализован в ядре операционной системы.

Стратегия MongoDB использования памяти, отображаемой на файлы, для хранения является достаточно интеллектуальной, но не лишенной недостатков. Во-первых, memory-mapped файлы подразумевают, что не существует никакого разделения между кэшем операционной системы и кэшем базы данных. Это означает, что также отсутствует избыточность. Во-вторых, кэширование управляется операционной системой, а отображение виртуальной памяти работает не одинаково на различных операционных системах. Таким образом, политики кэширования, которые определяют, что хранится в кэше, а что нет, также варьируются от одной операционной системы к другой. В-третьих, MongoDB может расширить свой кэш базы данных, чтобы использовать всю доступную память без каких-либо дополнительных настроек. Это означает, что производительность MongoDB может быть увеличена за счет приращения объема RAM и выделения большей виртуальной памяти.

Отображение памяти также предполагает некоторые ограничения. Например, реализация MongoDB ограничивает размер данных до 2 ГБ на 32-разрядных системах. Это ограничение не актуально для 64-битных систем.

Размер базы данных ограничен не только по размеру. Дополнительные ограничения лимитируют размер каждого документа и количество коллекций сервера MongoDB. Документ не может быть больше, чем 8 МБ. Это, очевидно, означает, что использование MongoDB для хранения больших элементов не подходит. При хранении больших документов необходимо использовать GridFS. Кроме того, существует ограничение на количество пространств имен, которые могут быть определены в экземпляре базы данных. По умолчанию, количество пространств имен – 24000. Каждая коллекция и каждый индекс использует пространства имен. Соответственно, два индекса на коллекцию позволит организовать максимум 8000 коллекций на одну базу данных. Как правило, такое количество достаточно. Тем не менее, если нужно, то можно увеличить количество пространств имен до числа, превышающего 24000.

Увеличение размера пространства имен влечет определенные последствия. Каждое пространство имен коллекций использует несколько килобайт. В MongoDB индекс реализован в виде B-дерева. Каждая страница B-дерева имеет размер 8 КБ. Таким образом, добавление дополнительных пространств имен, для коллекций или индексов, предполагает добавление несколько килобайт для каждого дополнительного экземпляра. Пространства имен для имени MyDB базы данных MongoDB сохраняются в файле с именем mydb.ns. Файл mydb.ns может вырасти до максимального размера 2 ГБ. Ограничение размера mydb.ns ограничивает и рост базы данных – понимание этого намного важнее чем знание моделей поведения коллекций и индексов.

3.3.2 Коллекции и индексы MongoDB

Нет формулы, которая определяла бы оптимальное количество коллекций в базе данных, но желательно избегать создания коллекций, объединяющих разрозненных данные в единую коллекцию. Такое хранение

создает сложности для индексирования. Одним из правил является следующее: дайте ответ на вопрос «Часто ли вам требуется запрашивать из коллекции разрозненные данные?». Если ответ «Да», то данные целесообразно сохранить вместе в одной коллекции.

Иногда коллекция может расти очень быстро и создается угроза превышения лимита в 2 ГБ. Тогда возникают предпосылки для использования ограниченных коллекций (capped collections). Ограниченная коллекций в MongoDB как стек, который имеет предопределенный размер. Когда сборную коллекция превышает свой лимит, старые записи коллекции будут удалены. Старые записи определяются в соответствии с алгоритмом LRU (Least Recently Used). Документ, выбираемый из ограниченной коллекции, следует алгоритму LIFO (Last-In-First-Out).

Поле _id индексирует каждую коллекцию MongoDB. Дополнительно, программист может создавать индексы по любым другим атрибутам документа. При запросе документы выбираются в соответствии с их порядком, определяемым по _id. Только ограниченные коллекции используют порядок доступа LIFO.

Курсоры возвращают данные в виде пакетов, каждый ограничен размером 8 МиБ. Обновление записей происходит локально. MongoDB предоставляет повышенную производительность, но это достигается за счет снижения надежности.

3.3.3 Надежность и долговечность

Прежде всего MongoDB не всегда реализует принцип атомарности, а также не определяет транзакционный механизм или уровни изоляции в процессе параллельных операций. Только определенный класс операций, операции модификации, предполагают атомарную согласованность.

MongoDB определяет несколько операций модификаций для атомарного обновления (таблица 3.1).

Table 3.1 – Операции модификации MongoDB

Name	Description
$inc	Инкрементация значения определенного поля
$set	Установка значения поля
$unset	Удаление поля
$push	Добавление значения к текущему значению поля
$pushAll	Добавление значения к массиву значений полей
$addToSet	Добавление значения в массив (успешно только при отсутствии такого значения в массиве)
$pop	Удаление последнего элемента в массиве
$pull	Удаление всех вхождений значения в массиве
$pullAll	Удаление всех значений, указанных в массиве, из значения поля
$rename	Переименование поля

Отсутствие уровней изоляции также иногда приводит к проблеме чтения фантомных данных. Курсоры не обновляются автоматически в случае изменения данных, на которых они были получены. По умолчанию, MongoDB сбрасывает на диск данные раз в минуту. Данные записывают на диск, когда происходит вставка или обновление. Любой сбой между двумя синхронизациями может привести к несогласованности. Можно настроить MongoDB таким образом, чтобы повысить частоту синхронизации, но это приведет к снижению производительности.

Для обеспечения отказоустойчивости применяется механизм репликации. Два экземпляра MongoDB могут быть развернуты в конфигурации master-slave (ведущий-ведомый) и осуществлять репликацию и поддерживать данные синхронизированными. Репликация – это асинхронный процесс, поэтому изменения не применяются мгновенно.

Тем не менее, лучше использовать механизм репликации, чем не реализовать ни одного доступного механизма поддержки стабильности. В текущей версии MongoDB пара реплик master и slave была заменена набором реплик, содержащим три реплики. Одна реплика из набора играет роль master, две другие – slave. Набор реплик позволяет реализовать автоматическое восстановление и автоматическое переключение на исправные узлы.

Если репликация рассматривается в качестве механизма восстановления после сбоев, то шардинг (sharding) связан с горизонтальным масштабированием систем MongoDB.

3.3.4 Горизонтальное масштабирование

Одна из главных причин использования MongoDB – это отсутствие необходимости задавать схему, высокая производительность и масштабируемость. В более поздних версиях MongoDB поддерживает механизм автоматического шардинга для простого горизонтального масштабирования.

Идея шардинга схожа с концепцией мастер-рабочий в столбцовых базах данных, когда данных распределеяются на множестве серверов диапазонов В MongoDB упорядоченные коллекции можно сохранять на множестве машин кластера. Каждая машина, на которой хранится часть коллекции, называется шардом. Шарды, таким образом, реплицированы для обеспечения отказоустойчивости. Таким образом, большая коллекция может быть разделена по 4 шардам и каждый шард реплицируется 3 раза. Такое размещение создает 12 узлов на сервере MongoDB. Две дополнительные копии каждого шарда играют роль узлов восстановления.

Шарды – это понятие уровня коллекций, а не уровня базы данных. Поэтому дона коллекция в базе данных может размещаться га единственном узле, тогда как другая коллекция может быть разнесена по нескольким узлам.

Каждый шард хранит последовательный набор упорядоченных документов. Такой набор называется чанк (chunk). Чанк – это новый термин в области баз данных, но в MongoDB это устоявшийся жаргонный термин. Каждый чанк идентифицируется тремя атрибутами: ключ первого документа (min key), ключ последнего документа (max key) и коллекция.

Коллекция может быть шардирована на основе любого паттерна с применением ключей шардинга. Любое поле документа или комбинация полей могут быть использованы для создания ключа шардинга. Ключи

шардинга содержат дополнительное свойство, указывающее направление упорядочивания. Направление упорядочивания может быть равно 1, что значит упорядочивание по возрастанию, или -1 – упорядочивание по убыванию. Выбор ключа шардинга – важный этап конфигурирования, ключ должен позволять эффективно разделять наборы больших данных и балансировать нагрузку.

Все данные о шардах и чанках содержатся в метаданных на сервере конфигурации. Как и сами шарды, конфигурационные сервера также поддерживают механизм репликации для повышения отказоустойчивости.

Процессы клиентов поучают доступ к кластеру MongoDB через процесс mongos. Процесс mongos не имеет постоянного состояния – конфигурация определяется серверами конфигурации. В кластере может существовать несколько процессов mongos. Эти процессы отвечают за маршрутизацию пользовательских запросов, а также за объединение результатов запросов, ели это требуется. Запрос к кластеру MongoDB может быть целевым или глобальным. Все запросы, которые используют ключ шардинга – целевые, в противном случае – глобальные. Целевые запросы более эффективны. Глобальные запросы можно рассматривать как команды, приводящие к полному перебору коллекции.

Рисунок 3.14 демонстрирует архитектуру шардинга для MongoDB.

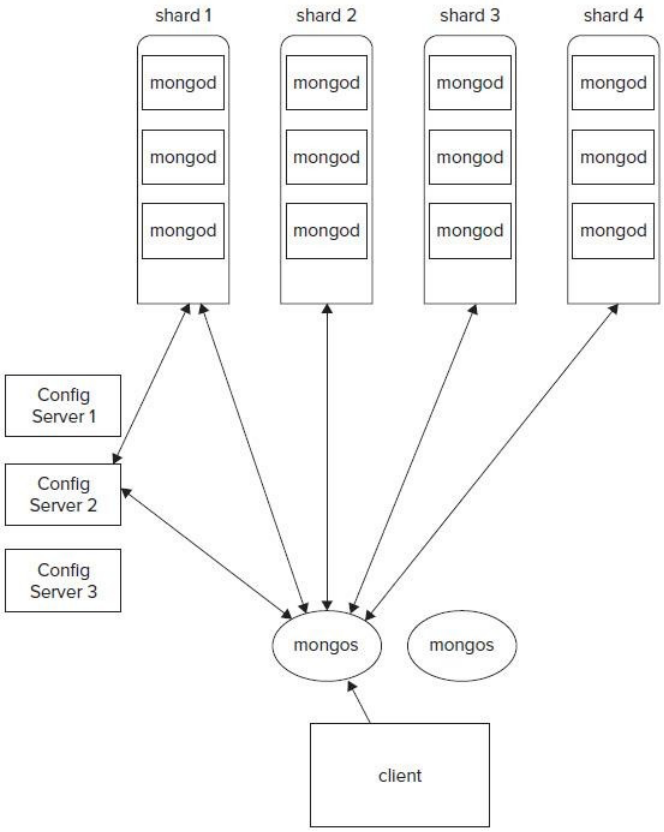


Рисунок 3.14 – Топология связей для реализации шардинга MongoDB

3.3.5 Хранилище типа ключ/значение в Memcached и Redis

Несмотря на то, что все хранилища типа ключ/значение отличаются, у них существуют общие базовые принципы построения. Например, они хранят данные в виде карт. Рассмотрим внутренние механизмы Memcached и Redis, чтобы продемонстрировать структуру надежных хранилищ типа ключ/значение.

Memcached – это распределенная высокопроизводительная система кэширования объектов. Она чрезвычайно популярна и используется в высокопроизводительных системах с высоким трафиком. Memcached используется на таких ресурсах как Facebook, Twitter, Википедия и YouTube. Memcached является простой системой с небольшим набором функций. Например, нет поддержки резервного копирования, отказоустойчивости, или восстановления. Memcached имеет простой API и может использоваться совместно с любым языком веб-программирования. Основная цель использования Memcached в стеке приложений – уменьшить нагрузку на базу

данных. На рис. 3.15 представлена возможная конфигурация для Memcached для веб-приложения.

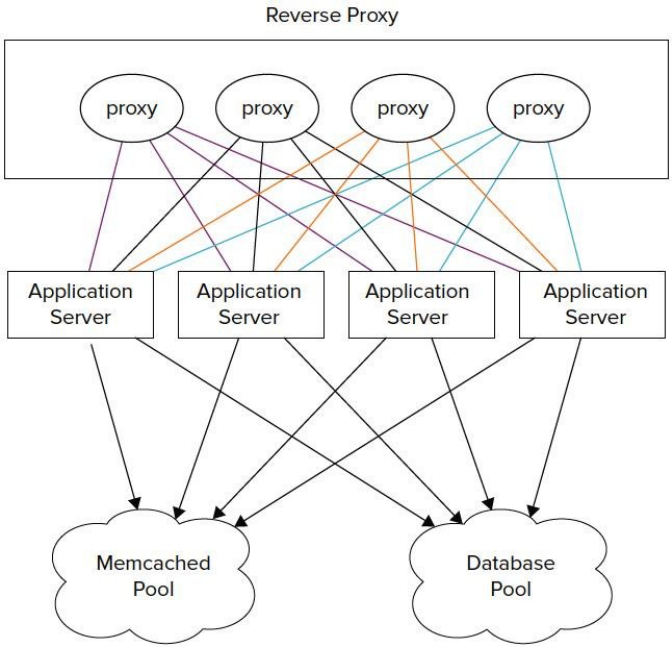


Рисунок 3.15 – Конфигурация Memcached

Центральным ядром Memcached является распределитель slab. Memcached сохраняет данные в slab-памяти. Slab состоит из страниц, которые распределяются по чанкам или корзинам. Наименьший размер slab равен 1 КБ, размер может расти кратно множителю 1,25. Memcached может хранить данные размером доя 1 МБ. Доступ к данным осуществляется по ключу. Размер ключа – до 250 байт. Каждый кэшированный объект хранится в чанке ограниченного размера. Это означает, что объект размером 1,4 КБ будет храниться в чанке размером 1,5625 КБ ($1,25^2$). Безусловно, такое поведение приводит к затратам памяти, особеннос в случаях, когда объекты ненамного превосходят размер чанка. По умолчанию, Memcached использует всю доступную память, размер ограничивается только архитектурой базового хранилища. На рис. 3.16 схематично представлен принцип размещения данных в Memcached.

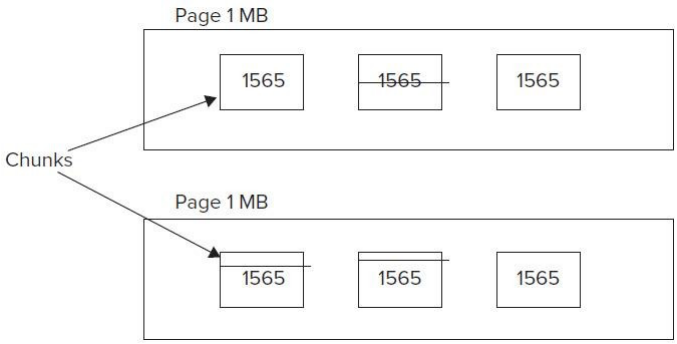


Рисунок 3.16 – Конфигурация Memcached

Алгоритмы LRU отвечает за удаление устаревших данных из кэша. Алгоритмы LRU основывается на работе фрагментированной slab-памяти. Фрагментация появляется при хранении и очистке памяти от объектов. Перераспределение памяти решает часть этой проблемы.

Memcached – это система кэширования объектов, которая не выполняет организацию элементов данных в коллекциях, таких как списки, наборы, отсортированные наборы, или карты. С другой стороны, Redis обеспечивает поддержку всех этих структур данных. Redis – это система подобная Memcached, но более надежная.

Все, что есть в Redis представляется в виде строки. Даже коллекции типа списков, множеств, сортированных множеств и карт, представлены в виде строк. В Redis определена специальная структура данных, называемая SDS (simple dynamic string). Эта структура состоит из трех частей:

1) buff – символьный массив для хранения строк;

2) len – численное значение, указывающее на длину массива buff;

3) free – количество дополнительных байт, доступных для использования.

Данные Redis располагаются в первичной памяти, перемещение их на диск происходит в определенные моменты. В отличие от MongoDB, эта система не использует memory-mapped файлы. Вместо этого, в Redis реализован собственная подсистема виртуальной памяти.

Дополнительно, в Redis реализован менеджер виртуальной памяти, который координирует неблокирующие сокетные операции. Архитектура Redis показана на рис. 3.17.

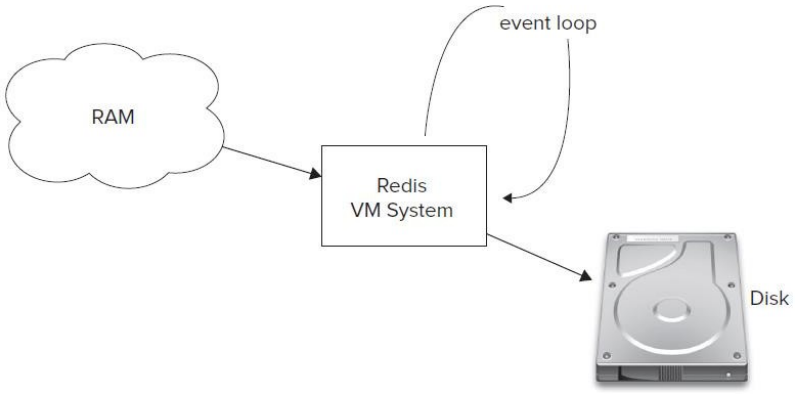


Рисунок 3.17 – Схема архитектуры Redis

Система Redis не связана с swap-системой операционной системмы, потому что:

1. Объекты Redis не реализуют маппинг один-к-одному с swap-страницами. Swap-страницы имеют размер 4,096 байт, а объекты Redis могут занимать более одной страницы или в одну swap-страницу могут быть помещены более одного объекта. В системе Redis, даже в случае низкого процента востребованных объектов в памяти, остаются доступными большое количество swap-страниц. Система swap операционной системы следит за использованием объектов (востребованностью данных). То есть, данные удаляются из swapping-системы даже если востребован один байт в странице.

2. В отличие от MongoDB, система Redis работает с различными форматами данных в памяти и на диске. Данные на диске хранятся в более компактном сжатом формате.

Контрольные вопросы

1. Опишите ключевые принципы построения реляционных баз данных.

2. Каковы главные отличия столбцовых баз данных и РСУБД?

3. Поясните содержание терминов: «встроенные карты», «пара ключ/значение».

4. В каких приложениях традиционно применяется Webtable?

5. Опишите обычную конфигурацию HBase.

6. Что такое сервер диапазонов в HBase?

7. Что такое регион в HBase?

8. Опишите структуру уникального идентификатора в MongoDB.

9. Что такое Memory-Mapped файл?

10. Опишите ключевые принципы согласованного хэширования.

3. MONGODB

Базы данных принимают самые разные формы. Предметные указатели в книгах, каталожные карточки, когда-то стоявшие в библиотеках, - это тоже базы данных, как и специально структурированные текстовые файлы, излюбленные программистами былых времен, работавшими на языке Perl. Но, наверное, чаще всего базы данных ассоциируются с изощренными реляционными СУБД, лежащими в основе огромного количества современных программных систем, теми самыми СУБД, на которых делаются состояния. Реляционные базы, с их идеализированной третьей нормальной формой и выразительным SQL-интерфейсом, все еще вызывают уважение старой гвардии - и вполне заслуженно.

Но несколько лет назад, разрабатывая веб-приложения, я мечтал опробовать только появляющиеся на свет альтернативы правящим в мире реляционным базам. Открыв для себя MongoDB, я сразу понял - это как раз то, что мне надо. Мне понравилась идея использовать JSON-подобные структуры для представления данных. Язык JSON прост, интуитивно понятен и удобен для человека. И тот факт, что язык запросов в MongoDB также основан на JSON вселяет в пользователя этой новой СУБД ощущение комфорта и гармонии. На первом месте стоит интерфейс. А наличие таких привлекательных средств, как простая репликация и сегментирование, только подстегивает интерес к пакету.

3.1 Основы MongoDB

MongoDB - это система управления базами данных, «заточенная» под веб-приложения и инфраструктуру Интернета. Модель данных и стратегия их постоянного хранения спроектированы для достижения высокой пропускной способности чтения и записи и обеспечивает простую масштабируемость с автоматическим переходом на резервный ресурс в случае отказа. Сколько бы

узлов ни требовалось приложению - один или десятки, - MongoDB сумеет обеспечить поразительно высокую производительность. Того, кто раньше мучился с масштабированием реляционных баз, эта новость обрадует. Но не всем приложениям необходим крупный масштаб. Быть может, вам всегда хватало одного сервера базы данных. Зачем тогда использовать MongoDB?

Как выясняется, привлекательность MongoDB объясняется в первую очередь не стратегией масштабирования, а интуитивно понятной моделью данных. Учитывая, что с помощью документо-ориентированной модели можно представить развитые иерархически организованные структуры данных, часто оказывается' возможно обойтись без присущих реляционным СУБД сложностей, связанных с соединением нескольких таблиц. Пусть, например, вы моделируете товары для сайта интернет-магазина. В полностью нормализованной базе данных информация об одном товаре может быть разбросана по десяткам таблиц. Чтобы получить его представление в интерактивной оболочке СУБД, придется написать сложный SQL-запрос с кучей соединений.

Поэтому разработчики, как правило, обращаются к дополнительным программам, когда хотят собрать разрозненные данные в нечто осмысленное. С другой стороны, в документной модели большая часть информации о товаре может быть представлена в виде одного документа. В оболочке MongoDB, построенной на основе языка JavaScript, нетрудно получить полное представление о товаре в виде иерархически организованной JSON-подобной структуры1. К ней можно предъявлять запросы, ей можно манипулировать. Средства составления запросов в MongoDB специально спроектированы для работы со структурированными документами, но так, чтобы пользователь, имеющий опыт работы с реляционными базами, располагал сравнимой выразительной мощью. К тому же, сегодня большинство разработчиков пишут на объектно-ориентированных языках, поэтому им нужно такое хранилище, которое было бы проще отобразить на объекты. В случае MongoDB объект, определенный на языке программирования, сохраняется «как есть» - без

дополнительной сложности, привносимой системами объектно-реляционного отображения.

7.1.1 Особенности MongoDB

База данных в значительной мере определяется своей моделью данных. В этом разделе мы рассмотрим документную модель данных, а затем покажем, какие особенности MongoDB позволяют эффективно работать с этой моделью. Мы также обсудим вопросы промышленной эксплуатации MongoDB, уделив особое внимание средствам репликации и стратегии горизонтального масштабирования.

Документная модель данных.

Модель данных MongoDB является документо-ориентированной. Для тех, кто не знаком с идеей документа в контексте баз данных, продемонстрировать ее проще всего на примере (рис. 7.1).

Рисунок 7.1 – Представление документа

В листинге 7.1 приведен пример документа, представляющего статью на социальном новостном сайте (вспомните о Digg). Как видите, документ - это по существу набор, состоящий из имен и значений свойств. Значение может быть представлено простым типом, например: строки, числа и даты. Но может

быть также массивом и даже другим документом (2) С помощью таких конструкций можно представлять весьма сложные структуры данных. Так, в нашем примере имеется свойство tags (1) - массив, в котором хранятся ассоциированные со статьей теги. Но еще интереснее свойство comments (3), которое ссылается на массив документов, содержащих комментарии.

Сделаем небольшую паузу и сравним это с представлением тех же данных в стандартной реляционной базе. На рис. 7.2 изображен типичный реляционный аналог. Поскольку таблицы по сути своей плоские, для представления связей типа один-ко-многим необходимо несколько таблиц. Мы начинаем с таблицы posts, в которой хранится основная информация о каждой статье. Затем создаем еще три таблицы, каждая из которых содержит поле post_id, ссылающееся на исходную статью. Эта техника разнесения данных объекта по нескольким таблицам называется нормализацией. Среди прочего, нормализованный набор данных характеризуется тем, что каждый элемент данных хранится только в одном месте.

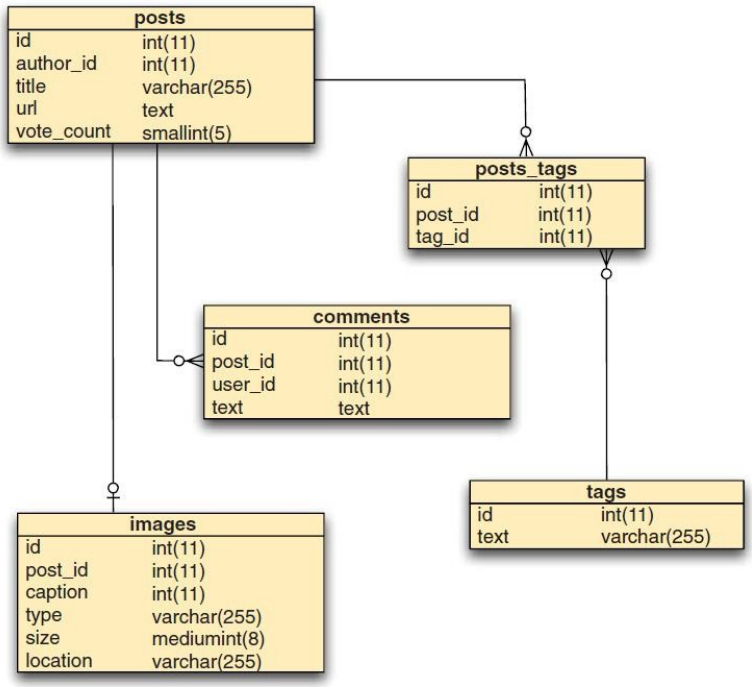


Рисунок 7.2 – Реляционная модель

Но за строгую нормализацию приходится платить. И самое главное то, что впоследствии данные нужно как-то собрать. Чтобы показать статью, необходимо выполнить соединение таблиц post и tags. А комментарии либо

запрашивать по отдельности, либо тоже включить их в соединение. В конечном итоге ответ на вопрос, нужна ли строгая нормализация, зависит от вида моделируемых данных. Я еще вернусь к этой теме в главе 4. Здесь же важно отметить, что документо-ориентированная модель позволяет естественно представить данные в агрегированной форме, то есть работать в объектом, как с единым целым: все данные, относящиеся к статье, от комментариев до тегов, хранятся в одном объекте базы данных.

Вероятно, вы обратили внимание, что документы не только позволяют представлять данные со сложной структурой, но и не нуждаются в заранее определенной схеме. В реляционной базе данных строки хранятся в таблице. У каждой таблицы имеется строго определенная схема, описывающая, какие столбцы и типы данных допустимы. Если окажется, что необходимо добавить еще одно поле, то таблицу надо будет явно изменить. В MongoDB документы группируются в коллекции - контейнеры, не налагающие на данные какую-либо схему Теоретически у каждого входящего в коллекцию документа может быть своя структура, но на практике документы в одной коллекции похожи друг на друга. Например, у всех документов в коллекции posts имеются поля title, tags, comments и т. д.

Отсутствие предопределенной схемы несет с собой некоторые преимущества. Во-первых, структуру данных определяет код приложения, а не база. Это позволяет ускорить разработку на ранних этапах, когда схема часто изменяется. Во-вторых, и это куда важнее, безсхемная модель позволяет представить данные с переменным набором свойств. Представьте, к примеру, каталог товаров к интернет-магазине. Заранее неизвестно, какие у товара будут атрибуты, поэтому приложение должно как-то адаптироваться к подобной изменчивости. Традиционно в базе данных с фиксированной схемой эта задача решается с помощью паттерна сущность-атрибут-значение3, как изображено на рис. 7.3. Здесь показан один раздел модели данных, применяемой в Magento, каркасе с открытым кодом для электронной торговли.

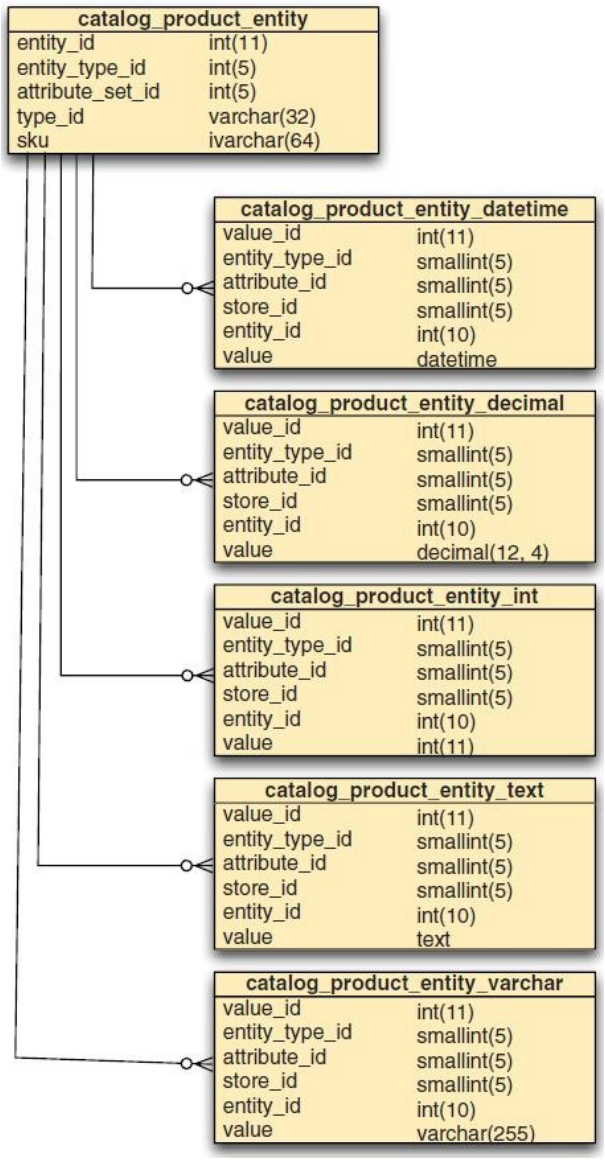


Рисунок 7.3 – Часть схемы проекта Magento, написанного на PHP

Обратите внимание на ряд таблиц, отличающихся только в одном поле value, которое в каждой таблице имеет свой тип. Такая структура позволяет администратору определять новые типы товаров вместе с их атрибутами, но ценой возрастания сложности. Представьте себе, какой запрос нужно было бы написать в оболочке MySQL, если бы потребовалось выбрать или обновить товар, смоделированный подобным образом; операции соединения таблиц оказались бы необычайно сложными. А при моделировании в виде документа никакого соединения не требуется, и добавлять новые атрибуты можно динамически.

Ad hoc запросы. Говорят, что система поддерживает произвольные запросы, если не

требуется заранее определять, какие запросы разрешены. Реляционная база данных таким свойством обладает; она честно выполнит любой правильно записанный SQL-запрос, сколько бы условий он ни содержал. Если вы работали только с реляционными СУБД, то, возможно, считаете произвольные запросы чем-то само собой разумеющимся. Однако не все базы данных поддерживают динамические запросы. Например, хранилища ключей и значений можно опрашивать только по одной оси: ключу. Как и многие другие системы, хранилища ключей и значений жертвуют гибкостью запросов в обмен на простоту модели масштабируемости. При проектировании MongoDB в частности ставилась задача по возможности сохранить выразительную мощь запросов, которая считается принципиально важной особенностью реляционных СУБД.

Принцип построения запросов в MongoDB мы рассмотрим на простом примере статей и комментариев. Пусть необходимо найти все статьи, помеченные тегом politics, за которые проголосовало более 10 посетителей. SQL-запрос для решения этой задачи выглядел бы так:

```
SELECT * FROM posts
INNER JOIN posts_tags ON posts.id = posts_tags.post_id
INNER JOIN tags ON posts_tags.tag_id == tags.id
WHERE tags.text = 'politics' AND posts.vote_count > 10;
```

Эквивалентный запрос в MongoDB формулируется путем задания документа-образца. Условие «больше» обозначается специальным ключом $gt.

```
db.posts.find({'tags': 'politics', 'vote_count': {'$gt': 10}});
```

Отметим, что в этих запросах предполагаются разные модели данных. SQL-запрос опирается на строго нормализованную модель, в которой статьи и теги хранятся в разных таблицах, тогда как в запросе для MongoDB считается, что теги хранятся внутри документа, описывающего статью. Однако же в

обоих случаях имеется возможность формулировать запрос с произвольной комбинацией атрибутов, что и составляет смысл понятия произвольных запросов.

Выше уже отмечалось, что не все базы данных поддерживают произвольные запросы - в угоду простоте модели. Так, хранилище ключей и значений можно опрашивать только по первичному ключу. С точки зрения запроса, значение, на которое указывает ключ, непрозрачно. Единственный способ включить в запрос дополнительный атрибут, в данном случае количество голосов, - написать специальный код, который вручную построит записи, где первичным ключом будет счетчик голосов, а значением - список первичных ключей документов, за которые подано соответствующее количество голосов. Если вы примените такой подход к хранилищу ключей и значений, то вас обвинят в трюкачестве. Хотя для небольших наборов данных это, возможно, и будет работать, но, вообще говоря, упаковывание нескольких индексов в физически единственный индекс - мысль не слишком удачная. К тому же, хешированные индексы, применяемые в хранилищах ключей и значений, не поддерживают запросы по диапазону, которые, скорее всего, необходимы для поиска с участием количества голосов.

Если вы раньше работали с реляционными базами данных, в которых произвольные запросы считаются нормой, то возьмите на за метку, что MongoDB обеспечивает аналогичную гибкость. Оценивая различные технологии баз данных, имейте в виду, что не все они поддерживают произвольные запросы, и если такая возможность необходима, то MongoDB может оказаться подходящим вариантом. Но одних лишь произвольных запросов недостаточно. Когда объем данных достигает определенного уровня, для обеспечения приемлемой эффективности становятся необходимы индексы. Наличие подходящих индексов увеличивает скорость выполнения запросов и сортировки на несколько порядков; следовательно, любая система, поддерживающая произвольные запросы, должна также поддерживать вторичные индексы.

7.1.2 Сервер и инструментальные средства MongoDB

СУБД MongoDB написана на языке C++ и активно разрабатывается компанией 10gen. Проект компилируется во всех основных операци-онных системах, включая Mac OS X, Windows и почти все дистрибутивы Linux. На сайте mongodb.org выложены откомпилированные двоичные файлы для каждой платформы. MongoDB - проект с открытым исходным кодом, распространяемый на условиях лицензии GNU-AGPL. Исходный код находится в открытом бесплатном доступе на сайте GitHub, в него часто включаются дополнения, предлагаемые сообществом. Но руководство проектом в целом осуществляет группа разработки сервера, состоящая из сотрудников компании lOgen, и подавляющее большинство обновлений в системе управления версиями исходит от нее.

The core server.

Исполняемый файл сервера базы данных называется mongod (mongodb. exe в Windows). Процесс mongod взаимодействует с клиентами по специальному протоколу, получая команды через сетевой сокет. Все используемые им файлы по умолчанию хранятся в каталоге /data/db6.

Сервер mongod может работать в нескольких режимах, самый употребительный - член набора реплик. Поскольку репликация рекомендуется, то обычно конфигурируется набор, состоящий из двух реплик и процесса-арбитра, расположенного на третьем сервере.

Если MongoDB работает в режиме автосегментирования, то процессы mongod в каждом сегменте конфигурируются как наборы реплик, а «сбоку» находятся специальные серверы метаданных, называемые конфигурационными серверами. Для отправки запросов конкретным сегментам используется также отдельный маршрутный сервер mongos.

Настройка процесса mongod проста по сравнению с другими СУБД, например MySQL. Конечно, можно определить номера портов и пути к каталогам данных, но для настройки собственно базы данных параметров не

так много. Оптимизация базы данных, которая в большинстве реляционных СУБД подразумевает манипуляции с кучей параметров, управляющих распределением памяти и другими аспектами работы сервера, превратилась в нечто сродни черной магии. Но при проектировании MongoDB было решено, что с управлением памятью операционная система справится лучше, чем администратор базы данных или разработчик приложения. Поэтому файлы данных проецируются на виртуальную память системы с помощью системного вызова mmap(). Тем самым забота об управлении памятью передается ядру ОС. Ниже я еще вернусь к вызову mmap (), а пока отмечу лишь, что почти полное отсутствие конфигурационных параметров - продуманная стратегия, а не ошибка.

The JavaScript shell.

Командная оболочка MongoDB - это инструмент для администрирования базы и манипулирования данными с помощью команд на языке JavaScript. Исполняемый файл mongo загружает оболочку и устанавливает соединение с указанным процессом mongod. Оболочка по своим возможностям сравнима с оболочкой MySQL с тем отличием, что язык SQL не используется. Вместо этого команды по большей части записываются в виде выражений языка JavaScript. Вот, например, как выбрать базу данных и вставить простой документ в коллекцию users:

```
> use mongodb-in-action
> db.users.insert({name: "Kyle"})
```

Первая команда, определяющая, какая база данных будет использоваться, знакома любому пользователю MySQL. Вторая команда - выражение JavaScript для вставки простого документа. Чтобы увидеть результат вставки, можно выполнить простой запрос.

Database drivers.

Если сама идея языкового драйвера вызывает у вас кошмарные ассоциации с низкоуровневыми драйверами устройств, расслабьтесь. Пользоваться драйверами MongoDB просто. Разработчики постарались

предоставить API, согласующийся с идиомами конкретного языка, обеспечив в то же время относительное единообразие интерфейса. Так, все драйверы реализуют общий набор методов для вставки документа в коллекцию, однако представление самого документа обычно выглядит естественно для данного языка. Например, в Ruby это будет хеш, в Python - словарь, а в Java, где нет аналогичного примитива на уровне языка, документ представляется с помощью специального класса конструктора документов, который реализует интерфейс LinkedHashMap.

Because the drivers provide a rich, language-centric interface to the database, little abstraction beyond the driver itself is required to build an application. This contrasts notably with the application design for an RDBMS, where a library is almost certainly necessary to mediate between the relational data model of the database and the object-oriented model of most modern programming languages. Still, even if the heft of an object-relational mapper isn’t required, many developers like using a thin wrapper over the drivers to handle associations, validations, and type checking.

На момент написания этой книги компания lOgen официально поддерживала драйверы для языков C, C++, C#, Erlang, Haskell, Java, Perl, PHP, Python, Scala и Ruby - и этот перечень постоянно расширяется. Если вам нужна поддержка для какого-то другого языка, то очень может быть, что уже имеется драйвер, написанный сообществом. А если такового не окажется, то на сайте mongodb.org опубликованы спецификации для разработки драйверов. Поскольку все официально поддерживаемые драйверы активно применяются в производственных системах и распространяются на условиях лицензии Apache, то авторы новых драйверов не испытывают недостатка в хороших примерах. Начиная с главы 3, я буду описывать, как работают драйверы и как ими пользоваться при написании программ.

Командные утилиты:

1. mongodump и mongorestore – стандартные утилиты резервного копирования и восстановления базы данных, mongodump сохраняет данные во

внутреннем формате BSON и потому применяется главным образом для создания резервных копий. Достоинством этой программы является возможность использования для снятия горячих резервных копий, из которых впоследствии можно восстановить базу данных с помощью утилиты mongorestore.

2. mongoexport и mongoimport – эти утилиты экспортируют и импортируют файлы в форматах JSON, CSV и TSV, поэтому они полезны, когда требуется представить данные в одном из широко распространенных форматов, mongoimport также удобна для начальной загрузки больших наборов данных, хотя стоит отметить, что перед импортом часто бывает желательно подправить модель данных, так чтобы можно было в полной мере задействовать все преимущества MongoDB. В таких случаях проще импортировать данные с помощью специально написанной программы на одном из поддерживаемых языков.

3. mongosniff – анализатор протокола для просмотра команд, посылаемых серверу базы данных. По сути дела транслирует передаваемые команды в формате BSON в понятные человеку команды оболочки..

4. mongostat – аналог iostat; постоянно опрашивает MongoDB и операционную систему для выдачи полезной статистики, в том числе количества операций (вставки, выборки, обновления, удаления и т. п.) в секунду, объема выделенной виртуальной памяти и числа подключений к серверу.

7.1.3 MongoDB и прочие базы данных

Количество имеющихся на рынке СУБД растет лавинообразно, и сравнивать одну с другой становится все труднее. К счастью, большинство из них попадает в одну из нескольких категорий. В последующих разделах я опишу простые и более изощренные хранилища ключей и значений, реляционные и документные базы данных и сопоставлю их с MongoDB.

Таблица 7.1 – Семейства баз данных

	Пример	Модель данных	Масштабирование	Применение
Простые хранилища ключей и значений	Memcached	Ключ-значение, где значением может быть произвольный двоичный объект.	Разные. Memcached может масштабироваться на несколько узлов, представляя всю доступную память в виде единого монолитного хранилища данных.	Кэширование, веб-приложения.
Развитые хранилища ключей и значений	Cassandra, Project Voldemort, Riak	Разные. В Cassandra используется структура ключзначение, называемая столбцом. В Voldemort хранятся двоичные объекты.	Согласованное в конечном счете аспределение по узлам,обеспечивающее высокую доступность и простую отработку отказа.	Вертикальные приложения с высокой пропускной способностью (веб-каналы активности, очереди сообщений). Кэширование. Веб-приложения.
Реляционные СУБД	Oracle database, MySQL, PostgreSQL	Таблицы.	Вертикальное масштабирование. Ограниченная поддержка кластеризации и ручного сегментирования.	Системы, нуждающиеся в транзакциях (банковские и финансовые приложения) или в языке SQL Нормализованная модель данных.

Назначение следует из названия - индексирование значение по ключу. Типичное применение - кэширование. Пусть, например, требуется кэшировать HTML-страницы, генерируемые приложением. В этом случае ключом может быть URL-адрес, а значением - сама HTML-страница. Отметим, что с точки зрения хранилища значение представляет собой непрозрачный массив байтов. Нет ни предопределенной схемы, как в реляционных базах данных, ни какого-либо понятия о типе данных. Это налагает естественное ограничение на операции, выполняемые хранилищами ключей и значений: можно только поместить новое значение, а затем получить или удалить его по ключу. Столь простые системы обычно работают очень быстро и хорошо масштабируются.

Из всех простых хранилищ ключей и значений наиболее широкое распространение получило memcached. Memcached хранит все данные в оперативной памяти, то есть приносит долговечность в жертву быстродействию. Хранилище распределенное - узлы memcached, размещенные на нескольких серверах, могут работать как единое хранилище данных, устраняя тем самым сложности, связанные с отслеживанием состояния кэша на разных машинах.

По сравнению с MongoDB простые хранилища ключей и значений типа memcached часто демонстрируют более высокую скорость операций чтения и записи. Но, в отличие от MongoDB, они редко используются в качестве основного хранилища данных. Они больше подходят на роль придатков, например, слоя кэширования поверх традиционной СУБД или простого слоя хранения для таких недолговечных структур, как очереди задач.

Простую модель ключей и значений можно развить, сделав ее пригодной для обработки более сложных схем чтения-записи или для предоставления улучшенной модели данных. Примером такого развитого хранилища ключей и значений может служить система Dynamo, разработанная компанией Amazon и описанная в широко известном документе «Dynamo: Amazon's Highly Available Key-value Store». Задача Dynamo - предложить базу данных, которая могла бы продолжать работу в условиях сетевых сбоев, отключения питания в центре обработки данных и других подобных отказов.

Таким образом, требуется, чтобы система при любых обстоятельствах обеспечивала чтение и запись, а это значит, что данные должны автоматически реплицироваться на несколько узлов. Если какой-то узел выходит из строя, то обслуживание пользователя системы, быть может, клиента Amazon, положившего что-то в корзину, не будет прервано. Dynamo располагает средствами разрешения неизбежных конфликтов, возникающих, когда системе разрешено записывать одни и те же данные на несколько узлов. В то же время Dynamo легко масштабируется. Поскольку в системе нет главного узла - все узлы.

7.1.4 Области применения MongoDB

MongoDB подходит на роль основного хранилища данных для веб-приложений. Даже в простом веб-приложении имеется много моделей данных для представления пользователей, сеансов, специфических данных, закачек и разрешений, не говоря уже о собственно предметной области. Все это прекрасно ложится как на табличную структуру, предлагаемую реляционными СУБД, так и на модель коллекций и документов в MongoDB. А поскольку с помощью документа можно представить сложные структуры данных, то количество коллекций обычно оказывается меньше, чем количество таблиц, необходимых для моделирования тех же данных в полностью нормализованной реляционной модели. Кроме того, с помощью динамических запросов и вторичных индексов можно без труда реализовать большинство запросов, знакомых пользователям SQL. Наконец, по мере роста приложения MongoDB предлагает простой механизм масштабирования.

На практике MongoDB подтвердила способность справляться со всеми аспектами приложения - от основных задач предметной области до таких вспомогательных аспектов, как протоколирование и аналитика в реальном времени. Примером может служить новостной сайт The Business Insider (TBE), который применяет MongoDB в качестве основного хранилища данных с января 2008 года. TBE обслуживает более миллиона уникальных просмотров страниц в день. Интересно, что помимо обработки основного содержимого сайта (статьи, комментарии, пользователи и т. д.), MongoDB также обрабатывает и хранит аналитические данные в режиме реального времени. Эти данные используются TBE для динамической генерации карт распределения интереса, на которых отражается кликабельность (CTR) различных новостей. Объем данных на сайте пока не настолько велик, чтобы требовалось сегментирование, но наборы реплик тем не менее используются для гарантированного перехода на резервный ресурс в случае отказа.

Гибкая разработка

Как бы вы относиться к движению за гибкую разработку, трудно отрицать, что приложение желательно создавать быстро. Немало коллективов, в частности Shutterfly и The New York Times, выбрали MongoDB отчасти за то, что она позволяет разрабатывать приложения быстрее, чем при использовании реляционных СУБД. Одна из причин, лежащая на поверхности, заключается в отсутствии фиксированной схемы, что позволяет не тратить время на разработку схемы, ее распространение и внесение в нее изменений.

Кроме того, меньше времени уходит на втискивание реляционного представления данных в объектно-ориентированную модель, на борьбу с капризами системы ORM и на оптимизацию сгенерированного ей кода. Поэтому MongoDB часто используется в проектах с коротким циклом разработки, над которыми трудятся гибкие команды среднего размера.

Аналитика и протоколирование.

Я уже упоминал, что MongoDB хорошо приспособлена для аналитики и протоколирования, и число приложений, в которых эта СУБД используется для таких целей, быстро растет. Нередко солидные компании с прочной репутацией начинают вторжение в мир MongoDB со специальных приложений, относящихся в аналитике. В качестве примеров приведу GitHub, Disqus, Justin.tv и Gilt Groupe. Для аналитических приложений MongoDB привлекательна своим быстродействием и двумя важными особенностями: атомарным обновлением и ограниченными коллекциями (capped collections). Атомарное обновление позволяет клиенту эффективно увеличивать счетчики и помещать новые значения в массив. Ограниченные коллекции, часто применяемые для протоколирования, имеют фиксированный размер, то есть старые элементы автоматически удаляются в порядке FIFO. Хранение протоколируемых данных в базе, а не в файловой системе упрощает администрирование и существенно расширяет возможности запросов. Теперь вместо использования grep или специализированной утилиты поиска в журнале пользователь может написать запрос на знакомом языке запросов MongoDB.

Кэширование.

Благодаря модели данных, позволяющей представлять объекты целиком, и повышенной скорости выполнения запросов MongoDB можно использовать вместо более традиционной связки MySQL и memcached. Например, вышеупомянутый сайт TBE смог отказаться от memcached в пользу непосредственной генерации страниц из данных, хранящихся в MongoDB.

3.2 Оболочка MongoDB Shell

В этой

главе мы с помощью оболочки MongoDB изучим основные концепции этой СУБД на ряде примеров. Вы научитесь создавать, читать, обновлять и удалять (create, read, update, delete - CRUD) документы и по ходу дела познакомитесь с языком запросов MongoDB. Кроме того, мы немного поговорим об индексах базы данных и об их применении для оптимизации запросов. И закончим главу рассмотрением некоторых простых административных команд, после чего я покажу, как можно получать справку по ходу работы. Можете считать эту главу одновременно более подробным изложением идей, упомянутых во введении, и практическим занятием по наиболее употребительным способам использования оболочки MongoDB.

7.2.1 Sell Basics

Если вы выполнили инструкции, приведенные в приложении A, то сейчас на ваш компьютер установлен работоспособный экземпляр MongoDB. Убедитесь, что процесс mongod работает, а затем запустите оболочку MongoDB:

./mongo

Если все нормально, то на экране появится приглашение, как на рис. 7.4. В заголовке отображается версия MongoDB, с которой вы работаете, и сведения о текущей базе данных. Если вы хоть немного знакомы с языком

JavaScript, то можете сразу же приступить к вводу кода и исследованию возможностей оболочки. В противном случае читайте дальше, и вы узнаете, как вставить свой первый документ.

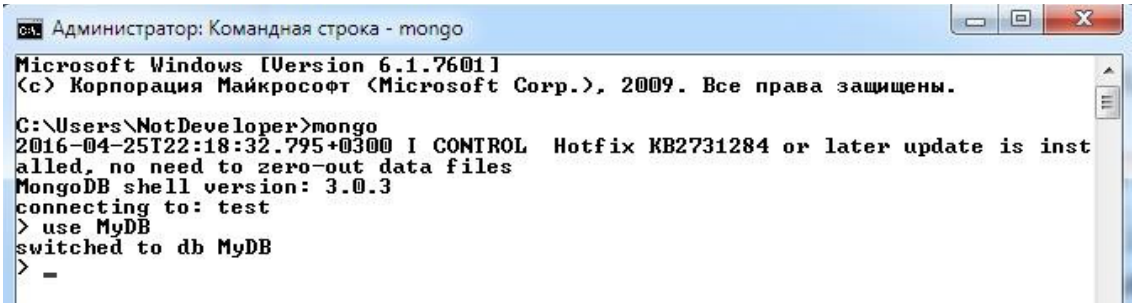


Рисунок 7.4 – Запуск оболочки MongoDB

Если при запуске не указана база данных, то по умолчанию оболочка выбирает базу test. Но чтобы во всех последующих упражнениях оставаться в одном и том же пространстве имен, переключимся на базу данных MyDB.

Возможно, у вас возник вопрос, как можно переключиться на базу данных tutorial, не создав ее явно. На самом деле, создавать базу данных необязательно. Базы данных и коллекции создаются при вставке первого документа. Такое поведение согласуется с принятым в MongoDB динамическим подходом к данным - точно также, как структуру документов не нужно определять заранее, так и отдельные коллекции и базы можно создавать прямо во время выполнения. Это упрощает и ускоряет процесс разработки и позволяет динамически назначать пространства имен, что очень часто бывает полезно. Тем не менее, если вас пугает возможность непреднамеренного создания базы данных или коллекции, то в большинстве драйверов имеется строгий режим, предотвращающий такого рода случайные ошибки.

Но пора уже создать первый документ. Так как мы работаем с JavaScript-оболочкой, то документы представляются в формате JSON (JavaScript Object Notation). Простейший документ, описывающий одного пользователя, мог бы выглядеть так:

```
{username: "Jones"}
```

Этот документ содержит одну пару ключ-значение для хранения имени пользователя Jones. Чтобы сохранить этот документ, нужно указать коллекцию. Коллекция users вполне подойдет:

> db.users.insert({username: "Jones"})

После ввода этого предложения может наблюдаться небольшая задержка. В это время на диске создается база данных tutorial и коллекция users. Задержка обусловлена созданием начальных файлов для того и другого.

Если вставка завершилась успешно, то ваш первый документ сохранен. Чтобы убедиться в этом, можно выполнить простой запрос:

> db.users.find()

Ответ будет выглядеть примерно так:

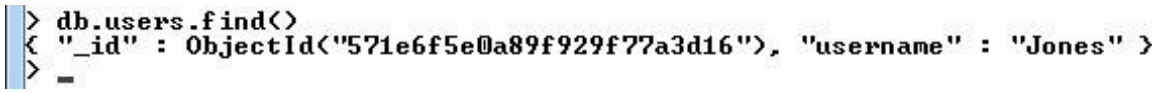


Рисунок 7.5 – Выполнение простого запроса

Обратите внимание, что в документ добавлено поле _id. Можете считать его значение первичным ключом документа. В любом документе MongoDB обязательно присутствует поле _id, и если в момент создания его значение не задано, то MongoDB генерирует специальный идентификатор объекта. На вашей консоли отобразится не тот же идентификатор, что в примере выше, но он гарантированно будет уникален среди всех значений _id в данной коллекции - единственное непререкаемое требование к этому полю.

Добавим нового пользователя:

> db.users.save({username: "Michel"})

Имея в коллекции более одного документа, мы можем попробовать более сложные запросы. Как и раньше, можно запросить все документы в коллекции:

> db.users.count()

2

Но можно также передать методу find простой селектор запроса. Селектором запроса называется документ, с которым сравниваются все

документы в коллекции. Чтобы найти все документы, в которых поле username равно jones, нужно задать такой селектор:

```
> db.users.find()
{ _id : ObjectId("4bf9bec50e32f82523389314"), username : "Jones" }
{ _id : ObjectId("4bf9bec90e32f82523389315"), username : "Michel" }
```

Селектор запроса {username: "jones"} возвращает все документы, в которых имя пользователя содержит строку jones - документобразец буквально сравнивается со всеми хранящимися в коллекции документами:

```
> db.users.find({username: "jones"})
{ _id : ObjectId("4bf9bec90e32f82523389315"), username : "Jones" }
```

Для обновления нужно задать по меньшей мере два аргумента. Первый определяет, какие документы обновлять, второй - как следует модифицировать отобранные документы. Существует два способа модификации; в этом разделе мы рассмотрим направленную модификацию (targeted modification) - одну из наиболее интересных и уникальных особенностей MongoDB.

Предположим, что пользователь smith решил указать свою страну проживания. Сделать это можно следующим образом:

```
> db.users.update({username: "Jones"}, {$set: {country: "Russia"}})
```

Здесь мы просим MongoDB найти документ, в котором поле username равно smith, и записать в свойство country значение Canada. Если теперь запросить этот документ, то мы увидим, что он обновился:

```
> db.users.find({username: "Jones"})
{"_id" : ObjectId("4bf9ec440e32f82523389316"),
"country" : "Russia", username : "Jones"}
```

Разовьем этот пример. Данные представляются в виде документов, которые, как мы видели в главе 1, могут содержать сложные структуры данных. Предположим, что, помимо профиля, пользователь желает хранить списки своих любимых вещей. Представление такого документа могло бы выглядеть так:

> db.users.update({username: "smith"}, {$unset: {country: 1}})

Ключ favorites указывает на объект, содержащий два других

ключа, указывающих на списки любимых городов и фильмов. Можете ли вы придумать, как с помощью того, что вам уже известно, модифицировать исходный документ о пользователе smith так, чтобы он принял такой вид? На ум должен сразу прийти оператор $set. Отметим, что в этом случае мы практически полностью переписываем документ, и $set ничего не имеет против: (Рисунок 7.6):

```
{
    username : "Jones",
    favorites : {
        cities : ["Moscow", "Stavropol"],
        movies : ["Crazy Cops", "The War", "12"]
    }
}
```

Рисунок 7.6 – Complex Document

Точка между favorites и movies означает, что нужно найти ключ favorites, который указывает на вложенный объект с ключом movies, а затем сравнить значение этого вложенного ключа с указанным в запросе. Этот запрос вернет оба документа.:

```
> db.users.update({username : "Jones"},{"$set": {favorites : {movies : ["12", "T
he War", "Crazy! Crazy!"]}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Рисунок 7.7 – Update complex document

Если не задавать никаких параметров, то операция удаления remove удалит из коллекции все документы. Так, чтобы избавиться от коллекции f oo, нужно выполнить такую команду::

> db.users.remove()

You often need to remove only a certain subset of a collection’s documents, and for that, you can pass a query selector to the remove() method. If you want to remove all users whose favorite city is Stavropol, the expression is pretty straightforward:

> db.users.remove({"favorites.cities": "Stavropol"})

Отметим, что операция remove () не уничтожает саму коллекцию, а лишь удаляет из нее документы. Можете считать ее аналогом команд SQL DELETE И TRUNCATE. Чтобы уничтожить коллекцию вместе со всеми построенными над ней индексами, используйте метод drop ():

> db.users.drop()

Создание, чтение, обновление и удаление - основные операции в любой базе данных. Если вы читали внимательно, то теперь можете самостоятельно поэкспериментировать с операциями CRUD в MongoDB. В следующем разделе мы продолжим изучение операций выборки, обновления и удаления, познакомившись с вторичными индексами.

7.2.2 Создание индексов и применение их в запросах

Индексы обычно создаются для повышения скорости выполнения запросов. К счастью, оболочка MongoDB позволяет создавать индексы безо всякого труда. Если ранее вам не доводилось работать с индексами базы данных, то из этого раздела вам станет ясно, зачем они нужны. А если у вас уже есть опыт работы с индексами, то вы узнаете, . как просто они создаются в MongoDB и как можно профилировать выполнение запросов с помощью метода explain().

Индексировать коллекцию имеет смысл, только когда в ней много документов. Поэтому добавим 200 000 простых документов в коллекцию numbers. Поскольку оболочка MongoDB одновременно является интерпретатором JavaScript, то сделать это несложно: (Рисунок 7.8).

```
> for (i=0; i<200000; i++){db.numbers.save({num:i})}
WriteResult({ "nInserted" : 1 })
> db.numbers.count()
200000
>
```

Рисунок 7.8 – Создание коллекции документов

200 000 документов - это немало, так что не удивляйтесь, если на выполнение команды уйдет несколько секунд. По завершении вы можете с помощью парочки запросов убедиться, что все документы на месте::

```
> db.numbers.find()
{ "_id" : ObjectId("571e773c0a89f929f77a3d17"), "num" : 0 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d18"), "num" : 1 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d19"), "num" : 2 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d1a"), "num" : 3 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d1b"), "num" : 4 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d1c"), "num" : 5 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d1d"), "num" : 6 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d1e"), "num" : 7 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d1f"), "num" : 8 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d20"), "num" : 9 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d21"), "num" : 10 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d22"), "num" : 11 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d23"), "num" : 12 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d24"), "num" : 13 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d25"), "num" : 14 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d26"), "num" : 15 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d27"), "num" : 16 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d28"), "num" : 17 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d29"), "num" : 18 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d2a"), "num" : 19 }
Type "it" for more
>
```

Рисунок 7.9 – Выполнение выборки

Команда count () показывает, что вставлено 200 000 документов. Второй запрос выводит первые 20 результатов. Чтобы показать следующую порцию, выполните команду it:

```
Type "it" for more
> it
{ "_id" : ObjectId("571e773c0a89f929f77a3d2b"), "num" : 20 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d2c"), "num" : 21 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d2d"), "num" : 22 }
{ "_id" : ObjectId("571e773c0a89f929f77a3d2e"), "num" : 23 }
```

Рисунок 7.10 – Запрос подмножества

Команда it просит оболочку вернуть следующий результирующий набор. Имея набор документов приличного размера, попробуем выполнить несколько запросов. Вас уже не удивит, что для поиска документа по его атрибуту num достаточно такого простого запроса:

```
> db.numbers.find({num : 500})
{ "_id" : ObjectId("571e773c0a89f929f77a3f0b"), "num" : 500 }
```

Более интересны запросы по диапазону, для которых предназначены специальные операторы $gt и $lt. Мы уже встречались с ними в главе 1 (gt означает greater than [больше], a it - less than [меньше]). Вот как запросить документы, для которых значение num больше 199 995:

```
> db.numbers.find({num : {"$gt" : 199995}})
{ "_id" : ObjectId("571e77730a89f929f77d4a53"), "num" : 199996 }
{ "_id" : ObjectId("571e77730a89f929f77d4a54"), "num" : 199997 }
{ "_id" : ObjectId("571e77730a89f929f77d4a55"), "num" : 199998 }
{ "_id" : ObjectId("571e77730a89f929f77d4a56"), "num" : 199999 }
>
```

Рисунок 7.11 – Использование оператора $gt

Как видите, с помощью простого JSON-документа можно сформулировать сложный запрос по диапазону, как на языке SQL. $gt и $lt - всего два из многочисленных ключевых слов, используемых в языке запросов MongoDB; в последующих главах мы встретим много других примеров.

Разумеется, от таких запросов мало толку, если они выполняются неэффективно. В следующем разделе мы впервые задумаемся об эффективности и начнем изучать имеющиеся в MongoDB средства индексирования.

Если вы долго работали с реляционными базами данных, то, наверное, знакомы с командой SQL explain. Она описывает путь выполнения запроса и позволяет выявить медленные операции, показывая, какие индексы были использованы. В MongoDB тоже имеется вариант explain с аналогичной функциональностью. Чтобы понять, как эта команда работает, применим ее к одному из предыдущих запросов. Выполните такую команду::

> db.numbers.find({num: {"$gt": 199995 }}).explain()

Результат должен выглядеть примерно так, как показано в листинге ниже.

```
{
  "cursor" : "BasicCursor",
  "nscanned" : 200000,
  "nscannedObjects" : 200000,
  "n" : 4,
  "millis" : 171,
  "nYields" : 0,
  "nChunkSkips" : 0,
  "isMultiKey" : false,
  "indexOnly" : false,
  "indexBounds" : { }
}
```

Рисунок 7.12 – Результат команды explain()

Изучение распечатки, выданной explain () показывает, что для

возврата всего четырех результатов (n) серверу пришлось просканировать всю коллекцию из 200 000 (nscanned) документов. Тип курсора BasicCursor подтверждает, что для формирования результирующего набора индексы не использовались. Столь большая разница между количеством просмотренных и возвращенных документов свидетельствует о неэффективности выполнения запроса. В реальном приложении, когда и коллекция больше, и сами документы объемнее, время выполнения запроса окажется существенно больше 171 миллисекунды, как в данном примере.

Этой коллекции явно недостает индекса. Построить индекс по ключу num можно с помощью метода ensure index (). Введите такую команду:

> db.numbers.ensureIndex({num: 1})

Как и для любой другой операции MongoDB, например выборки или обновления, методу ensurelndex () передается документ, определяющий, по каким ключам следует индексировать. В данном случае документ {num: 1} говорит, что над коллекцией numbers нужно построить индекс по ключу num в порядке возрастания. Убедиться в том, что индекс действительно построен, позволит метод get Indexes ():

```
> db.numbers.getIndexes()
[
  {
    "name" : "_id_",
    "ns" : "tutorial.numbers",
    "key" : {
      "_id" : 1
    }
  },
  {
    "_id" : ObjectId("4bfc646b2f95a56b5581efd3"),
    "ns" : "tutorial.numbers",
    "key" : {
      "num" : 1
    },
    "name" : "num_1"
  }
]
```

Рисунок 7.13 – Результат команды getIndexes()

Теперь над этой коллекцией построено два индекса. Первый - по стандартному ключу _id - автоматически строится для любой коллекции;

второй - по ключу num - мы только что создали сами. Если сейчас выполнить тот же запрос с помощью метода explain (), то будет заметна ощутимая разница во времени выполнения, что отражено в листинге ниже.

```
> db.numbers.find({num: {"$gt": 199995 }}).explain()
{
  "cursor" : "BtreeCursor num_1",
  "indexBounds" : [
    [
      {
        "num" : 199995
      },
      {
        "num" : 1.7976931348623157e+308
      }
    ]
  ],
  "nscanned" : 5,
  "nscannedObjects" : 4,
  "n" : 4,
  "millis" : 0
}
```

Рисунок 7.14 – Результат команды explain()

Теперь, когда используется индекс по ключу num, запрос просматривает только пять документов. В результате общее время выполнения сократилось со 171 мс до менее чем 1 мс.

Контрольные вопросы

1. В каком случае оправдан выбор MongoDB?

2. Каковы основные компоненты MongoDB?

3. Опишите различия MongoDB и прочими типами СУБД.

4. Опишите стратегии применения MongoDB.

5. Что такое MongoDB Shell? Какие типы операций поддерживает MongoDB shell?

6. Какой оператор используется для создания документов?

7. Какой оператор используется для получения количество документов в коллекции?

8. Поясните механизм инициализации коллекции в MongoDB.

9. Опишите значение терминов: collection, document, index и document-oriented database.

ЗАКЛЮЧЕНИЕ

В пособии были рассмотрены множество теоретических аспектов Big Data. Подведем итоги. MongoDB - документо-ориентированная СУБД с открытым исходным кодом. Будучи спроектирована с учетом требований к данным и масштабируемости, предъявляемым современными веб-приложениями, MongoDB поддерживает динамические запросы и вторичные индексы, быстрое атомарное обновление и сложные способы агрегирования, репликацию с автоматической отработкой отказов и сегментирование, обеспечивающее масштабирование по горизонтали.

Мы познакомились с документной моделью данных на практике и продемонстрировали типичные операции MongoDB. Вы научились создавать индексы и с помощью команды explain () убедились, что индексы повышают производительность выполнения запросов. Вы также умеете получать информацию о базах данных и коллекциях, имеющихся в системе, и знаете всё об «умной» коллекции $cmd. А если потребуется помощь, то у вас найдется парочка трюков, которые помогут выбраться на правильную дорогу. Многому можно научиться, работая в оболочке MongoDB, но ничто не сможет заменить опыт создания реального приложения.

СПИСОК ЛИТЕРАТУРЫ

Основная литература

1. Лэм, Ч. Hadoop в действии. / Чак Лэм. – М.: ДМК Пресс, 2012. – 424 с. – ISBN 978-5-94074-785-7.

2. Бэнкер, К. MongoDB в действии, пер. А. Слинкин / Кайл Бэнкер. – М.: ДМК Пресс, 2014. – 394 с. – ISBN 978-5-94074-831-1, 978-1-93518-287-0, 978-5-97060-057-3.

3. Фаулер, М. NoSQL: новая методология разработки нереляционных баз данных. Пер. с англ. / М. Фаулер, Дж. Садаладж. – М.: Вильямс, 2013. – 192 с.

4. Venner, J. Pro Hadoop. 2nd edition. / Jason Venner. – Apress; 2nd ed. (September 9, 2014). – 444 p.

5. Holmes, A. Hadoop in Practice / Alex Holmes. – Manning Publications; 2nd ed. (October 12, 2014). – 512 p.

Дополнительная литература

6. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. “The Google File System”; pub. 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October 2003. URL: http://labs.google.com/papers/gfs.html

7. Jeffrey Dean and Sanjay Ghemawat. “MapReduce: Simplifi ed Data Processing on Large Clusters”; pub. OSDI’04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December 2004. URL: http://labs.google.com/papers/mapreduce.html

8. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. “Bigtable: A Distributed Storage System for Structured Data”; pub. OSDI’06:

Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November 2006. URL: http://labs.google.com/papers/bigtable.html.

9. Mike Burrows. “The Chubby Lock Service for Loosely-Coupled Distributed Systems”; pub. OSDI’06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November 2006. URL: http://labs.google.com/papers/chubby.html.

10. http://adoop.apache.org.

11. White, T. Hadoop: The Definitive Guide. 4th edition. / Tom White. – O'Reilly Media; 4th ed. (April 11, 2015). – 756 p. ISBN-10: 1491901632.

12. Marz, N. Big Data: Principles and best practices of scalable realtime data systems. / Nathan Marz. – Manning Publications; 1st ed. (May 10, 2015). – 328 p.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ «Технология разработки NoSQL решений»

для обучающихся по направлению 09.03.02 «Информационных системы и технологии», профиль «Разработка web-ориентированных информационных систем» всех форм обучения

Составители:

Саргсян Эрик Ромович

Рындин Никита Александрович

Подписано в печать 04.06.2021

Формат 60x84 1/8 Бумага для множительных аппаратов

Уч.-изд. л. 3,3 Усл. печ. л. 3,0.

ФГБОУ ВО «Воронежский государственный технический университет»

396026 Воронеж, Московский просп., 14

Участок оперативной полиграфии издательства ВГТУ

396026 Воронеж, Московский просп., 14