Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени первого Президента России Б. Н. Ельцина»

УТВЕРЖДАЮ

Директор по образовательной деятельности

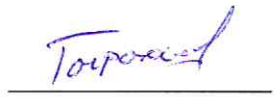
_____ С.Т. Князев
«7» сентября 2023 г.

Нереляционные базы данных

Учебно-методические материалы по направлению подготовки
09.03.03 Прикладная информатика
Образовательная программа «Прикладной искусственный интеллект»

Екатеринбург

РАЗРАБОТЧИКИ УЧЕБНО-МЕТОДИЧЕСКИХ МАТЕРИАЛОВ

Доцент Департамента информационных технологий и автоматики, канд.техн.наук		К.А. Аксенов
Ассистент Департамента информационных технологий и автоматики		А.А. Тарасьев

СОДЕРЖАНИЕ

МАТЕРИАЛЫ ЛЕКЦИЙ

Раздел 1. Введение в нереляционные базы данных

Содержание:

- Основные причины возникновения нереляционных баз данных
- Особенности и преимущества нереляционных баз данных
- Сравнение с реляционными базами данных
- Преимущества и недостатки каждого подхода
- Критерии выбора между нереляционными и реляционными базами данных
- Использование гибридных подходов

Первые базы данных были основаны на реляционной модели данных, которая включала в себя таблицы, столбцы и связи между ними. Однако, с течением времени выяснилось, что в некоторых случаях реляционная модель не является наиболее эффективной и гибкой.

Основные причины появления нереляционных баз данных включают:

- **Масштабируемость**: Реляционные базы данных могут столкнуться с проблемами масштабирования при обработке больших объемов данных. Нереляционные базы данных позволяют более эффективно масштабировать систему горизонтально или вертикально, чтобы обеспечить обработку больших объемов данных.
- **Гибкость модели данных**: Реляционные базы данных требуют определения схемы данных заранее, что ограничивает возможности изменения структуры данных. Нереляционные базы данных позволяют более гибкое добавление, изменение и удаление данных без необходимости изменения всей структуры данных.
- **Быстродействие**: Реляционные базы данных используют сложные операции присоединения (join) для объединения данных из разных таблиц. В нереляционных базах данных данные могут быть предварительно агрегированы или денормализованы, что увеличивает скорость доступа к данным.

Важность данных принципов обусловлена следствиями из известной теоремы CAP.

CAP (Consistency, Availability, Partition tolerance) – теорема, которая была предложена в 2000 году *Эриком Брюэром, Сетом Джедем и Эндрю Стосселом*. Она связана с проблемами, возникающими при разработке

распределенных систем, а именно, с достижением согласованности данных, обеспечением доступности и возможностью работы при разделении системы на составные части или узлы.

Согласованность (Consistency) обозначает, что все копии данных в распределенной системе должны быть в согласии друг с другом. Это означает, что если изменяется одна копия данных, другие копии должны обновиться для отражения этих изменений. То есть, все пользователи системы всегда видят одну и ту же версию данных.

Доступность (Availability) – это свойство системы, которое позволяет ей оставаться доступной и функционирующей даже в случае сбоев или отказов в отдельных компонентах или узлах. Доступность означает, что пользователи всегда могут получить доступ к системе и использовать ее функциональность.

Разделение на составные части (Partition tolerance) означает возможность работы системы, даже если она разделена на несколько частей или узлов, которые не могут обмениваться данными между собой или не имеют постоянного соединения.

Теорема CAP гласит, что в распределенной системе **невозможно** одновременно достичь всех трех свойств – согласованности, доступности и разделения на составные части. В лучшем случае, система может гарантировать только два из трех этих свойств.

Таким образом, при проектировании распределенной системы разработчики должны принимать решение о том, какие два свойства они хотят обеспечить в приоритете. В зависимости от конкретных требований системы и ее целей, выбор может быть различным.

Существует три основных модели систем, которые могут быть созданы на основе теоремы CAP:

CP (Consistency and Partition tolerance) – эта модель обеспечивает согласованность данных и разделение на составные части, но может быть недоступна в случае разделения системы. Такие системы обычно используются в банковских приложениях и других областях, где необходимо строгое соблюдение целостности данных.

AP (Availability and Partition tolerance) – эта модель обеспечивает доступность данных и разделение на составные части, но может потерять согласованность данных. Такие системы нацелены на обеспечение отказоустойчивости и масштабируемости, и могут использоваться, например, в социальных сетях или интернет-магазинах.

CA (Consistency and Availability) – это модель, которая гарантирует согласованность данных и их доступность, но не обеспечивает разделение на составные части. Такие системы обеспечивают высокую надежность и производительность, но могут быть недоступны в случае сетевых сбоев или разделения системы.

Конечно, каждый конкретный случай требует индивидуального подхода и выбора наиболее подходящей модели системы, с учетом всех требований и ограничений.

Таким образом, теорема CAP предоставляет руководства и предупреждения для разработчиков распределенных систем. Выбор между согласованностью, доступностью и разделением на составные части – сложное решение, но правильный выбор поможет достичь наилучшего баланса между требованиями системы и пользовательским опытом.

Исходя из данных выводов, нереляционные или постреляционные базы данных как правило проектируются в пользу организации систем, соответствующих требованиям **AP** для обеспечения доступности масштабируемости (разделению на составные части).

Как следствие, при построении подобных систем используется принцип **BASE** в противовес привычному для реляционных СУБД **ACID**.

Принцип BASE (Basically Available, Soft State, Eventually Consistent) является альтернативой строгой согласованности данных в распределенных системах. Он является основополагающей идеей в проектировании и разработке распределенных систем с высокой доступностью и горизонтальным масштабированием.

Основные принципы BASE

1. Basically Available (В основном доступно)

Данный принцип предполагает, что система всегда должна оставаться доступной для запросов даже при возникновении отказов или сбоев. Вместо блокирования доступа к данным, система предоставляет пользователю механизм доступа в любое время, даже если данные находятся во временном состоянии или не соблюдают строгие правила целостности.

2. Soft State (Мягкое состояние)

Данный принцип BASE предполагает, что состояние распределенной системы может меняться с течением времени без каких-либо гарантий согласованности данных или компонентов распределенной системы или обязательств. Каждый узел системы может иметь свое собственное представление о состоянии системы в данный момент, и это состояние может изменяться, пока система продолжает обрабатывать запросы.

3. Eventually Consistent (В конечном итоге согласованное)

Данный принцип говорит о том, что согласованность данных в распределенной системе достигается *в конечном итоге*, но не обязательно

мгновенно. Система не гарантирует моментальную согласованность данных между различными узлами, однако, со временем данные конвергируются и достигают согласованного состояния.

Отличия между ACID и BASE

Принцип BASE противопоставляется традиционному для реляционных баз данных принципу **ACID (Atomicity, Consistency, Isolation, Durability)**, который стремится к сильной целостности и согласованности данных с помощью транзакций. Рассмотрим основные отличия:

- ACID гарантирует строгую согласованность данных в системе, тогда как BASE признает возможность временной несогласованности.
- ACID требует полного выполнения всех операций транзакции перед их фиксацией, в то время как BASE позволяет системе продолжать обрабатывать запросы, даже если данные не находятся в *консистентном* состоянии.
- ACID склонен к блокированию операций при конфликтах, чтобы избежать несогласованности данных. В то же время BASE предпочитает минимизировать блокировки и предоставлять доступ к данным или фрагментам данных в любое время.
- ACID гарантирует долговременное хранение данных, тогда как BASE может временно хранить изменения данных до достижения согласованного состояния.

Примеры применения принципа BASE

Принцип BASE нашел широкое применение в различных распределенных системах и технологиях. К основным примерам относятся:

- ***NoSQL базы данных***, такие как Apache Cassandra и MongoDB, применяют принцип BASE для обеспечения высокой доступности и горизонтального масштабирования.
- Шардирование баз данных, где данные разделены на различные фрагменты (шарды) и каждый шард может иметь свое собственное состояние, применяет принцип BASE для обеспечения доступности и возможности обработки запросов независимо от состояния других шардов.
- Веб-приложения и микросервисные архитектуры, которые могут использовать принцип BASE для обеспечения отказоустойчивости, доступности и горизонтального масштабирования.

Основные особенности нереляционных баз данных:

- **Поддержка различных моделей данных**: Нереляционные базы данных поддерживают различные модели данных, такие как ключ-значение (key-value), столбцовые (columnar), документо-ориентированные (document),

графовые (graph) и др. Это позволяет выбрать наиболее подходящую модель для определенного типа данных.

- **Горизонтальное и вертикальное масштабирование**: Нереляционные базы данных обеспечивают более эффективное масштабирование системы, позволяя добавление дополнительных серверов с целью распределения и шардирования или увеличение доступных ресурсов на существующих серверах.
- **Высокая производительность**: Нереляционные базы данных предлагают высокую производительность и скорость доступа к данным, так как они обычно используют более простые операции чтения/записи данных без сложных операций соединения таблиц. В зависимости от типа NoSQL баз данных приоритет может отдаваться быстродействию чтения или, наоборот, записи данных.
- **Географическое распределение данных**: Нереляционные базы данных могут легко распределять данные по различным географическим областям, что позволяет улучшить доступность и отказоустойчивость системы.

Сравнение нереляционных и реляционных баз данных позволяет выявить основные различия и сходства между ними:

- **Схема данных**: Реляционные базы данных требуют определения схемы данных заранее, в то время как нереляционные базы данных могут быть более гибкими по отношению к структуре данных (именно здесь в первую очередь кроется противопоставление BASE и ACID).
- **Язык запросов**: Реляционные базы данных используют SQL для выполнения запросов, тогда как нереляционные базы данных имеют свои собственные языки запросов (например, MongoDB использует JavaScript-подобный язык, т.к. данные хранятся в формате документа JSON).
- **Сложность операций**: Реляционные базы данных обладают более сложными операциями соединения таблиц, тогда как нереляционные базы данных могут быть более простыми в использовании и более человекопонятными.

Преимущества нереляционных баз данных:

- Масштабируемость и производительность
- Гибкость структуры данных
- Высокая доступность и отказоустойчивость
- Поддержка больших объемов данных

Недостатки нереляционных баз данных:

- Отсутствие стандартного языка запросов (некоторые базы данных используют собственные языки)
- Ограниченная поддержка для сложных операций соединения данных

Преимущества реляционных баз данных:

- Четкая схема данных и стандартный язык запросов (SQL)
- Надежность и целостность данных
- Широкая поддержка со стороны разработчиков и большое сообщество пользователей

Недостатки реляционных баз данных:

- Ограничения на масштабируемость (особенно при больших объемах данных)
- Большие накладные расходы на операции соединения данных

При выборе между нереляционными и реляционными базами данных следует учитывать следующие критерии:

- **Структура данных**: Если структура данных предполагает жесткую схему и высокую структурированность, то реляционные базы данных могут быть предпочтительными. Если же структура данных неоднородная или изменчивая, то нереляционные базы данных могут быть более подходящим выбором.
- **Объем данных**: Если требуется обработка больших объемов данных или горизонтальное масштабирование, то нереляционные базы данных могут быть предпочтительными. Реляционные базы данных лучше подходят для относительно небольших объемов данных.
- **Производительность и скорость**: Если требуется быстрый доступ к данным и выполнение простых операций, то нереляционные базы данных могут быть более эффективными. Реляционные базы данных могут быть более подходящими для выполнения сложных операций соединения.
- **Отказоустойчивость и доступность**: Если система должна быть отказоустойчивой и обладать высокой доступностью, то нереляционные базы данных могут быть предпочтительными.

Гибридные подходы комбинируют возможности нереляционных и реляционных баз данных для достижения наилучших результатов:

- **Частичная денормализация данных**: Возможно использование нереляционной базы данных для частичной денормализации данных, что может увеличить производительность при выполнении операций чтения.
- **Репликация данных**: Реляционная база данных может быть использована для хранения основных данных, а нереляционная база данных может использоваться для репликации данных и обеспечения высокой доступности.

- **Использование индексов**: Реляционная база данных может использоваться для создания индексов и выполнения сложных операций соединения, в то время как нереляционная база данных может использоваться для хранения основных данных.

Гибридные подходы позволяют комбинировать преимущества обоих типов баз данных и достичь наилучших результатов в соответствующих случаях использования.

Гибридный подход может быть достигнут как совмещением в системе релационных и нереляционных СУБД, так и применением СУБД, позволяющих использовать оба подхода, как, например, в PostgresSQL (реляционная СУБД, позволяющая хранить JSON документы в виде отдельных полей реляционных сущностей). Следует отметить, что также некоторые системы могут применять гибридные постреляционные СУБД и нереляционные для достижения потребностей архитектуры.

Раздел 2. Категории нереляционных баз данных

Содержание:

- Иерархические базы данных
- Сетевые базы данных
- Документоориентированные базы данных
- Ключ-значение базы данных
- Столбцовые базы данных
- Графовые базы данных

В последнее время наблюдается значительный рост данных, и традиционные реляционные базы данных не всегда эффективно справляются с обработкой такого объема и разнообразия данных. В ответ на это возникли нереляционные (NoSQL) базы данных, которые обладают высокой масштабируемостью, производительностью и гибкостью. В этой лекции мы рассмотрим различные категории нереляционных баз данных, такие как иерархические, сетевые, документоориентированные, ключ-значение, столбцовые и графовые базы данных.

- **Иерархические базы данных**

Иерархические базы данных организуют данные в виде древовидной структуры, где каждый узел может иметь несколько потомков и только одного родителя. Они эффективно применяются для моделирования древовидной иерархии, например, файловой системы. Пример такой базы данных - IMS (Information Management System) от IBM.

- **Сетевые базы данных**

Сетевые базы данных представляют данные в виде графической структуры с использованием связей между узлами. В отличие от иерархических баз данных, здесь узел может иметь несколько родителей, что обеспечивает более гибкое представление данных. Сетевые базы данных были популярны в 1960-х и 1970-х годах, но сейчас их использование сильно снизилось. Пример такой базы данных - IDMS (Integrated Database Management System) от Computer Associates.

- **Документоориентированные базы данных**

Документоориентированные базы данных хранят, индексируют и извлекают данные в формате документов, обычно в JSON или XML. Некоторые современные базы данных также могут использовать собственные форматы документов. Документы могут быть иерархическими или плоскими и представляют собой независимые единицы информации. Документоориентированные базы данных позволяют более гибкое хранение и

запросы к сложным структурам данных. Примеры таких баз данных - **MongoDB**, CouchDB.

- **Ключ-значение базы данных**

Ключ-значение базы данных представляют данные в виде ассоциативного массива, где каждое значение связано с уникальным ключом. Эта категория баз данных проста, эффективна и быстро масштабируется. Она используется для хранения крупных объемов данных с высокой скоростью чтения и записи, но она не поддерживает сложные запросы и аналитические операции. Как правило, такие базы данных используются в сложных распределенных системах в качестве вспомогательных систем для реализации взаимодействия между компонентами системы или прочих вспомогательных нужд. В качестве основной системы хранения данных в информационных системах используются достаточно редко, и, как правило, для специфичных программных продуктов. Примеры таких баз данных - Redis, Amazon DynamoDB.

- **Столбцовые базы данных**

Столбцовые базы данных организуют данные в виде таблицы с несколькими столбцами, в отличие от реляционных баз данных, которые организуют данные в виде строк. Это позволяет эффективное хранение и обработку больших объемов структурированных данных. Столбцовые базы данных часто используются для аналитических задач и решения Big Data, построения информационных дашбордов, метрик или логов. Примеры таких баз данных - Apache Cassandra, Google Bigtable, Yandex ClickHouse.

- **Графовые базы данных**

Графовые базы данных моделируют данные в виде графов, где узлы представляют сущности, а ребра - связи между ними. Эта модель идеально подходит для моделирования социальных сетей, рекомендательных систем, географических данных и т.д. Графовые базы данных обеспечивают эффективный поиск и анализ данных, основанный на связях между сущностями. Основными областями применения таких баз данных являются социальные сети, рекомендательные системы, биоинформатика, логистика и маршрутизация и другие. Примеры таких баз данных - Neo4j, Amazon Neptune.

Нереляционные базы данных предоставляют широкий спектр возможностей для хранения и обработки данных, не требуя жесткого соблюдения схемы данных, применяемой в реляционных базах данных. Каждая категория нереляционных баз данных имеет свои преимущества и недостатки, и должна быть выбрана в зависимости от конкретных требований проекта и характеристик данных.

Раздел 3. Модели данных в нереляционных базах данных

Содержание:

- Модель ключ-значение
- Модель столбцов
- Модель документоориентированная
- Модель графовая
- Сравнение моделей данных

В этом разделе рассматриваются четыре основные модели данных, используемые в нереляционных базах данных: модель ключ-значение, модель столбцов, документоориентированная модель и графовая модель. Также приводится сравнение этих моделей, обсуждаются их преимущества и ограничения, а также рассматриваются примеры реализации каждой из этих моделей.

1. Модель ключ-значение:

Модель ключ-значение является одной из самых простых моделей данных в нереляционных базах данных. В этой модели данные представлены в виде пар ключ-значение, где ключ уникален и используется для идентификации записи, а значение представляет собой некоторый набор данных или объект. Такой принцип должен быть понятен любому программисту, т.к он идеально согласуется со стандартными принципами обработки массивов (списков/коллекций) и/или словарей (в зависимости от особенностей используемого языка программирования).

- **Принцип построения:**

При использовании модели ключ-значение каждая запись хранится в наборе пар ключ-значение. Каждая запись имеет уникальный ключ, по которому ее можно идентифицировать, и значение, которое может быть представлено в различных форматах, таких как строка, число, список и т.д. В некоторых системах, значения также могут быть представлены в виде сложных структур данных, таких как хеш-таблицы или JSON, что роднит данные системы с документоориентированными, но в менее гибком формате.

Примером использования модели ключ-значение является система кэширования, где ключ представляет собой URL страницы, а значение представляет собой HTML-код этой страницы. Когда пользователь делает запрос к странице, система сначала проверяет наличие этой страницы в кэше по ключу. Если страница присутствует в кэше, она возвращается пользователю без выполнения запроса к базе данных.

- Преимущества:

Простота: модель ключ-значение очень проста в использовании и позволяет быстро работать с данными

Гибкость: возможность хранить данные в различных форматах делает эту модель гибкой и адаптивной к различным типам данных

Высокая производительность: благодаря простоте и скорости доступа к данным, модель ключ-значение обеспечивает высокую производительность

- Ограничения:

Отсутствие сложных операций запроса: модель ключ-значение не поддерживает сложные операции запроса, такие как объединение таблиц или сложные условия фильтрации данных

Ограниченные возможности по поиску данных: поиск данных в модели ключ-значение осуществляется только по ключу, что существенно ограничивает возможности поиска и фильтрации данных

Пример реализации модели ключ-значение: Redis, Memcached.

2. Модель столбцов:

Модель столбцов является одной из наиболее распространенных моделей данных в нереляционных базах данных. В этой модели данные организуются и сохраняются в виде набора столбцов, где каждая запись представляет собой набор значений для этих столбцов.

- **Принцип построения:**

При использовании модели столбцов данные разбиваются на отдельные столбцы, каждый из которых содержит некоторое значение. Эти значения хранятся в ячейке, которая идентифицируется по строке и столбцу. Записи могут иметь разное количество столбцов и значения могут иметь различные типы данных.

Примером использования модели столбцов может служить система учета товаров на складе, где каждая запись представляет собой отдельный товар, а столбцы представляют информацию о товаре, такую как название, цена, количество и т.д. Каждый столбец содержит соответствующее значение для каждого товара.

Такой подход на первый взгляд может быть очень похож на традиционные реляционные базы данных.

Однако, столбцовые и реляционные базы данных имеют несколько различий:

В реляционных базах данных данные хранятся и организованы в виде таблиц с рядами и столбцами, где каждый столбец содержит отдельное поле данных. В столбцовых базах данных данные организованы вертикально, то есть каждая колонка содержит набор значений с одним и тем же типом данных. Столбцовые базы данных обеспечивают более гибкую структуру данных, чем реляционные базы данных. Они позволяют добавлять и удалять

столбцы независимо друг от друга, без изменения схемы или структуры базы данных. В реляционных базах данных добавление или удаление столбцов может потребовать изменения схемы базы данных и обновление всех записей в таблице.

С другой стороны, в столбцовых базах данных можно выполнять операции только над определенными столбцами, что позволяет значительно сократить время выполнения запросов. Например, если нужно произвести вычисления только над числовыми данными в определенном столбце, то столбцовые базы данных позволяют выполнить это быстрее, так как не требуется обращаться ко всей строке. Также в столбцовых базах данных можно эффективно сжимать и хранить данные, что позволяет значительно снизить издержки хранения и улучшить производительность. Это особенно полезно для хранения больших объемов данных и обработки аналитических запросов.

- Преимущества:

Гибкость: модель столбцов позволяет хранить и организовывать данные в более гибком формате по сравнению с реляционными базами данных

Высокая производительность: благодаря механизму хранения данных в виде столбцов, модель столбцов обеспечивает быстрый доступ к данным и высокую производительность

- Ограничения:

Комплексность запросов: модель столбцов может быть сложной для понимания и использования при выполнении сложных операций запроса, таких как объединение таблиц или сложные условия фильтрации данных

Ограниченные возможности по поиску данных: поиск данных в модели столбцов может быть сложным, особенно при запросах, требующих обработки больших объемов данных.

Пример реализации модели столбцов: Apache Cassandra, Google Bigtable.

3. Документоориентированная модель:

Документоориентированная модель является одной из наиболее гибких моделей данных в нереляционных базах данных. В этой модели данные хранятся в виде документов, которые содержат информацию о сущностях в формате, близком к формату JSON.

- **Принцип построения:**

При использовании документоориентированной модели данные организуются и сохраняются в виде коллекции документов. Каждый документ представляет собой набор полей, где каждое поле имеет уникальное имя и значение. Документы могут иметь разное количество полей, и значения полей могут быть различных типов данных.

Примером использования документоориентированной модели может служить система управления задачами, где каждая задача представлена в виде отдельного документа. Каждый документ содержит информацию о задаче, такую как название, описание, сроки выполнения и т.д. Каждое поле представляет соответствующую информацию задачи.

- Преимущества:

Гибкость: документоориентированная модель позволяет хранить и организовывать данные в гибком формате, что делает ее идеальной для хранения и обработки иерархических данных

Простота: документоориентированная модель достаточно проста для использования и позволяет быстро работать с данными

Поддержка сложных операций запроса: документоориентированная модель позволяет выполнять сложные операции запроса, такие как фильтрация данных по различным условиям или поиск по вложенным полям

- Ограничения:

Избыточность данных: использование документоориентированной модели может привести к избыточности данных, поскольку каждый документ содержит всю информацию о сущности, включая повторяющуюся информацию

Ограниченные возможности по поиску данных: поиск данных в документоориентированной модели может быть сложным, особенно при запросах, требующих обработки больших объемов данных

Пример реализации документоориентированной модели: MongoDB, CouchDB.

4. Модель графовая:

Модель графовая является одной из самых мощных и эффективных моделей данных в нереляционных базах данных. В этой модели данные представлены в виде графов, где сущности представляются в виде узлов, а связи между сущностями - в виде ребер.

- **Принцип построения:**

При использовании модели графовая данные организуются и сохраняются в виде графовой структуры, где каждый узел представляет отдельную сущность, а каждое ребро представляет связь между сущностями. Узлы могут иметь различные атрибуты, которые хранятся в виде пар ключ-значение, а ребра могут иметь свои атрибуты и направления.

Примером использования модели графовая является система социальной сети, где каждый пользователь представлен в виде узла, а отношения между пользователями - в виде ребер. Каждый узел содержит информацию о пользователе, такую как имя, возраст, место жительства, а каждое ребро

содержит информацию о связи между пользователями, такую как дружба, подписка или общие интересы.

- Преимущества:

Мощность: модель графовая позволяет эффективно работать с данными, связанными между собой, и выполнять сложные операции на графах, такие как поиск кратчайших путей, анализ связей и т.д.

Гибкость: модель графовая позволяет хранить и организовывать данные в неструктурированной форме, что делает ее гибкой и адаптивной к различным типам данных

Легкая масштабируемость: модель графовая обеспечивает легкую масштабируемость, поскольку она позволяет добавлять новые узлы и ребра без перестройки всей базы данных

- Ограничения:

Сложность хранения и обработки данных: модель графовая может быть сложной для понимания и использования, особенно при хранении и обработке больших объемов данных

Ограниченная производительность при выполнении операций запроса: выполнение сложных операций запроса на графах может быть времязатратным и требовать больших вычислительных ресурсов

Пример реализации модели графовая: Neo4j, Amazon Neptune.

Сравнение моделей данных:

В таблице ниже приведено сравнение моделей данных в нереляционных базах данных.

Модель данных	**Принцип построения**	**Преимущества**	**Ограничения**
Модель ключ-значение	Пара ключ-значение	Простота, гибкость, производительность	Ограниченный поиск, отсутствие сложных операций
Модель столбцов	Столбцы, поля	Гибкость, высокая производительность	Сложность запросов, ограниченные возможности для поиска

Документоориентированная модель	Документы, поля	Гибкость, простота, поддержка сложных запросов	Избыточность данных, ограниченные возможности для поиска
Модель графовая	Узлы, ребра	Мощность, гибкость, легкая масштабируемость	Сложность хранения и обработки данных, ограниченная производительность

Модели данных в нереляционных базах данных предлагают новые подходы к организации и хранению данных, которые могут быть более подходящими для некоторых типов приложений и использования данных. Модель ключ-значение обеспечивает простоту и высокую производительность, модель столбцов - гибкость и производительность, модель документоориентированная - гибкость и поддержку сложных запросов, а модель графовая - мощность и легкую масштабируемость. При выборе модели данных необходимо учитывать особенности приложения и требования к хранению и обработке данных, чтобы выбрать наиболее подходящую модель.

Раздел 4. Язык запросов для нереляционных баз данных

Содержание:

- Основные принципы языка запросов
- Примеры запросов в различных моделях данных
- Операции чтения и записи данных

В данном разделе мы будем изучать основные принципы языка запросов для работы с нереляционными базами данных. Нереляционные базы данных - это базы данных, которые не используют традиционную реляционную модель, а вместо этого используют другие модели данных, такие как документы, графы, столбцы или ключ-значение.

- **Основные принципы языка запросов:**

Гибкость и масштабируемость

Одним из главных принципов языка запросов для NoSQL является гибкость и масштабируемость. Это означает, что разработчик может создавать запросы, которые могут легко изменяться в зависимости от требований приложения. Кроме того, язык запросов должен поддерживать работу с большими объемами данных, позволяя выполнять запросы эффективно и быстро.

Поддержка различных моделей данных

Различные нереляционные базы данных поддерживают различные модели данных. Например, существуют базы данных на основе документов, ключ-значение, столбцов, графов и других моделей данных. Язык запросов должен быть способен работать с каждой из этих моделей данных и предоставлять соответствующие операции и возможности для извлечения данных.

Поддержка высокой производительности

Язык запросов для нереляционных баз данных должен обеспечивать высокую производительность при выполнении запросов. Это включает в себя использование оптимизаторов запросов, распределенные вычисления, возможность параллельной обработки запросов и другие техники, которые могут ускорить выполнение запросов и улучшить общую производительность системы.

- **Язык запросов для модели данных документа:**

Язык запросов для модели данных документа, такой как MongoDB, обычно основан на языке JavaScript. Запросы в таком языке позволяют выполнять операции поиска, фильтрации, сортировки и обновления данных внутри документов.

Пример запроса для MongoDB:
db.collection.find({ age: { $gt: 25 } }).sort({ name: 1 })

- **Язык запросов для модели данных графа:**

Язык запросов для модели данных графа, такой как Neo4j, позволяет выполнять операции поиска, фильтрации и анализа данных в графовых структурах. Запросы в таких языках обычно основаны на языке Cypher.

Пример запроса для Neo4j:
MATCH (n:Person)-[r:KNOWS]->(m:Person) WHERE n.name = "John" RETURN m.name

- **Язык запросов для модели данных столбцов:**

Язык запросов для модели данных столбцов, такой как Cassandra, позволяет выполнять операции чтения и записи данных из столбцов таблицы. Запросы в такой модели данных обычно основаны на SQL.

Пример запроса для Cassandra:
SELECT * FROM table WHERE column1 = "value"

- **Язык запросов для модели данных ключ-значение:**

Язык запросов для модели данных ключ-значение, такой как Redis, позволяет выполнять операции чтения и записи данных по заданному ключу. Операции в таком языке запросов обычно основаны на протоколе Redis.

Пример запроса для Redis:
GET key

- **Примеры запросов в различных моделях данных:**

Пример запроса в модели данных документа: Представим, что у нас есть нереляционная база данных, использующая модель данных документа. В этой базе данных у нас есть коллекция "users" с документами, представляющими информацию о пользователях. Давайте попробуем найти всех пользователей старше 25 лет:
db.collection.find({ age: { $gt: 25 } })

Пример запроса в модели данных графа: Представим, что у нас есть нереляционная база данных, использующая модель данных графа. В этой базе данных у нас есть узлы "Person" и связи "KNOWS", представляющие информацию о людях и их связи. Давайте попробуем найти всех людей, которых знает человек по имени "John":
MATCH (n:Person)-[r:KNOWS]->(m:Person) WHERE n.name = "John" RETURN m.name

Пример запроса в модели данных столбцов: Представим, что у нас есть нереляционная база данных, использующая модель данных столбцов. В этой базе данных у нас есть таблица "table" с колонками "column1", "column2" и "column3". Давайте попробуем найти все строки, у которых значение в колонке "column1" равно "value":

SELECT * FROM table WHERE column1 = "value"

Пример запроса в модели данных ключ-значение: Представим, что у нас есть нереляционная база данных, использующая модель данных ключ-значение. В этой базе данных у нас есть ключ "key" и соответствующее значение. Давайте попробуем получить значение по заданному ключу:

GET key

- **Операции чтения и записи данных:**

Чтение данных:

GET: операция получения значения по ключу в базе данных ключ-значение.

SELECT: операция выборки данных по заданному условию в разных моделях данных, таких как столбцовые и документоориентированные базы данных.

Запись данных:

SET: операция установки значения по ключу в базе данных ключ-значение.

INSERT: операция вставки нового документа в документоориентированные базы данных.

CREATE COLUMN FAMILY: операция создания семейства столбцов в столбцовых базах данных.

Рассмотрим подробнее синтаксис языка запросов для документоориентированной базы данных MongoDB:

Основы синтаксиса MQL

Язык запросов для MongoDB называется MongoDB Query Language (MQL), и он используется для поиска и манипуляции данных в базе данных MongoDB.

Первым шагом в работе с MQL является выбор коллекции, с которой мы будем работать. Коллекциями называются схожие документы, которые объединены общим форматом.

Синтаксис языка в своей реализации подобен языку JS. Мы обращаемся к коллекции базы данных как к объекту и можем вызывать встроенные методы этого объекта.

Для начала работы следует указать название контекста базы данных, в которой будет осуществляться работа.

Для этого используется ключевое слово use <Имя БД>

Основной единицей хранения, доступа и обработки в базе MongoDB является документ. Документ имеет набор свойств — атрибутов. Каждый атрибут в документе задается парой <Имя атрибута> : <Значение атрибута>.

Наборы документов хранятся в коллекциях. Хотя в одну коллекцию обычно помещаются документы, представляющие однотипные объекты, документы в одной коллекции не обязаны иметь одинаковые наборы атрибутов. Также не требуется совпадения типов значений одноименных атрибутов в разных документах коллекции.

Для идентификации каждый документ имеет атрибут с именем _id, содержащий уникальный ключ и набор заданных пользователем произвольных атрибутов. Если при включении документа ключ не задан, то он генерируется автоматически.

Автоматически созданный ключ соответствует 24-разрядному шестнадцатеричному числу. В целом для задания документа используется формат JSON (JavaScript Object Notation).

Основная работа осуществляется далее с содержимым базы данных, структурированным по сущностям-коллекциям. Для просмотра списка существующих коллекций в БД можно воспользоваться командой show collections.

Для вывода названия текущей используемой БД можно воспользоваться командой db.getName ()

Команды создания и обработки коллекции в БД:

db.createCollection (<Имя коллекции>. [,{size:… [, capped: …, max: …}]]) – создание новой коллекции

db.getCollectionNames () – вывод списка имен коллекций в виде массива.

db.<Имя коллекции>.find () – вывод документов из коллекции

db.<Имя существ. коллекции>. renameCollection ("<Новое имя коллекции>") – переименование коллекции

db.<Имя коллекции>.drop () – удаление коллекции.

Количественные характеристики коллекции выводит ее метод stats (): >db.<Имя коллекции>.stats ()

Добавление нового документа в коллекцию выполняет метод Insert (): db.<Имя коллекции>.insert (<Документ в формате JSON>) или db.<Имя коллекции>.insert ({<Имя атрибута>:<Значение> [,..}).

В момент включения документа сервер MongoDB автоматически создал для него ключевой атрибут _id, имеющий специальный тип ObjectId с

уникальным значением ObjectId. Тем не менее, пользователь при добавлении документа методом insert может задать свое уникальное в коллекции значение атрибута _id.

Другой способ наполнения коллекции предоставляет утилита mongoimport.exe. Источником данных для mongoimport служит текстовый файл. Кодировка символов в файле-источнике должна быть в формате UTF-8, преобразующем 16-битные символы Юникода в 8-битные. Имя загружаемой базы, коллекции и исходный файл задаются значениями ключей в вызове утилиты.

Документы в загружаемом файле могут быть заданы не только в JSON, но и в форматах, предназначенных для представления таблиц: csv, tsv. Справку по всем ключам утилиты можно получить ее вызовом с ключом — help: mongoimport. exe — help. Для вывода документов из коллекции в текстовый файл предназначена утилита mongoexport.exe.

Основным запросом является запрос поиска, которым в синтаксисе MQL является ключевое слово find, которое вызывается как метод объекта коллекции.

Целью запроса к базе MongoDB являются документы из определенной коллекции. Для поиска (отбора) документов предназначены методы коллекции find и findOne.

Вызовы методов find и findOne имеют одинаковую структуру и содержат критерий отбора документов (селектор), имена выводимых атрибутов и правило сортировки выбранных документов. Различие методов в том, что метод find возвращает все документы, соответствующие критерию отбора, а метод findOne — только один (первый) документ из удовлетворяющих селектору.

Основной синтаксис вызова метода find () имеет вид:

db.<Коллекция>.find ([{<Селектор>} [, {<Список атрибутов>}]]);

В простейшем виде запрос db.<Коллекция>.find (); возвращает все документы заданной коллекции.

Отсутствие селектора и списка атрибутов приводит к выводу всех документов с полными наборами их атрибутов.

Параметр <список атрибутов> содержит имена выводимых или, напротив, исключаемых из вывода атрибутов найденных документов. Кроме параметров find и findOne имеют присоединенные методы, выполняющие сортировку, ограничение количества и способ вывода найденных документов.

С присоединенными методами запрос имеет вид:

db.<Коллекция>.find ([{<Селектор>} [, {<Список атрибутов>}]]) [.sort ({<Атрибут>: 1 |-1, …})] [.limit (<Число выводимых документов>)] [.skip (<Количество пропускаемых документов>)] [.pretty ()]; — выводит каждый атрибут в отдельной строке.

Количество документов, удовлетворяющих запросу, возвращает метод коллекции count: db.<Коллекция>.count ({<Селектор>}).

Далее мы можем выполнять различные операции с данными, используя операторы MQL. Рассмотрим некоторые из них:

Оператор $eq - это оператор равенства и он используется для поиска документов, значения указанных полей которых равны заданным значениям. Например, мы можем найти все документы, у которых поле "age" равно 25:

```
db.collection.find({ age: { $eq: 25 } })
```

Оператор $ne - это оператор неравенства и он используется для поиска документов, значения указанных полей которых не равны заданным значениям. Название оператора является «говорящим» и может быть прочитано как not equal. Например, мы можем найти все документы, у которых поле "status" не равно "inactive":

```
db.collection.find({ status: { $ne: "inactive" } })
```

Оператор $gt - это оператор больше и он используется для поиска документов, значения указанных полей которых больше заданных значений. Название оператора является «говорящим» и может быть прочитано как greater than. Например, мы можем найти все документы, у которых поле "salary" больше 5000:

```
db.collection.find({ salary: { $gt: 5000 } })
```

Оператор $lt - это оператор меньше и он используется для поиска документов, значения указанных полей которых меньше заданных значений. Название оператора является «говорящим» и может быть прочитано как less than. Например, мы можем найти все документы, у которых поле "rating" меньше 4.5:

```
db.collection.find({ rating: { $lt: 4.5 } })
```

Оператор $in - это оператор включения и он используется для поиска документов, значения указанных полей которых содержатся в заданных значениях. Например, мы можем найти все документы, у которых поле "name" содержит значение "John" или "Jane":

```
db.collection.find({ name: { $in: ["John", "Jane"] } })
```

Оператор $and - это оператор логического "И" и он используется для комбинирования нескольких условий. Например, мы можем найти все документы, у которых поле "age" равно 25 и поле "status" равно "active":

```
db.collection.find({ $and: [{ age: { $eq: 25 } }, { status: { $eq: "active" } }] })
```

Оператор $or - это оператор логического "ИЛИ" и он используется для поиска документов, которые соответствуют хотя бы одному из заданных

условий. Например, мы можем найти все документы, у которых поле "age" равно 25 или поле "age" равно 30:

db.collection.find({ $or: [{ age: { $eq: 25 } }, { age: { $eq: 30 } }] })

Команда db.collection.help () позволяет получить справку по методам, доступных к использованию для объекта коллекции.

Список выводимых атрибутов задается вторым параметром в методах find и findOne. В параметре атрибут, который необходимо вывести, указывается со значением 1, а не требуемый для вывода — 0.

db.<коллекция>.find ([{<селектор>}[, {<имя атрибута>: 1 |0, …}]]);

Для сортировки результата запроса find () используется присоединенный метод db.<Коллекция>.find ().sort({<Атрибут>: 1 |-1,..}).

Указание атрибута со значением 1 приводит к сортировке по возрастанию, а -1 (минус 1) — по убыванию значений атрибута.

Это лишь некоторые из операторов MQL. Для детального ознакомления рекомендуется ознакомиться с официальной документацией MongoDB для получения полного списка операторов и их подробного описания.

Раздел 5. Распределенные нереляционные базы данных

Содержание:

- Распределение данных и партицирование
- Репликация данных
- Консистентность и доступность данных
- Управление конфликтами

- **Распределение данных и партицирование**

Распределение данных и партицирование являются одними из ключевых концепций в области нереляционных баз данных. Они позволяют эффективно хранить и обрабатывать большие объемы данных.

Распределение данных относится к способу физического распределения данных на несколько узлов или серверов. Это позволяет балансировать нагрузку и повышать производительность обработки запросов. В нереляционных базах данных распределение данных осуществляется на уровне кластера, то есть на нескольких серверах или узлах.

Существуют различные стратегии распределения данных. Одна из них - это горизонтальное распределение или шардирование (sharding). При этом данные разделяются на некоторое количество горизонтальных частей, и каждая часть хранится на отдельном сервере. Горизонтальное распределение особенно полезно при работе с большими объемами данных, так как позволяет параллельно обрабатывать запросы к разным частям данных.

Стратегия распределения данных может быть основана на различных параметрах, таких как ключи, хэши, диапазоны и др.

Например, можно использовать хэш-функции для распределения данных по различным узлам на основе их значений. Это позволяет равномерно распределить данные между узлами и обеспечить эффективное выполнение запросов.

Партицирование, с другой стороны, относится к логическому разделению данных на отдельные разделы внутри базы данных. Каждый раздел (partition) содержит свою собственную подмножество данных. Партицирование позволяет сократить время выполнения запросов и обеспечить лучшую масштабируемость.

Существует несколько видов партицирования данных. Одним из самых распространенных является горизонтальное партицирование, при котором данные разделяются на несколько горизонтальных частей на основе определенного критерия, например, значения ключа или диапазона значений.

Это позволяет улучшить производительность обработки запросов, так как запросы могут быть выполнены отдельно для каждой части данных.

В вертикальном партицировании данные разделяются на разные таблицы внутри базы данных на основе атрибутов или столбцов. Например, можно отделить столбцы с базовыми данными от столбцов со сложными вычисляемыми значениями. Это позволяет эффективно использовать ресурсы хранения и улучшает производительность запросов.

Обычно распределение данных и партицирование используются вместе для обеспечения оптимальной производительности и масштабируемости системы. Например, данные могут быть сначала разделены на горизонтальные части, а затем каждая часть может быть разделена на более мелкие разделы для лучшей организации данных.

Распределение данных и партицирование являются важными концепциями для эффективного хранения и обработки больших объемов данных в нереляционных базах данных. Они позволяют улучшить производительность запросов и обеспечить лучшую масштабируемость системы. Выбор стратегии распределения данных и партицирования зависит от требований к системе и ее конкретного контекста использования.

- **Репликация данных**

Репликация данных является одной из ключевых функций, предоставляемых нереляционными базами данных. Она позволяет создать несколько копий данных и распределить их на различных узлах в сети. Такой подход позволяет улучшить производительность системы, обеспечить отказоустойчивость и обработку большого количества запросов.

Основные принципы репликации данных:

1. *Мастер-слейв репликация*: Это самый распространенный подход к репликации данных. В этой модели некоторые узлы назначаются в качестве "мастер" узлов, которые принимают записи данных и реплицируют их на другие "слейв" узлы. Слейв узлы используются в основном для чтения данных и предоставления обслуживания пользователей. Мастер узлы обеспечивают согласованность данных и реплицируют изменения на всех слейв узлах.
2. *Мульти-мастер репликация:* В этом подходе все узлы могут принимать как запись, так и чтение данных. Это позволяет балансировать нагрузку и повышать отказоустойчивость системы. Однако, возникают проблемы с согласованностью данных, так как несколько мастер узлов могут конфликтовать при обновлении одного и того же набора данных.
3. *P2P репликация:* В этом типе репликации данных каждый узел является как мастером, так и слейвом. Он реплицирует данные на другие узлы и принимает обновления от других узлов. Это позволяет распределить данные

и нагрузку между узлами системы. Однако, возникают проблемы с согласованностью данных и конфликтами при обновлении данных.

Способы реализации репликации данных:

1. *Master-Slave репликация:* В этом случае мастер узел принимает записи данных от клиентов и реплицирует их на один или несколько слейв узлов.
   - Однонаправленная репликация: Мастер узел отправляет обновления на слейв узлы, но слейв узлы не отправляют обновления обратно мастер узлу. Это позволяет использовать слейв узлы только для чтения данных и тем самым увеличивает производительность системы.
   - Двухнаправленная репликация: Обновления данных могут быть отправлены как с мастер узла на слейв узлы, так и наоборот. Это обеспечивает согласованность данных, но требует больше ресурсов для обработки обновлений.
2. *Мульти-мастер репликация:* В этом случае все узлы могут принимать записи данных и реплицировать их на другие узлы.
   - Конфликтное разрешение: При возникновении конфликта в обновлениях данных, необходимо иметь механизм разрешения конфликтов. Это может быть либо уровнем приложения, либо встроенными механизмами базы данных.
   - Транзакционная репликация: В этом случае обновления данных отправляются на другие узлы внутри транзакции. Если одно из обновлений не проходит, транзакция откатывается на всех узлах. Это обеспечивает целостность данных, но может привести к задержкам при выполнении транзакций.
3. *P2P репликация*: В этом случае каждый узел является и мастером, и слейвом. Каждый узел реплицирует данные на другие узлы и принимает обновления от других узлов.
   - Гарантированная доставка: При распределении обновлений между узлами требуется механизм, который гарантирует доставку всех обновлений. Это может быть реализовано через механизмы проверки данных или журнализации обновлений.
   - Конфликтное разрешение: Конфликты при обновлении данных могут возникать, когда несколько узлов пытаются обновить одни и те же данные одновременно. Необходимо иметь механизм разрешения конфликтов, чтобы избежать потери данных или сбоев в системе.

Репликация данных в нереляционных базах данных является важной функцией, которая позволяет достичь высокой производительности, отказоустойчивости и распределения нагрузки между узлами. Различные типы репликации данных обладают своими преимуществами и ограничениями, и выбор конкретного типа зависит от требований и особенностей конкретной системы. Важно учитывать согласованность данных и разрешение конфликтов

при репликации данных, чтобы обеспечить целостность и надежность работы системы.

- **Консистентность и доступность данных**

Консистентность и доступность данных - два важных понятия в контексте нереляционных баз данных. Оба аспекта имеют влияние на способность базы данных функционировать в условиях высоких нагрузок и обеспечить требуемую надежность и производительность.

Консистентность означает, что данные, хранящиеся в базе данных, всегда находятся в согласованном состоянии. Это означает, что при выполнении определенных операций, таких как добавление, обновление или удаление данных, база данных должна гарантировать, что после завершения операции данные останутся в согласованном состоянии. Например, если мы добавляем новую запись в базу данных, все связанные данные и связи должны быть корректно обновлены, чтобы поддерживать целостность базы данных.

Доступность данных, с другой стороны, определяет, насколько доступны данные и возможность обращаться к ним в любое время. Доступность играет важную роль в распределенных системах или системах с большими объемами данных, где высокая доступность имеет критическое значение. Это означает, что система должна быть способна обрабатывать запросы и обеспечивать доступ к данным даже в случае отказа в некоторых компонентах или при возникновении сбоев.

Основной вызов для нереляционных баз данных заключается в достижении согласованности и доступности данных одновременно. Обычно существует некоторая компромиссная точка между этими двумя аспектами, и инженеры по базам данных должны учитывать специфические требования и контекст при проектировании и выборе системы баз данных.

Одним из популярных решений для достижения высокой консистентности данных являются сильные согласованности. Это означает, что данные будут обновляться в единую точку в любой момент времени и будут доступны всем клиентам только после завершения операции обновления. Данный подход обеспечивает высокую консистентность данных, но может снижать доступность, особенно в условиях нагрузки.

Другой подход - использование слабых согласованностей. В случае слабых согласованностей система может устанавливать определенные ограничения на потенциально несогласованные данные или сообщать о конфликтах между разными операциями обновления. Это может обеспечить более высокую доступность, но может приводить к возникновению временных расхождений между данными или неоднозначности в системе.

Компромиссный вариант может быть достигнут с использованием уровней изоляции транзакций и механизмов репликации данных. Уровни изоляции транзакций определяют, насколько видимы изменения данных для других операций или клиентов, работающих параллельно. Например, уровень "Read Committed" позволяет читать только подтвержденные данные, что

может приводить к возникновению несогласованных или устаревших данных. С другой стороны, уровень "Serializable" гарантирует полную изоляцию и согласованность данных, но может снижать производительность.

Механизмы репликации данных могут использоваться для обеспечения высокой доступности данных. Репликация позволяет создать несколько копий данных на разных серверах или узлах в кластере. При отказе одного узла данные все еще могут быть доступны на других узлах. Однако репликация может быть подвержена задержкам в обновлении данных между узлами, что может привести к некоторым конфликтам или расхождениям данных.

В завершение, консистентность и доступность данных являются двумя важнейшими факторами, которые следует учитывать при проектировании и выборе нереляционных баз данных. Оба аспекта являются взаимосвязанными и требуют компромисса, чтобы найти оптимальное решение в соответствии с требованиями и контекстом системы.

- **Управление конфликтами**

Конфликт в контексте баз данных возникает, когда две или более операции пытаются изменить одни и те же данные одновременно. Это может привести к непредсказуемым и некорректным результатам, которые могут повлиять на целостность данных и работоспособность системы.

Уровни конфликтов:

Уровень конфликтов может быть разделен на несколько уровней.

Первый уровень - это конфликты чтения/записи (Read/Write conflicts), которые возникают, когда одна операция пытается прочитать данные, которые другая операция только что изменила или изменить данные, которые другая операция уже прочитала.

Второй уровень - это конфликты записи/записи (Write/Write conflicts), которые возникают, когда две или более операции пытаются одновременно изменить одни и те же данные.

Наконец, *третий уровень* - это конфликты совместного чтения (Shared Read conflicts), которые возникают, когда одна операция пытается прочитать данные, которые другая операция только что прочитала.

Методы управления конфликтами:

Существует несколько методов управления конфликтами в нереляционных базах данных. Один из таких методов - это использование транзакций и блокировок. Транзакции позволяют выполнять группу операций как одну атомарную единицу, что позволяет управлять конфликтами. Блокировки могут быть использованы для блокировки доступа к данным во время выполнения операций, предотвращая тем самым конфликты. Еще одним методом является оптимистическое управление конфликтами, которое предполагает, что конфликты происходят редко, и данные могут быть сохранены без блокировок. Однако, если возникнет конфликт, система должна

предусмотреть способы его разрешения, например, через проверку целостности данных или резервные копии.

Применение конфликтных моделей:

Каждая база данных может иметь свою собственную конфликтную модель, которая определяет правила и поведение при возникновении конфликтов. Конфликтные модели могут варьироваться в зависимости от типа базы данных, таких как графовые или документные базы данных. Например, в графовой базе данных конфликт может возникнуть, если две операции пытаются изменить одно и то же ребро или вершину. Конфликтные модели также могут определять конкретные стратегии разрешения конфликтов, например, последовательное или параллельное выполнение операций.

Раздел 6. Применение нереляционных баз данных

Содержание:

- Использование в веб-приложениях
- Big Data и обработка больших объемов данных
- IoT и сенсорные сети
- Примеры реальных применений

Применение нереляционных баз данных становится все более популярным и востребованным среди разработчиков и аналитиков данных. Нереляционные базы данных или NoSQL базы данных предлагают гибкие и масштабируемые решения для хранения и обработки данных, которые не соответствуют традиционной структуре таблиц и строк, используемой в реляционных базах данных.

Вообще термин NoSQL обозначает «не только SQL» (Not Only SQL), характеризуя ответвление от традиционного подхода к проектированию баз данных. Изначально так называлась опенсорсная база данных, созданная Карло Строззи, которая хранила все данные как ASCII-файлы, а вместо SQL-запросов доступа к данным использовала шелловские скрипты.

В начале 2000-х годов Google построил свою поисковую систему и прочие сервисы, решив проблемы масштабируемости и параллельной обработки больших объёмов данных. Так была создана распределённые файловая и координирующая системы, а также колоночное хранилище (column family store), основанное на вычислительной модели MapReduce.

MapReduce можно по праву назвать главной технологией Big Data, т.к. она изначально ориентирована на параллельные вычисления в распределенных кластерах. Суть MapReduce состоит в разделении информационного массива на части, параллельной обработки каждой части на отдельном узле и финального объединения всех результатов.

Программы, использующие MapReduce, автоматически распараллеливаются и исполняются на распределенных узлах кластера, при этом исполнительная система сама заботится о деталях реализации (разбиение входных данных на части, разделение задач по узлам кластера, обработка сбоев и сообщение между распределенными компьютерами). Благодаря этому программисты могут легко и эффективно использовать ресурсы распределённых Big Data систем.

Технология практически универсальна: она может использоваться для индексации веб-контента, подсчета слов в большом файле, счётчиков частоты обращений к заданному адресу, вычисления объём всех веб-страниц с каждого URL-адреса конкретного хост-узла, создания списка всех адресов с

необходимыми данными и прочих задач обработки огромных массивов распределенной информации. Также к областям применения MapReduce относится распределённый поиск и сортировка данных, обращение графа веб-ссылок, обработка статистики логов сети, построение инвертированных индексов, кластеризация документов, машинное обучение и статистический машинный перевод. Также MapReduce адаптирована под многопроцессорные системы, добровольные вычислительные, динамические облачные и мобильные среды.

Авторами этой вычислительной модели считаются сотрудники Google Джеффри Дин (Jeffrey Dean) и Санджай Гемават (Sanjay Ghemawat), взявшие за основу две процедуры функционального программирования: map, применяющая нужную функцию к каждому элементу списка, и reduce, объединяющая результаты работы map. В процессе вычисления множество входных пар ключ/значение преобразуется в множество выходных пар ключ/значение.

После того, как корпорация Google опубликовала описание этих технологий, они стали очень популярны у разработчиков открытого программного обеспечения. В результате этого был создан Apache Hadoop и запущены основные связанные с ним проекты. Например, в 2007 году другой ИТ-гигант, Amazon.com, опубликовав статьи о своей высокодоступной базе данных Amazon DynamoDB. Далее в эту гонку NoSQL- технологий для управления большими данными включилось множество корпораций: IBM, Facebook, Netflix, eBay, Hulu, Yahoo! и другие ИТ-компаний со своими проприетарными и открытыми решениями

NoSQL-базы данных появились в эпоху больших объемов информации и широкополосного интернета. Они имеют преимущества перед классическими реляционными БД, но полностью заменить их не могут, особенно в таких областях, где требуется высокая надежность транзакций. Тем не менее отличная масштабируемость, гибкость и простота делают их популярным решением в самых различных сферах, от электронной коммерции и научной деятельности до игровой индустрии и интернета вещей.

Одним из наиболее распространенных применений нереляционных баз данных является их использование в веб-приложениях. Традиционно, реляционные базы данных предлагают жесткую схему данных, что означает, что все данные должны соответствовать определенной структуре. Веб-приложения, особенно в сфере социальных сетей и электронной коммерции, требуют гибкости в структуре данных, поскольку пользователи могут добавлять, изменять или удалять различные типы данных. В этих случаях нереляционные базы данных, такие как ключ-значение (Key-Value) базы данных и документоориентированные базы данных, могут предложить гибкое

хранение и обработку данных без необходимости изменять схему базы данных при каждом изменении данных в приложении.

Еще одним применением нереляционных баз данных является обработка больших объемов данных, что связано с идеей Big Data. Обработка больших данных требует эффективного хранения и обработки данных, которые могут быть весьма разнообразными, с неопределенной схемой или большим объемом. При работе с Big Data, традиционные реляционные базы данных могут столкнуться с ограничениями производительности или масштабируемости. Нереляционные базы данных, такие как графовые базы данных и колоночные базы данных, предлагают более эффективное хранение и обработку данных, даже при очень больших объемах.

В Big Data, обычно, имеются множественные источники данных, где информация поступает в больших объемах, со сложной структурой и высокой скоростью. NoSQL базы данных спроектированы специально для работы с такими типами данных, предлагая ряд преимуществ по сравнению с реляционными базами данных.

Одно из применений NoSQL в Big Data - это хранение и обработка больших объемов данных в реальном времени. По сравнению с традиционными SQL базами данных, NoSQL базы данных могут легко масштабироваться, чтобы обрабатывать миллионы и миллиарды записей в секунду. Это делает их идеальным решением для систем, собирающих и анализирующих данные в режиме реального времени, таких как системы мониторинга сенсорных сетей или системы аналитики социальных медиа.

NoSQL также применяется в системах аналитики Big Data, где требуется хранение и обработка больших объемов неструктурированных данных, таких как тексты, изображения, аудио и видео. NoSQL базы данных предоставляют гибкую схему, позволяющую хранить данные разных типов и форматов, без необходимости предварительного определения схемы данных. Это позволяет легко добавлять и изменять структуру данных по мере необходимости, что особенно полезно в Big Data сценариях, где структура данных может меняться со временем.

Другим применением NoSQL в Big Data является обработка и анализ графовых данных. Графовые базы данных, основанные на принципах NoSQL, позволяют хранить и обрабатывать связи между данными, такие как связи в социальных сетях, сети транспорта или полей связей в медицинских данных. NoSQL базы данных используют специализированные алгоритмы для обработки и анализа графовых данных, что помогает в поиске связей, обнаружении паттернов и выявлении структурных зависимостей в данных.

Следующим примером применения нереляционных баз данных является IoT (Internet of Things) и сенсорные сети. IoT представляет собой концепцию о взаимодействии различных физических объектов, таких как устройства, датчики, автомобили, домашние устройства и т.д., через интернет. Эти объекты генерируют огромные объемы данных, которые включают в себя информацию о состоянии объекта, его местоположение, температуру, влажность и многое другое. Нереляционные базы данных, такие как временные ряды баз данных и базы данных для геопространственных данных, предлагают эффективное хранение и обработку такого типа данных, обеспечивая высокую производительность и надежность.

Примеры реальных применений нереляционных баз данных включают такие компании, как Facebook, Amazon и Google. Facebook использует нереляционные базы данных для хранения и обработки большого объема данных пользователей, таких как сообщения, комментарии и лайки. Amazon использует нереляционные базы данных для обеспечения гибкого хранения и обработки данных клиентов, а также для поддержки своей коммерческой инфраструктуры, такой как инвентаризация и отслеживание заказов. Google использует нереляционные базы данных для обработки больших объемов данных, связанных с индексацией и поиском веб-страниц.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

Лабораторная №1. Создание и конфигурирование окружения для работы с документоориентированной СУБД MongoDB

Цель работы

Целью данной лабораторной работы является ознакомление с базой данных MongoDB, изучение основных операций CRUD (Create, Read, Update, Delete), а также использование инструментов для работы с MongoDB.

Подготовка к работе

1. Убедитесь, что на вашем компьютере установлена MongoDB. Если нет, скачайте и установите ее с официального сайта MongoDB.
2. Установите клиент для работы с MongoDB, например, MongoDB Compass, Robo 3T или используйте командную строку для взаимодействия с базой данных.

Задания

1. Знакомство с оболочкой

Запустите MongoDB сервер, если он не запущен. Вы можете использовать команду mongod для запуска сервера.

Запустите MongoDB Compass. При запуске укажите параметры подключения к серверу MongoDB (адрес сервера, порт и другие настройки).

2. Мониторинг и администрирование

1. Изучите раздел "Performance" в MongoDB Compass. Отследите запросы и операции, происходящие в реальном времени.
2. В разделе "Performance" также изучите индексы и их использование.
3. Ознакомьтесь с разделом "Users" в MongoDB Compass. Создайте нового пользователя с административными правами.

Завершение работы

По завершении выполнения всех заданий, сохраните изменения в базе данных и закройте соединение с MongoDB.

Лабораторная №2. Создание базы данных с использованием MongoDB

Цель работы

Целью данной лабораторной работы является ознакомление с базой данных MongoDB, создание базы данных и импорт первых данных для дальнейшей работы.

Подготовка к работе

1. Убедитесь, что на вашем компьютере установлена MongoDB. Если нет, скачайте и установите ее с официального сайта MongoDB.
2. Установите клиент для работы с MongoDB, например, MongoDB Compass, Robo 3T или используйте командную строку для взаимодействия с базой данных.

Задания

1. Создание базы данных и коллекции

Создайте новую базу данных с названием "mydb" и коллекцию "students". В коллекции "students" создайте следующие документы (записи):

```
{ "name": "Иванов Иван", "age": 25, "major": "Информатика" },
{ "name": "Петрова Мария", "age": 22, "major": "Математика" },
{ "name": "Сидоров Алексей", "age": 28, "major": "Физика" }
```

2. Операции чтения данных

- Напишите запрос для получения всех студентов из коллекции "students".
- Напишите запрос для получения студента по имени "Иванов Иван".
- Напишите запрос для получения студентов, возраст которых младше 25 лет.

Завершение работы

По завершении выполнения всех заданий, сохраните изменения в базе данных и закройте соединение с MongoDB.

Лабораторная №3. Запросы на изменение и удаление данных

Цель работы

Целью данной лабораторной работы является ознакомление с базой данных MongoDB, изучение основных операций CRUD (Create, Update, Delete)

Задания

1. Операции обновления данных

- Обновите возраст студента "Петрова Мария" на 23 года.
- Добавьте поле "город" со значением "Москва" для студента "Иванов Иван".
- Повторите запросы из предыдущей работы.
- Напишите запрос для получения студентов из города Москва. Оставьте данные разнородными (город указан только у одного студента).
- Напишите запрос для получения студентов из любого другого города, кроме Москвы. Оставьте данные разнородными.
- Добавьте всем студентам поле город.
- Измените поле город для всех студентов таким образом, чтобы параметр содержал вложенную информацию о стране (заменить строку на объект).

2. Операции удаления данных

- Удалите студента "Сидоров Алексей" из коллекции "students".
- Удалите поле city у студента "Петрова Мария".

3. Дополнительные задания

- Создайте новую коллекцию "courses" и добавьте несколько документов, представляющих различные курсы.
- Напишите запрос для получения списка всех курсов.
- Напишите запрос для получения списка курсов, которые посещает студент "Иванов Иван".
- Напишите запрос, вычисляющий средний возраст всех студентов из всех документов коллекции.

Завершение работы

По завершении выполнения всех заданий, сохраните изменения в базе данных и закройте соединение с MongoDB.

Лабораторная №4. Создание схемы коллекций, импорт данных, написание запросов для поиска, сортировки и фильтрации данных

Цель работы

Целью данной лабораторной работы является ознакомление с запросами к базе данных MongoDB, включая операции фильтрации, сортировки и использование селекторов.

Подготовка к работе

1. Убедитесь, что на вашем компьютере установлена MongoDB. Если нет, скачайте и установите ее с официального сайта MongoDB.
2. Запустите MongoDB сервер, если он не запущен. Вы можете использовать команду **mongod** для запуска сервера.
3. Для выполнения запросов и работы с базой данных используйте MongoDB Compass или любой другой удобный клиент.

Задания

1. Запросы с фильтрацией

1. Создайте базу данных с названием "ecommerce" и коллекцию "products".
2. В коллекции "products" создайте несколько документов, описывающих товары. Каждый документ должен содержать поля:
    - "name" (название товара)
    - "category" (категория товара)
    - "price" (цена товара)
    - "stock" (количество на складе)
3. Напишите запросы для выполнения следующих задач с фильтрацией:
    - Получение всех товаров определенной категории.
    - Получение товаров с ценой ниже заданной суммы.
    - Получение товаров, количество которых на складе меньше заданного значения.

2. Запросы с сортировкой

1. Используя ту же коллекцию "products", напишите запросы для выполнения следующих задач с сортировкой:
    - Получение списка товаров, отсортированных по цене по возрастанию.
    - Получение списка товаров, отсортированных по количеству на складе по убыванию.

3. Использование селекторов

1. Напишите запрос для получения списка товаров, но в результатах отобразите только названия и цены товаров, исключив остальные поля.
2. Напишите запрос для получения информации о товаре с наибольшим количеством на складе. Выведите все поля этого товара.

Завершение работы

По завершении выполнения всех заданий, сохраните изменения в базе данных, закройте соединение с клиентом MongoDB и остановите сервер MongoDB.