Тема 6. Введение в NoSQL и MongoDB.

Цель занятия:

Получить представление о том, как использовать транзакции и ограничения для обеспечения целостности данных в MySQL.

Учебные вопросы:

1. Введение в NoSQL.
2. Определение MongoDB как NoSQL базы данных.
3. Сравнение с реляционными базами данных.
4. Области применения MongoDB.
5. Установка и настройка MongoDB.
6. Основные понятия MongoDB.
7. Создание базы данных и первой коллекции.

1. Введение в NoSQL.

NoSQL (Not Only SQL) — это категория систем управления базами данных, которые обеспечивают гибкие модели хранения данных и подходы к их обработке, отличные от традиционных реляционных баз данных.

NoSQL базы данных позволяют работать с неструктурированными и полуструктурированными данными, поддерживают горизонтальное масштабирование и высокую производительность.

Основные типы NoSQL баз данных включают:

- Документные (например, MongoDB)
- Ключ-значение (например, Redis)
- Колонковые (например, Apache Cassandra)
- Графовые (например, Neo4j)

Причины роста популярности NoSQL:

- Гибкость схемы: NoSQL базы данных позволяют использовать динамическую схему, что облегчает адаптацию к изменениям в данных.
- Масштабируемость: горизонтальное масштабирование позволяет легко добавлять новые серверы и поддерживать высокую производительность при увеличении объема данных.
- Обработка больших данных: NoSQL решения оптимизированы для работы с большими объемами неструктурированных данных, что делает их идеальными для приложений Big Data.
- Высокая доступность: многие NoSQL базы данных предлагают репликацию и распределенное хранение, что повышает доступность и отказоустойчивость.
- Поддержка разнообразных типов данных: NoSQL базы данных могут обрабатывать текст, изображения, видео и другие форматы, что делает их более универсальными для современных приложений.

2. Определение MongoDB как NoSQL базы данных.

MongoDB — это документно-ориентированная NoSQL база данных, предназначенная для хранения и обработки больших объемов данных.

Она использует гибкую схему данных, что позволяет разработчикам хранить данные в формате, удобном для приложений, и быстро адаптироваться к изменениям в требованиях.

MongoDB предоставляет мощные инструменты для масштабирования, высокую производительность и поддержку разнообразных типов данных.

Основные характеристики MongoDB:

- Документная модель данных: Данные хранятся в формате документов (в основном в формате JSON), что позволяет легко хранить и извлекать сложные структуры данных.
- Гибкость схемы: MongoDB не требует жесткой схемы, что дает возможность изменять структуру данных по мере необходимости.
- Высокая производительность: MongoDB обеспечивает быструю запись и чтение данных благодаря индексированию и эффективному управлению памятью.
- Масштабируемость: Поддержка горизонтального масштабирования позволяет легко добавлять новые серверы и обрабатывать большие объемы данных.
- Репликация и отказоустойчивость: MongoDB поддерживает репликацию данных для повышения доступности и защиты от потери данных.
- Агрегация и обработка данных: Встроенные функции агрегации позволяют выполнять сложные операции над данными непосредственно в базе данных.

Архитектура MongoDB

Архитектура MongoDB построена на основе документно-ориентированной модели, что означает, что данные хранятся в виде документов, которые могут иметь произвольную структуру.

Каждый документ представляет собой структуру данных, содержащую пары ключ-значение, аналогично JSON. Документы организованы в коллекции, которые, в свою очередь, группируются в базы данных.

Документы: Основная единица хранения данных в MongoDB - документ. Каждый документ имеет уникальный идентификатор (ObjectID) и может содержать различные поля и типы данных.

Коллекции: Группа документов, которая может быть рассмотрена как аналог таблицы в реляционных базах данных. Коллекции могут содержать документы с разными структурами.

Базы данных: Содержат коллекции и представляют собой контейнер для организации данных.

JSON-подобные документы и BSON

JSON (JavaScript Object Notation): Это легкий формат обмена данными, который легко читается человеком и воспринимается компьютером. MongoDB использует JSON-подобный формат для представления документов.

BSON (Binary JSON): Это бинарный формат, используемый MongoDB для хранения данных. BSON расширяет возможности JSON, добавляя поддержку таких типов данных, как даты, двоичные данные и другие. Он оптимизирован для быстрого чтения и записи, что повышает производительность при работе с данными.

```
{
  "user": {
    "id": 1,
    "name": "Иван Иванов",
    "email": "ivan.ivanov@example.com",
    "age": 30,
    "address": {
      "street": "Улица Пушкина",
      "city": "Москва",
      "postalCode": "123456"
    },
    "isActive": true,
    "createdAt": "2024-10-01T12:00:00Z"
  }
}
```

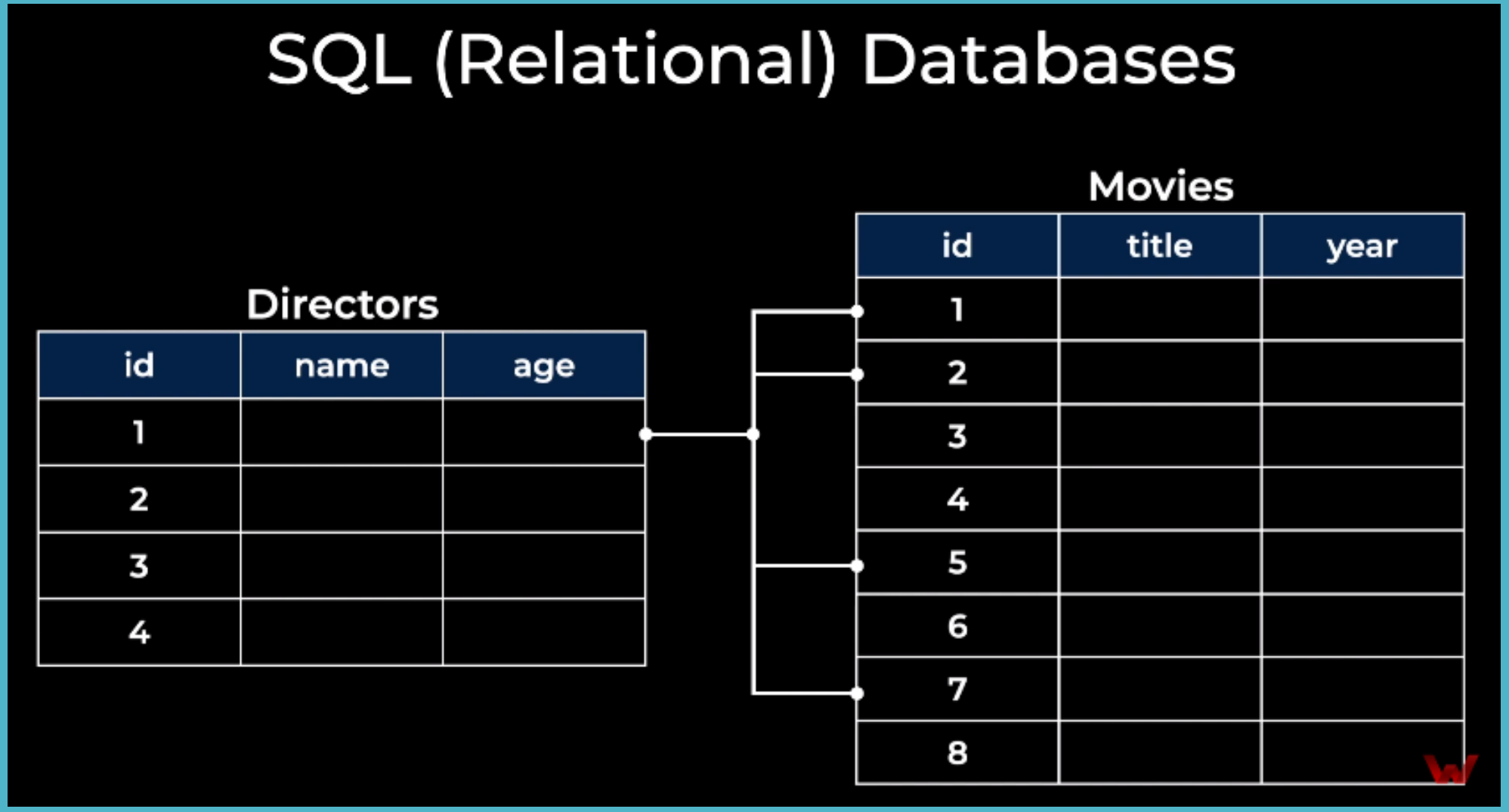
3. Сравнение с реляционными базами данных.

MongoDB и реляционные базы данных (РБД) представляют два разных подхода к хранению и управлению данными.

Реляционные базы данных используют строго структурированные таблицы, а MongoDB — гибкие документы.

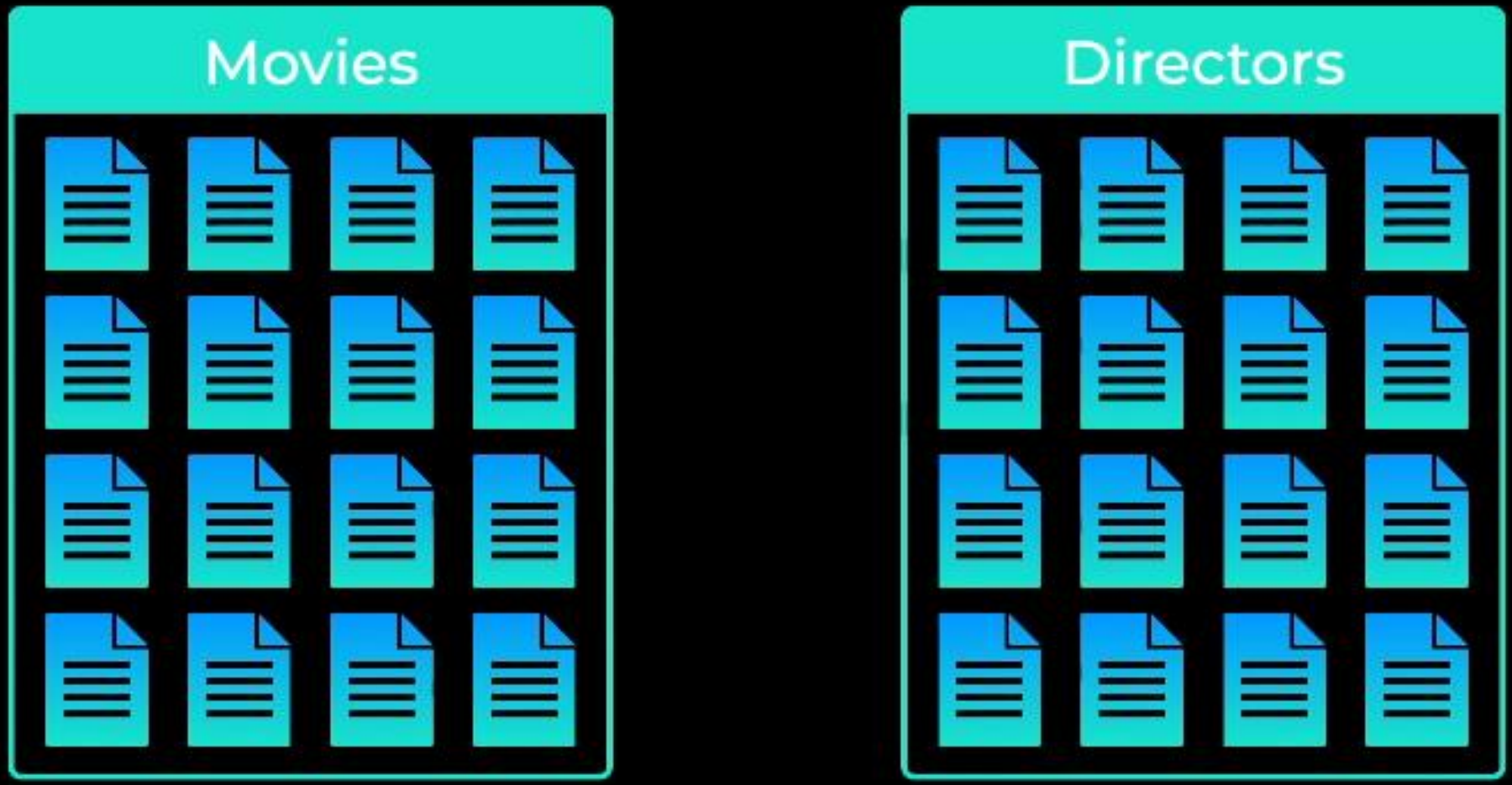
Параметр	Реляционные БД (SQL)	MongoDB (NoSQL)
Модель данных	Таблицы, строки и столбцы	Документы (JSON/BSON)
Схема данных	Фиксированная (предопределена)	Гибкая (динамическая)
Язык запросов	SQL	MongoDB Query Language (MQL)
Транзакции	Поддержка сложных транзакций	Ограниченная поддержка транзакций (начиная с версии 4.0 улучшена)
Масштабирование	Вертикальное масштабирование	Горизонтальное масштабирование (шардинг)
Связи между данными	Четкие связи (например, с помощью внешних ключей)	Связи через вложенные документы или ссылки

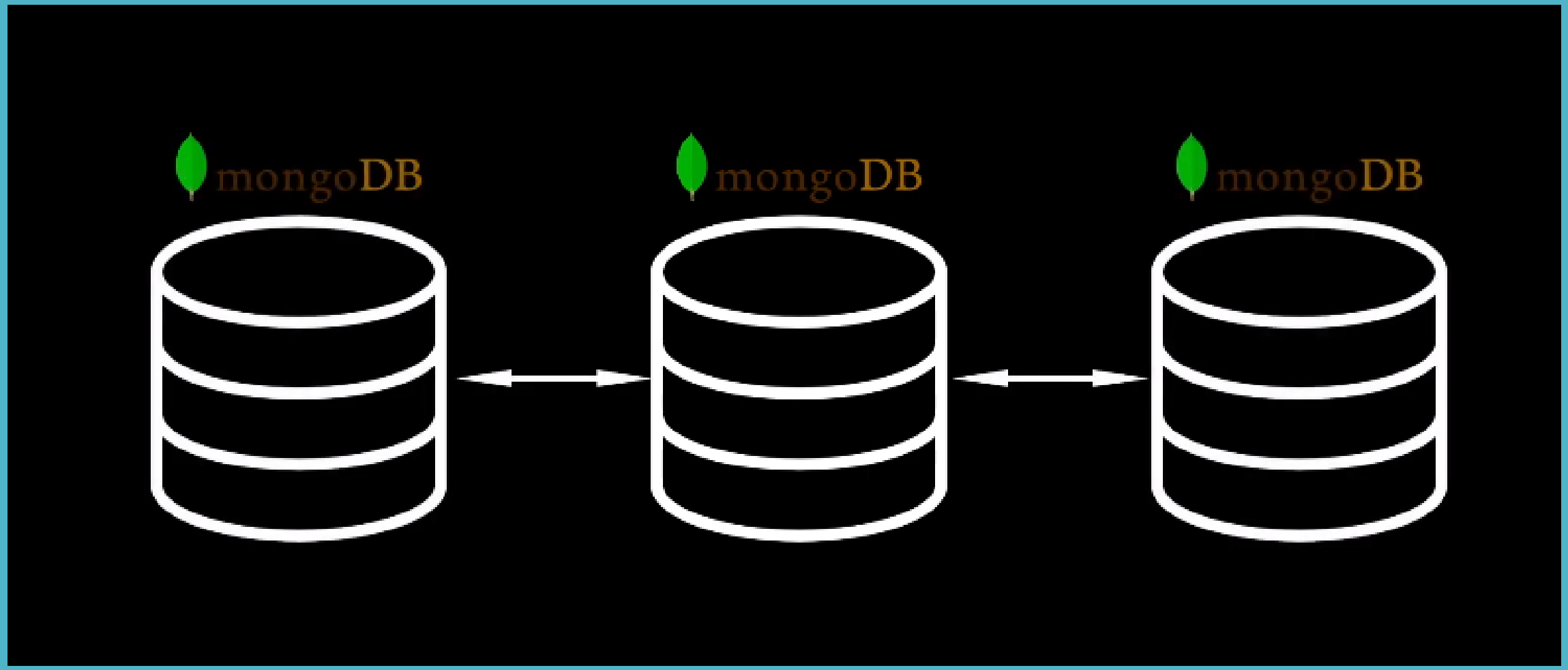
SQL (Relational) Databases



SELECT * FROM directors

NoSQL Databases




mongoDB
mongoDB
mongoDB

```
{
  "_id": ObjectId("628f2bdf4dc394dae55c7473"),
  "title": "Pulp Fiction",
  "director": "Quentin Tarantino",
  "year": 1994,
  "genres": ["crime", "drama"],
  "rating": 8.9,




},
```

Документ коллекции movies

Вложенные документы, позволяют избегать дополнительные связи между коллекциями.

Структура данных: таблицы vs. документы

Таблицы (Реляционные БД):

- Строки и столбцы: Реляционные БД организуют данные в виде строк и столбцов в таблицах. Каждая таблица имеет фиксированную схему — все строки должны соответствовать заранее определенной структуре.
- Связи между таблицами: Данные разбиваются по таблицам, и связи между ними устанавливаются через внешние ключи (foreign keys).

Документы (MongoDB):

- Документы: В MongoDB данные хранятся в виде JSON-подобных документов. Каждый документ может иметь произвольную структуру и может отличаться от других документов в коллекции.
- Вложенные данные: MongoDB поддерживает вложенные документы, что позволяет хранить данные, которые в реляционной базе потребовали бы нескольких таблиц, в одном документе.

```
{
  "id": 1,
  "name": "Иван Иванов",
  "age": 30,
  "address": {
    "city": "Москва",
    "street": "Улица Пушкина"
  }
}
```

Способы работы с данными (SQL vs. MongoDB Query Language)

SQL (Structured Query Language):

Используется для работы с реляционными базами данных.

Запросы требуют точного указания таблиц, полей и условий.

Стандартные операции: SELECT, INSERT, UPDATE, DELETE.

Пример SQL-запроса:

```sql
SELECT name, age FROM users WHERE city = 'Москва';
```

MongoDB Query Language (MQL):

MongoDB использует свой собственный язык запросов для работы с данными.

Запросы могут включать фильтры, сортировку, агрегации и вложенные структуры.

MQL поддерживает операции над документами, такие как поиск, вставка, обновление и удаление.

Пример MQL-запроса:

```
db.users.find({ "address.city": "Москва" }, { "name": 1, "age": 1 });
```

Гибкость схемы: фиксированные схемы vs. динамические схемы

Фиксированные схемы (Реляционные БД):

- В реляционных базах данных структура данных жестко фиксирована. Каждый столбец таблицы имеет определенный тип данных (например, INT, VARCHAR), и каждый ряд должен соответствовать этой структуре.
- Любые изменения схемы (например, добавление нового столбца) требуют модификации таблицы и могут быть сложными в больших системах.

Динамические схемы (MongoDB):

- В MongoDB данные хранятся в документах с динамической схемой. Это означает, что каждый документ может иметь свою уникальную структуру, и структура может меняться со временем.
- Разработчики могут легко добавлять новые поля в документы без необходимости изменять всю базу данных.

Пример динамической схемы MongoDB:

```
{
  "id": 1,
  "name": "Иван Иванов",
  "age": 30
}
```

```
{
  "id": 2,
  "name": "Мария Петрова",
  "email": "maria@example.com"
}
```

Преимущества MongoDB:

- Гибкость схемы: В MongoDB нет жестких ограничений на структуру данных, что позволяет легко изменять формат данных по мере роста и развития приложения. Документы могут содержать разнообразные поля и структуры, а их изменение не требует миграции данных.
- Высокая производительность при масштабировании: MongoDB поддерживает горизонтальное масштабирование через шардинг (разделение данных на разные серверы), что обеспечивает высокую производительность при работе с большими объемами данных.
- Простота работы с неструктурированными и полуструктурированными данными: MongoDB идеально подходит для хранения документов, JSON-подобных объектов и других типов данных, которые не вписываются в традиционную таблицу реляционной базы данных.
- Поддержка встроенных методов агрегации: MongoDB предоставляет мощные инструменты для обработки данных внутри базы, такие как агрегационные пайплайны, которые позволяют выполнять сложные операции без необходимости извлекать данные в приложение.
- Масштабируемость и отказоустойчивость: Благодаря встроенной поддержке репликации и механизмам отказоустойчивости MongoDB может легко адаптироваться к росту данных и обеспечивать высокую доступность системы.
- Поддержка больших данных и высоких нагрузок: MongoDB хорошо справляется с большими объемами данных, что делает её популярным выбором для приложений в областях анализа данных и Big Data.
- Поддержка встроенных транзакций: Начиная с версии 4.0, MongoDB поддерживает ACID-транзакции для работы с несколькими документами, что обеспечивает целостность данных в сложных операциях.

Недостатки MongoDB:

- Ограниченная поддержка сложных транзакций: Хотя MongoDB поддерживает транзакции, они всё ещё уступают по гибкости и возможностям транзакциям в реляционных базах данных, особенно при работе с высокими нагрузками или большим количеством документов.
- Высокий объем памяти при хранении данных: Из-за использования формата BSON (Binary JSON), MongoDB может потреблять больше места для хранения данных по сравнению с реляционными базами данных.
- Ограниченная поддержка сложных запросов и операций: В MongoDB нет аналогов некоторых сложных возможностей SQL (например, сложных джойнов или подзапросов), что может стать проблемой при необходимости выполнения таких операций.
- Повышенные требования к администрированию и настройке: Для эффективной работы с MongoDB требуется квалифицированное администрирование, особенно в больших системах с шардированием и репликацией. Неправильная конфигурация может привести к потерям данных или снижению производительности.
- Не всегда оптимальна для небольших приложений: Для небольших или сильно структурированных приложений использование MongoDB может быть избыточным, поскольку в таких случаях реляционные базы данных могут быть проще и более производительны.
- Проблемы с консистентностью данных при масштабировании: В некоторых случаях масштабирования MongoDB может сталкиваться с проблемами консистентности данных, особенно если настройки репликации и шардинга недостаточно оптимизированы.

Общий итог:

MongoDB лучше всего подходит для приложений, которым требуется гибкость в работе с неструктурированными данными, высокая производительность и масштабируемость.

Однако для приложений с жёсткими требованиями к транзакциям и сложной обработкой данных реляционные базы данных могут быть более подходящим выбором.

4. Области применения MongoDB.

MongoDB применяется в различных областях, где требуется гибкость данных, масштабируемость и высокая производительность. Вот несколько ключевых областей использования:

1. Веб-приложения и мобильные приложения

MongoDB часто используется для хранения данных в современных веб- и мобильных приложениях благодаря своей гибкой схеме данных. Он идеально подходит для приложений с динамическими требованиями к данным, где структура может меняться по мере развития продукта.

Примеры: социальные сети, платформы электронной коммерции, сервисы бронирования.

2. Big Data и аналитика

MongoDB используется в проектах, связанных с большими данными (Big Data), благодаря возможности обрабатывать большие объёмы данных и поддержке горизонтального масштабирования. С помощью встроенных агрегационных пайплайнов можно эффективно выполнять аналитику данных непосредственно в базе.

Примеры: системы аналитики для бизнес-отчётности, системы мониторинга и обработки больших объёмов данных.

3. IoT (Интернет вещей)

В приложениях IoT MongoDB используется для хранения и обработки огромных потоков данных, поступающих от устройств. Благодаря масштабируемости и поддержке неструктурированных данных, MongoDB подходит для хранения данных с датчиков и устройств в режиме реального времени.

Примеры: системы мониторинга производственного оборудования, управление умными домами и городами.

4. Реализация контент-менеджмент систем (CMS)

MongoDB идеально подходит для управления мультимедийным контентом, таким как изображения, видео, аудиофайлы и документы. Гибкая схема позволяет хранить различные типы контента в одной коллекции, упрощая управление мультимедийными ресурсами.

Примеры: платформы для публикации новостей, блоги, сайты с мультимедийными галереями.

5. Электронная коммерция

В eCommerce MongoDB используется для работы с каталогами товаров, заказами, пользователями и корзинами покупок. Возможность быстро изменять структуру данных и хранить разные типы информации о продуктах делает MongoDB удобной для использования в интернет-магазинах.

Примеры: платформы для онлайн-продаж, системы управления продуктами и клиентскими заказами.

6. Обработка данных реального времени

MongoDB широко применяется в системах, требующих обработки данных в реальном времени. Его гибкость и масштабируемость позволяют быстро обновлять и извлекать данные, что важно для приложений с большим количеством пользователей или систем с потоками данных в реальном времени.

Примеры: системы управления пользователями в реальном времени, приложения для чатов и мессенджеров, мониторинг и логирование в реальном времени.

7. Системы управления пользователями и профилями

MongoDB используется для хранения профилей пользователей, учетных записей и настроек в приложениях, где структура данных может варьироваться для разных пользователей. Это позволяет разработчикам гибко настраивать пользовательские профили.

Примеры: социальные сети, онлайн-игры, системы с персонализированным контентом.

8. Финансовые технологии (FinTech)

В финтех-приложениях MongoDB используется для управления транзакциями, пользовательскими данными и финансовыми операциями. Гибкость схемы и поддержка высокой нагрузки делают его подходящим выбором для работы с финансовыми данными.

Примеры: системы управления электронными кошельками, обработка платежей, управление инвестициями.

9. Игровая индустрия

MongoDB часто применяется в игровых приложениях для хранения данных об игроках, их прогрессе и игровых событиях. Она обеспечивает быстрое чтение и запись данных, что важно для игр с высокой нагрузкой и сложной логикой.

Примеры: онлайн-игры, платформы для мультиплеера.

10. Системы рекомендаций

MongoDB подходит для реализации систем рекомендаций, благодаря гибкости в хранении информации о предпочтениях пользователей, истории просмотров и взаимодействий. Такие системы могут быстро адаптироваться под изменяющиеся данные и предоставлять персонализированные рекомендации.

Примеры: рекомендации на сайтах eCommerce, видеоплатформы, музыкальные сервисы.

5. Установка и настройка MongoDB.

1. Требования к системе для установки MongoDB

MongoDB можно установить на различные операционные системы, включая Windows, Linux и macOS. Ниже приведены минимальные требования:

Операционная система:

- Linux (поддерживаются дистрибутивы Ubuntu, Debian, Red Hat, CentOS и другие)
- Windows Server 2016 и выше, Windows 10 (64-bit)
- macOS 10.14 и выше

Оперативная память (RAM): минимум 2 ГБ (рекомендуется 4 ГБ и выше для производительности)

Процессор: x86_64 архитектура

Дисковое пространство: минимум 10 ГБ для базы данных (в зависимости от объёма данных)

Дополнительные требования:

- MongoDB рекомендует использовать XFS или EXT4 на Linux, и NTFS на Windows.
- Важно использовать файловую систему с поддержкой журналирования.

6. Основные понятия MongoDB.

MongoDB — это **NoSQL** база данных, в которой данные хранятся в формате **документов**.

Она отличается от реляционных баз данных своей документно-ориентированной моделью.

Ниже представлены основные термины и понятия, которые необходимо знать для работы с MongoDB.

1. База данных (Database)

В MongoDB база данных — это контейнер для коллекций.

Каждая база данных содержит одну или несколько коллекций.

В одной инсталляции MongoDB может быть несколько баз данных.

Пример создания базы данных:

```
use myDatabase
```

Это создаст новую базу данных (или выберет существующую), если вставить в неё данные.

2. Коллекция (Collection)

Коллекция в MongoDB аналогична таблице в реляционных базах данных.

Она содержит множество документов, однако, в отличие от таблиц, документы в одной коллекции могут иметь разную структуру.

Коллекция не требует явного создания схемы и может хранить документы с различными наборами полей.

Пример создания коллекции:

```
db.createCollection("myCollection")
```

Коллекции автоматически создаются при первой вставке данных, если они не существуют.

3. Документ (Document)

Документ в MongoDB — это базовая единица данных, которая хранится в коллекции.

Документ — это структура данных в формате JSON-подобного объекта, который содержит пары ключ-значение.

Документы MongoDB используют BSON (Binary JSON) — бинарное представление JSON для хранения и передачи данных.

Документы в одной коллекции могут иметь разные наборы полей и значения, в отличие от строк в реляционной базе данных.

Пример документа:

```
{
    "_id": ObjectId("507f191e810c19729de860ea"),
    "name": "John Doe",
    "age": 29,
    "profession": "Engineer"
}
```

Поле _id уникально для каждого документа и автоматически создается MongoDB, если его не указать вручную.

4. Поле (Field)

Поле в документе — это ключ в паре ключ-значение.

Каждое поле имеет уникальное имя внутри одного документа и может содержать различные типы данных: строку, число, массив, объект и другие.

Пример полей в документе:

```
{
    "name": "Alice",
    "age": 25,
    "skills": ["JavaScript", "Python", "MongoDB"]
}
```

Поля могут быть вложенными (содержать объекты и массивы).

5. Типы данных

MongoDB поддерживает различные типы данных, в том числе:

String — строка текста.

Number — целые или дробные числа.

Boolean — логическое значение true или false.

Array — массив значений.

Object — вложенный объект.

Date — дата и время.

ObjectId — уникальный идентификатор документа.

Null — пустое значение.

```
{
    "name": "Company",
    "founded": 2005,
    "employees": ["John", "Anna", "Mike"],
    "isActive": true,
    "location": {
        "city": "New York",
        "country": "USA"
    },
    "createdAt": ISODate("2023-09-15T00:00:00Z"),
    "website": null
}
```

6. Идентификатор документа (_id)

Поле _id — это уникальный идентификатор документа в коллекции.

MongoDB автоматически создаёт значение _id, если оно не указано, используя тип данных ObjectId.

Поле _id должно быть уникальным в рамках коллекции и не может быть изменено после создания документа.

7. BSON (Binary JSON)

BSON — это двоичный формат хранения данных, который MongoDB использует для сериализации документов.

BSON поддерживает более широкий набор типов данных, чем стандартный JSON, включая даты и байтовые массивы.

BSON формат эффективен для передачи и хранения данных.

8. Модель данных в MongoDB

MongoDB поддерживает гибкую схему: структура документов может меняться от одного документа к другому в одной и той же коллекции.

Документно-ориентированная модель обеспечивает динамическое добавление новых полей и возможность вложения объектов, что увеличивает гибкость при работе с данными.

7. Создание базы данных и первой коллекции.

1. Основные команды для работы с базами данных и коллекциями в MongoDB

MongoDB позволяет создавать базы данных и коллекции динамически, без необходимости задавать строгие схемы, как в реляционных базах данных.

Для работы с MongoDB используется Mongo Shell или драйверы для различных языков программирования.

Подключение к MongoDB:

Для начала работы, подключитесь к MongoDB через консольный клиент Mongo Shell:

```
C:\Users\User>mongosh
Current Mongosh Log ID: 67077aa9cfe778dd8fc73bf7
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&
serverSelectionTimeoutMS=2000&appName=mongosh+2.3.1
Using MongoDB:          8.0.0
Using Mongosh:          2.3.1

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/


------
   The server generated these startup warnings when booting
   2024-10-10T07:50:17.166+05:00: Access control is not enabled for the d
atabase. Read and write access to data and configuration is unrestricted
------

test>
```

Создание базы данных

MongoDB автоматически создаёт базу данных при первой вставке данных в коллекцию. Тем не менее, базу данных можно выбрать или создать явно с помощью команды use.

Команда для выбора или создания базы данных:

```
use myDatabase
```

Эта команда переключит вас на базу данных myDatabase. Если она не существует, MongoDB создаст её при вставке первого документа.

Проверка текущей базы данных:

```
db
```

Показывает, с какой базой данных вы работаете в данный момент.

Создание первой коллекции

Коллекции в MongoDB также создаются автоматически при вставке первого документа, но их можно создать вручную с помощью команды createCollection.

Команда для создания коллекции:

```
db.createCollection("myCollection")
```

Эта команда создаст коллекцию с именем myCollection в выбранной базе данных.

Проверка списка коллекций:

```
show collections
```

Показывает список коллекций, которые уже существуют в текущей базе данных.

Вставка данных в коллекцию

MongoDB работает с данными в формате JSON-подобных документов, и документы можно вставлять в коллекцию с помощью команд insertOne (для одного документа) или insertMany (для нескольких документов).

Вставка одного документа:

```
db.myCollection.insertOne({
    name: "John Doe",
    age: 30,
    profession: "Software Developer"
})
```

Эта команда вставляет один документ в коллекцию myCollection. Если коллекции не существует, она создастся автоматически.

Вставка нескольких документов:

```
db.myCollection.insertMany([
    { name: "Alice", age: 25, profession: "Data Scientist" },
    { name: "Bob", age: 28, profession: "DevOps Engineer" }
])
```

Вставляются сразу несколько документов в коллекцию.

Просмотр всех документов в коллекции:

```
db.myCollection.find()
```

Эта команда возвращает все документы, которые содержатся в коллекции.

Поиск с фильтром:

```
db.myCollection.find({ age: 30 })
```

Эта команда вернёт все документы, в которых поле age равно 30.

Практика:

https://labex.io/ru/labs/mongodb-your-first-mongodb-lab-420660