MongoDB: Основы работы с NoSQL базами данных

□ План семинара

Блок 1: Установка и знакомство с MongoDB

- Подключение к MongoDB
- Обзор интерфейса MongoDB Compass / NoSQL Booster
- Создание базы данных и коллекции
- Импорт тестовых данных

Блок 2 : CRUD операции

- **Create** - вставка документов
- **Read** - поиск и фильтрация
- **Update** - обновление документов
- **Delete** - удаление документов

Блок 3: Индексы и агрегации

- Создание индексов
- Анализ производительности
- Простые агрегации

□ Датасет: Фильмы MongoDB Sample

Источник: MongoDB Sample Mflix Dataset
Ссылка: https://raw.githubusercontent.com/neelabalan/mongodb-sample-dataset/main/sample_mflix/movies.json

Структура документа фильма:

```
{
  "_id": ObjectId("573a1390f29313caabcd4135"),
  "title": "Inception",
  "year": 2010,
  "genres": ["Sci-Fi", "Thriller"],
  "runtime": 148,
  "cast": ["Leonardo DiCaprio", "Marion Cotillard"],
  "directors": ["Christopher Nolan"],
  "imdb": {
    "rating": 8.8,
    "votes": 2000000
  },
  "tomatoes": {
    "viewer": {
      "rating": 4.2,
      "numReviews": 150000
    }
  },
  "plot": "Dom Cobb is a skilled thief...",
  "countries": ["USA", "UK"],
  "released": ISODate("2010-07-16T00:00:00.000Z")
}
```

□ БЛОК 1: Установка и настройка

Шаг 1: Подключение к MongoDB

Вариант A: MongoDB в Docker (рекомендуется)

```
# Запуск MongoDB в контейнере
docker run -d \
  --name mongodb \
  -p 27017:27017 \
  -e MONGO_INITDB_ROOT_USERNAME=admin \
  -e MONGO_INITDB_ROOT_PASSWORD=secret \
  mongo:latest

# Проверка, что контейнер запущен
docker ps

# Подключение через mongo shell
docker exec -it mongodb mongosh -u admin -p secret
```

Строка подключения: `mongodb://admin:secret@localhost:27017`

Вариант B: MongoDB Compass

1. Открыть MongoDB Compass
2. Подключиться к серверу: `mongodb://admin:secret@localhost:27017`
3. Создать новую базу данных: `movies_db`

Вариант C: NoSQL Booster / MongoDB Compose

1. Открыть приложение
2. Создать новое подключение: `mongodb://admin:secret@localhost:27017`
3. Подключиться к серверу

Шаг 2: Создание базы данных

```
// В MongoDB Shell или встроенном редакторе
use movies_db
```

Шаг 3: Импорт данных

Через Compass:

1. Collections → Create Collection → `movies`
2. Import Data → Select File → movies.json
3. File Type: JSON

Через командную строку:

```
mongoimport --db movies_db --collection movies --file movies.json --jsonArray
```

Проверка импорта:

```
db.movies.countDocuments()

db.movies.findOne()
```

□ БЛОК 2: CRUD операции

2.1 CREATE - Вставка документов

Вставка одного документа

```
db.movies.insertOne({
  "title": "Мой любимый фильм",
  "year": 2024,
  "genres": ["Драма", "Комедия"],
  "directors": ["Студент Первый"],
  "cast": ["Актер А", "Актер Б"],
  "runtime": 120,
  "imdb": {
    "rating": 9.5,
    "votes": 1000
  },
  "plot": "История о том, как студент изучал MongoDB"
})
```

Вставка нескольких документов

```
db.movies.insertMany([
  {
    "title": "MongoDB: Фильм",
    "year": 2024,
    "genres": ["Документальный"],
    "runtime": 90,
    "imdb": {"rating": 8.0, "votes": 500}
  },
  {
    "title": "NoSQL Приключения",
    "year": 2024,
    "genres": ["Фантастика"],
    "runtime": 105,
    "imdb": {"rating": 7.5, "votes": 300}
  }
])
```

Практическое задание 1

Создайте документ для своего любимого фильма со всеми полями

2.2 READ - Чтение и поиск

Базовые запросы

Найти все документы

```
db.movies.find()
```

Найти один документ

```
db.movies.findOne()
```

Поиск с условием

```
// Фильмы 2010 года
db.movies.find({"year": 2010})

// Фильмы с высоким рейтингом
db.movies.find({"imdb.rating": {"$gte": 8.0}})
```

Операторы сравнения

```
// Равенство
db.movies.find({"year": 2010})

// Больше чем
db.movies.find({"year": {"$gt": 2000}})

// Больше или равно
db.movies.find({"imdb.rating": {"$gte": 8.0}})

// Меньше чем
db.movies.find({"runtime": {"$lt": 90}})

// Не равно
db.movies.find({"year": {"$ne": 2010}})

// В списке значений
db.movies.find({"year": {"$in": [2010, 2015, 2020]}})

// Не в списке
db.movies.find({"year": {"$nin": [1990, 1995]}})
```

Работа с массивами

```
// Фильмы с жанром "Drama"
db.movies.find({"genres": "Drama"})

// Фильмы с несколькими жанрами
db.movies.find({"genres": {"$all": ["Drama", "Romance"]}})

// Фильмы с Leonardo DiCaprio
db.movies.find({"cast": "Leonardo DiCaprio"})
```

Работа со строками (regex)

```
// Фильмы, в названии которых есть "Love"
db.movies.find({"title": {"$regex": "Love", "$options": "i"}})

// Фильмы, начинающиеся с "The"
db.movies.find({"title": {"$regex": "^The"}})
```

Логические операторы

```
// И (AND)
db.movies.find({
  "$and": [
    {"year": {"$gte": 2000}},
    {"imdb.rating": {"$gte": 7.0}}
  ]
})

// ИЛИ (OR)
db.movies.find({
  "$or": [
    {"genres": "Comedy"},
    {"genres": "Romance"}
  ]
})
```

Проекция (выбор полей)

```
// Показать только название и год
db.movies.find({}, {"title": 1, "year": 1, "_id": 0})

// Исключить определенные поля
db.movies.find({}, {"plot": 0, "fullplot": 0})
```

Сортировка и ограничения

```
// Сортировка по году (убывание)
db.movies.find().sort({"year": -1})

// Ограничить результат 5 документами
db.movies.find().limit(5)

// Пропустить первые 10 документов
db.movies.find().skip(10)

// Комбинация
db.movies.find({"imdb.rating": {"$gte": 8.0}})
         .sort({"imdb.rating": -1})
         .limit(10)
```

Практические задания

1. **Найдите все фильмы с рейтингом IMDB выше 8.5**
2. **Найдите комедии, выпущенные после 2000 года**
3. **Найдите фильмы с Tom Hanks в актерском составе**
4. **Найдите топ-10 фильмов по рейтингу IMDB**
5. **Найдите фильмы продолжительностью от 90 до 120 минут**

2.3 UPDATE - Обновление документов

Результаты операций обновления

MongoDB всегда возвращает информацию о результате:

```
// Пример результата updateOne()
{
  "acknowledged": true,
  "matchedCount": 1,    // Найдено документов
  "modifiedCount": 1,   // Изменено документов
  "upsertedId": null    // ID созданного документа (если upsert)
}
```

Важно всегда проверять результаты операций!

Обновление одного документа

```
// Сначала посмотрим на текущий рейтинг
db.movies.findOne({"title": "Inception"}, {"title": 1, "imdb.rating": 1})

// Обновить рейтинг конкретного фильма
db.movies.update(
  {"title": "Inception"},
  {"$set": {"imdb.rating": 9.0}}
)

// Проверим результат
db.movies.findOne({"title": "Inception"}, {"title": 1, "imdb.rating": 1})
```

Модификаторы обновления

$set - установить значение

```
// До обновления
db.movies.findOne({"title": "Titanic"}, {"title": 1, "imdb.rating": 1, "runtime": 1})

// Обновление
db.movies.update(
  {"title": "Titanic"},
  {"$set": {
    "imdb.rating": 7.8,
    "runtime": 195,
    "updated": new Date()
  }}
)

// После обновления - проверяем результат
db.movies.findOne({"title": "Titanic"}, {"title": 1, "imdb.rating": 1, "runtime": 1, "updated": 1})
```

$inc - увеличить значение

```
// До изменения
db.movies.findOne({"title": "The Dark Knight"}, {"title": 1, "imdb.votes": 1})

// Увеличить количество голосов на 100
db.movies.update(
  {"title": "The Dark Knight"},
  {"$inc": {"imdb.votes": 100}}
)

// Проверяем результат
db.movies.findOne({"title": "The Dark Knight"}, {"title": 1, "imdb.votes": 1})
```

$push - добавить элемент в массив

```
// Сначала смотрим текущие жанры
db.movies.findOne({"title": "Interstellar"}, {"title": 1, "genres": 1})

// Добавить жанр
db.movies.update(
  {"title": "Interstellar"},
  {"$push": {"genres": "Epic"}}
)

// Проверяем обновленные жанры
db.movies.findOne({"title": "Interstellar"}, {"title": 1, "genres": 1})

// Добавить несколько элементов в cast
db.movies.update(
  {"title": "Avatar"},
  {"$push": {"cast": {"$each": ["Новый Актер 1", "Новый Актер 2"]}}}
)

// Проверяем обновленный cast
db.movies.findOne({"title": "Avatar"}, {"title": 1, "cast": 1})
```

$pull - удалить элемент из массива

```
// Найдем фильм с жанром Horror
db.movies.findOne({"genres": "Horror"}, {"title": 1, "genres": 1})

// Удалить жанр Horror из найденного фильма (замените на реальное название)
db.movies.update(
  {"title": "НАЗВАНИЕ_ФИЛЬМА"},
  {"$pull": {"genres": "Horror"}}
)

// Проверяем результат
db.movies.findOne({"title": "НАЗВАНИЕ_ФИЛЬМА"}, {"title": 1, "genres": 1})
```

$addToSet - добавить уникальный элемент в массив

```
// До добавления
db.movies.findOne({"genres": "Action"}, {"title": 1, "genres": 1})

// Добавляем жанр (не дублируется, если уже есть)
db.movies.update(
  {"genres": "Action"},
  {"$addToSet": {"genres": "Thriller"}}
)

// Проверяем результат
db.movies.findOne({"title": "НАЗВАНИЕ_НАЙДЕННОГО_ФИЛЬМА"}, {"title": 1, "genres": 1})
```

Обновление нескольких документов

```
// Сначала посмотрим, сколько фильмов 1990 года без поля "decade"
db.movies.countDocuments({"year": 1990, "decade": {"$exists": false}})

// Обновить все фильмы 1990 года (используем параметр multi: true)
db.movies.update(
  {"year": 1990},
  {"$set": {"decade": "90s"}},
  {"multi": true}
)

// Проверяем результат - сколько документов было обновлено
db.movies.countDocuments({"year": 1990, "decade": "90s"})

// Посмотрим на один из обновленных документов
db.movies.findOne({"year": 1990}, {"title": 1, "year": 1, "decade": 1})
```

Upsert - вставить, если не существует

```
// Сначала проверим, существует ли фильм
db.movies.findOne({"title": "Новый фильм 2024"})

// Upsert операция
db.movies.update(
  {"title": "Новый фильм 2024"},
  {"$set": {
    "year": 2024,
    "genres": ["Sci-Fi"],
    "imdb": {"rating": 7.0}
  }},
  {"upsert": true}  // Создаст документ, если не найден
)

// Проверяем, что документ был создан
db.movies.findOne({"title": "Новый фильм 2024"})

// Попробуем upsert еще раз (должен только обновить существующий)
db.movies.update(
  {"title": "Новый фильм 2024"},
  {"$set": {"imdb.rating": 7.5}},
  {"upsert": true}
)

// Проверяем обновленный рейтинг
db.movies.findOne({"title": "Новый фильм 2024"}, {"title": 1, "imdb.rating": 1})
```

Практические задания

1. **Обновите рейтинг своего любимого фильма**

   ```
   // Найдите фильм
   db.movies.findOne({"title": "НАЗВАНИЕ"}, {"title": 1, "imdb.rating": 1})

   // Обновите рейтинг
   db.movies.update({"title": "НАЗВАНИЕ"}, {"$set": {"imdb.rating": 9.5}})

   // Проверьте результат
   db.movies.findOne({"title": "НАЗВАНИЕ"}, {"title": 1, "imdb.rating": 1})
   ```

2. **Добавьте жанр "Classic" всем фильмам до 1980 года**

   ```
   // Сначала посмотрим количество
   db.movies.countDocuments({"year": {"$lt": 1980}})

   // Обновим (не забываем multi: true для всех документов)
   db.movies.update(
     {"year": {"$lt": 1980}},
     {"$addToSet": {"genres": "Classic"}},
     {"multi": true}
   )

   // Проверим результат
   db.movies.findOne({"year": {"$lt": 1980}}, {"title": 1, "year": 1, "genres": 1})
   ```

3. **Увеличьте количество голосов на 1000 для фильмов с рейтингом выше 9.0**

```
// Найдем фильмы с высоким рейтингом
db.movies.find({"imdb.rating": {"$gt": 9.0}}, {"title": 1, "imdb.rating": 1, "imdb.votes": 1})

// Обновим голоса
db.movies.update(
  {"imdb.rating": {"$gt": 9.0}},
  {"$inc": {"imdb.votes": 1000}},
  {"multi": true}
)

// Проверим результат
db.movies.find({"imdb.rating": {"$gt": 9.0}}, {"title": 1, "imdb.rating": 1, "imdb.votes": 1})
```

4. **Используйте upsert для создания нового фильма**

```
// Проверим, что фильма нет
db.movies.findOne({"title": "Мой фильм 2024"})

// Создадим через upsert
db.movies.update(
  {"title": "Мой фильм 2024"},
  {"$set": {"year": 2024, "genres": ["Drama"], "imdb": {"rating": 8.5}}},
  {"upsert": true}
)

// Убедимся, что создался
db.movies.findOne({"title": "Мой фильм 2024"})
```

2.4 DELETE - Удаление документов (10 мин)

Удаление одного документа

```
// Удалить фильм по названию
db.movies.deleteOne({"title": "Bad Movie"})
```

Удаление нескольких документов

```
// Удалить все фильмы с рейтингом ниже 2.0
db.movies.deleteMany({"imdb.rating": {"$lt": 2.0}})

// Удалить все фильмы без рейтинга
db.movies.deleteMany({"imdb.rating": {"$exists": false}})
```

Удаление всех документов в коллекции

```
// ВНИМАНИЕ: Удалит ВСЕ документы!
db.movies.deleteMany({})

// Альтернатива - удаление коллекции
db.movies.drop()
```

Практическое задание (5 мин)

Удалите фильмы, которые вы создали для тестирования

БЛОК 3: Индексы и агрегации (15 мин)

3.1 Работа с индексами (8 мин)

Создание индексов

```
// Простой индекс по году выпуска
db.movies.createIndex({"year": 1})

// Индекс по рейтингу (убывание)
db.movies.createIndex({"imdb.rating": -1})

// Составной индекс
db.movies.createIndex({"year": 1, "imdb.rating": -1})

// Индекс по тексту (для поиска в названиях)
db.movies.createIndex({"title": "text", "plot": "text"})
```

Просмотр индексов

```
// Показать все индексы коллекции
db.movies.getIndexes()
```

Анализ производительности с explain()

```
// Запрос БЕЗ индекса - смотрим производительность
db.movies.find({"year": 2010}).explain("executionStats")

// Создаем индекс по году
db.movies.createIndex({"year": 1})

// Тот же запрос С индексом
db.movies.find({"year": 2010}).explain("executionStats")
```

Как читать результаты explain()

Основные поля для анализа:

Поле	Значение
`stage`	Этап выполнения (`COLLSCAN` - сканирование коллекции, `IXSCAN` - использование индекса)
`filter`	Условия фильтрации, к которым применяются индексы
`docsExamined`	Количество просмотренных документов
`nReturned`	Количество документов, возвращённых запросом
`totalKeysExamined`	Сколько ключей индекса просмотрено (>0 если индекс используется)
`executionTimeMillis`	Время выполнения запроса в миллисекундах
`winningPlan`	План, который MongoDB выбрала для выполнения
`rejectedPlans`	Планы, которые MongoDB отклонила

Примеры анализа:

```
// ❌ Плохая производительность (без индекса):
{
  "stage": "COLLSCAN",           // ❌ Сканирование всей коллекции
  "docsExamined": 23539,         // ❌ Просмотрено много документов
  "nReturned": 279,              // ✅ Вернули только нужные
  "executionTimeMillis": 25      // ❌ Медленно
}

// ✅ Хорошая производительность (с индексом):
{
  "stage": "IXSCAN",             // ✅ Использование индекса
  "docsExamined": 279,           // ✅ Просмотрено минимум документов
  "nReturned": 279,              // ✅ Идеальное соотношение 1:1
  "totalKeysExamined": 279,      // ✅ Использовался индекс
  "executionTimeMillis": 2       // ✅ Очень быстро
}
```

Практические примеры анализа

```
// 1. Запрос по году (должен использовать индекс)
db.movies.find({"year": 2010}).explain("executionStats")

// 2. Запрос по рейтингу (должен использовать индекс)
db.movies.find({"imdb.rating": {"$gte": 8.0}}).explain("executionStats")

// 3. Составной запрос (может использовать составной индекс)
db.movies.find({"year": 2010, "imdb.rating": {"$gte": 7.0}}).explain("executionStats")

// 4. Запрос без индекса (должен показать COLLSCAN)
db.movies.find({"runtime": {"$gt": 120}}).explain("executionStats")
```

Полезные индексы для нашего датасета:

```
// Основные индексы для улучшения производительности
db.movies.createIndex({"year": 1})
db.movies.createIndex({"imdb.rating": -1})
db.movies.createIndex({"genres": 1})
db.movies.createIndex({"year": 1, "imdb.rating": -1})
```

3.2 Простые агрегации (7 мин)

Подсчет фильмов по жанрам

```
// Простая группировка: сколько фильмов каждого жанра
db.movies.aggregate([
  {"$unwind": "$genres"},           // Разворачиваем массив жанров
  {"$group": {                      // Группируем по жанрам
    "_id": "$genres",
    "количество": {"$sum": 1}       // Считаем количество
  }},
  {"$sort": {"количество": -1}},    // Сортируем по убыванию
  {"$limit": 5}                     // Показываем топ-5
])
```

Средний рейтинг по годам

```
// Средний рейтинг IMDB по годам
db.movies.aggregate([
  {"$group": {
    "_id": "$year",                    // Группируем по годам
    "средний_рейтинг": {"$avg": "$imdb.rating"},
    "количество_фильмов": {"$sum": 1}
  }},
  {"$sort": {"_id": -1}},            // От новых к старым годам
  {"$limit": 10}
])
```

Самые популярные актеры

```
// Подсчет, сколько фильмов у каждого актера
db.movies.aggregate([
  {"$unwind": "$cast"},              // Разворачиваем массив актеров
  {"$group": {
    "_id": "$cast",                  // Группируем по актерам
    "фильмов": {"$sum": 1}
  }},
  {"$sort": {"фильмов": -1}},        // По убыванию количества
  {"$limit": 10}
])
```

Статистика по продолжительности

```
// Простая статистика по длительности фильмов
db.movies.aggregate([
  {"$group": {
    "_id": null,                     // Одна группа для всех
    "средняя_длительность": {"$avg": "$runtime"},
    "максимальная": {"$max": "$runtime"},
    "минимальная": {"$min": "$runtime"},
    "всего_фильмов": {"$sum": 1}
  }}
])
```

Домашнее задание

Структура данных

Ваша база данных `shop` будет состоять из двух основных коллекций, каждая из которых хранит документы в формате BSON (Binary JSON).

Коллекция `products` — каталог товаров:

```
{
  name: "iPhone 15",
  price: 80000,
  quantity: 10,
  description: "Новый iPhone с чипом A17 Pro"
}
```

Каждый товар представлен отдельным документом со всей необходимой информацией. Поле `name` содержит название товара, `price` — цену в рублях, `quantity` — количество на складе, а `description` — подробное описание. MongoDB автоматически добавит уникальное поле `_id` для каждого документа.

Коллекция `orders` — история заказов клиентов:

```
{
  customer_email: "ivan@mail.ru",
  product_name: "iPhone 15",
  quantity: 2,
  total_price: 160000,
  order_date: new Date("2024-12-15")
}
```

Каждый заказ связывает клиента с товаром через email и название продукта. Поле `quantity` показывает количество единиц товара в заказе, `total_price` — общую стоимость позиции, а `order_date` — дату оформления заказа. Такая структура позволяет легко анализировать покупательское поведение и продажи во времени.

Задания

1. **Создайте базу данных и добавьте данные.**
2. **Выполните запросы:**
    - Найти все товары дороже 50000 рублей
    - Обновить цену товара по названию
    - Удалить товар с количеством 0
3. **Напишите агрегации:**
    - Подсчитайте среднюю цену товаров в каждой категории
    - Найдите топ-3 самых дорогих товара с их категориями
    - Посчитайте общую стоимость всех товаров на складе (цена × количество)
    - Найдите категории, где средняя цена товаров больше 30000 рублей
    - Посчитать сумму всех заказов
    - Посчитатс сумму всех заказов по customer_email и отсортировать по убыванию общей суммы
    - Найти топ-3 самых продавамых товаров