Базы данных
Лекция 7

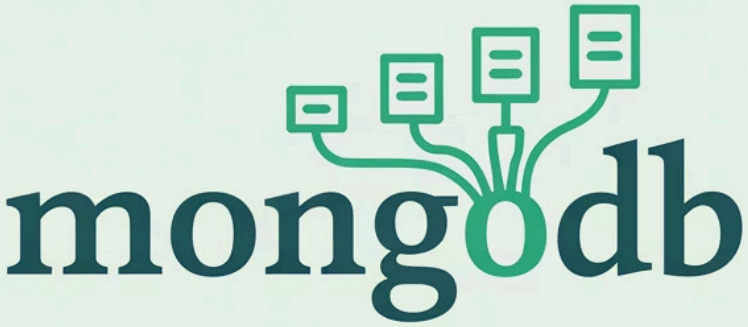
MongoDB

Мгер Аршакян

План лекции

1. Введение в MongoDB
2. Модель данных
3. Индексирование и производительность
4. Агрегационный фреймворк
5. Распределённые возможности

Введение в MongoDB

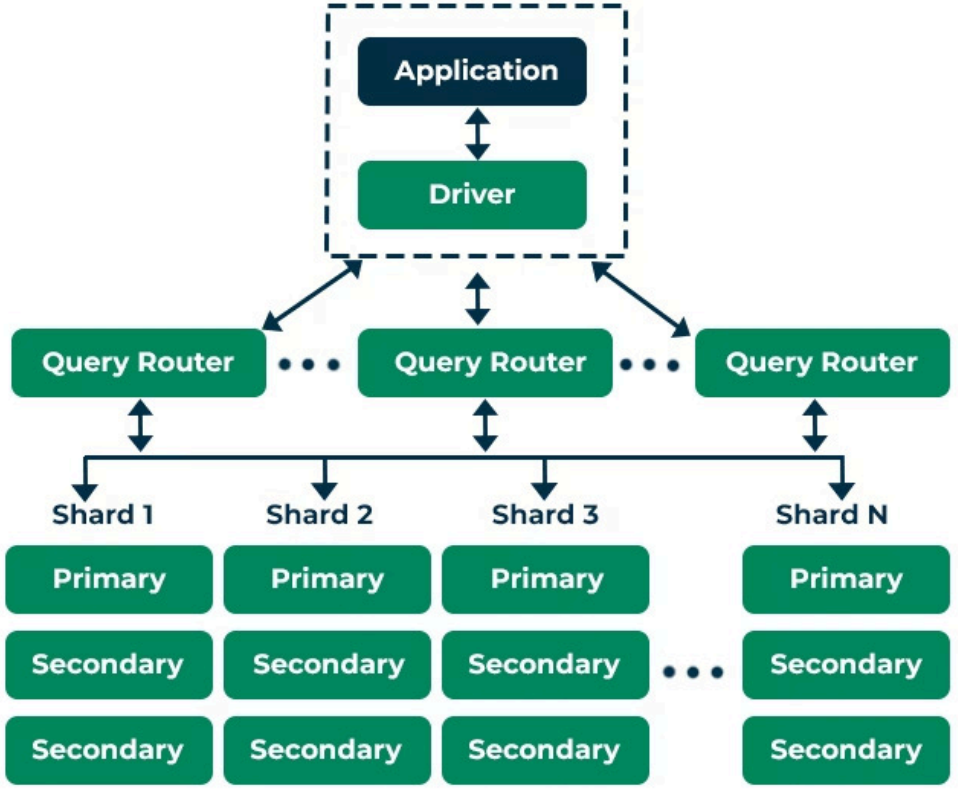


Что такое MongoDB?

1. Главная идея — документы (BSON -бинарный JSON)
2. NoSQL — принципы BASE вместо ACID (но «A» ≠ отсутствие транзакций!)
3. Отсутствие схемы
4. Мощный движок для агрегации

Архитектура MongoDB

client □ mongos □ shards



Модель данных

Соответствие SQL и MongoDB

- **Database** → **Database**
  В обоих случаях это логический контейнер для данных.
- **Table** → **Collection**
  Набор строк в SQL соответствует набору документов в MongoDB.
- **Row (Record)** → **Document**
  Один экземпляр данных: JSON-/BSON-документ вместо строкового набора ячеек.
- **Column** → **Field**
  Ключ–значение внутри документа.
- **Primary Key** → `_id` (ObjectId или свой тип)
  Уникально идентифицирует документ; генерируется автоматически, если не задать.

Формат BSON

- Двоичный JSON с расширенными типами данных
- Порядок полей сохранён, поддержка null vs missing
- ObjectId = 12 байт (timestamp, machine, PID, counter)
- Ограничение 16 MB на документ
- Сжатие на диске: zstd

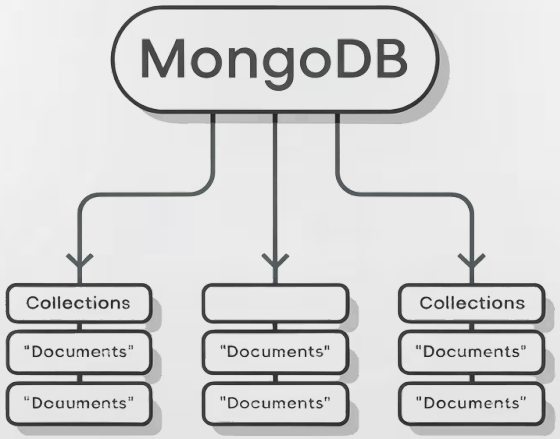
ObjectId Format

The **ObjectId** is composed of 12 bytes, broken down as follows:

- The first **4** bytes represent the Unix Timestamp of the document.
- The next **3** bytes are the machine **ID** on which the MongoDB server is running.
- The next **2** bytes are of the process ID.
- The last Field is 3 bytes used for incrementing the objectid.

Типичный документ в MongoDB

```
{
    "_id": ObjectId("655f4b6e9d1e8a6a8c3b6e21"),          // ObjectId  (уникальный ключ)
    "name": "Ирина Ковалёва",                             // String
    "active": true,                                       // Boolean
    "balance": NumberDecimal("12345.67"),                 // Decimal128 (финансы без потери точности)
    "rating": 4.8,                                        // Double
    "visits": NumberInt(54),                              // 32-битовое целое
    "lastLogin": ISODate("2025-05-18T21:45:11Z"),         // Date
    "createdTS": Timestamp(1684493300, 1),                // BSON Timestamp
    "avatar": BinData(0, "iVBORw0KGgoAAAANSUhEUg…"),      // Binary Data (здесь — начало файла PNG, Base64-код)
    "interests": ["фото", "путешествия", "йога"],         // Массив строк
    "scores": [                                           // Массив вложенных документов
        { "date": ISODate("2025-04-01"), "value": NumberInt(10) },
        { "date": ISODate("2025-05-01"), "value": NumberInt(15) }
    ],
    "deletedAt": null                                     // Null (помечаем «мягкое» удаление)
}
```



Коллекции и БД

- Коллекция — упорядоченный набор документов без фикс-схемы
- Validation rules (JSON Schema) opt-in
- Индекс `_id` создаётся автоматически

Пример

```
// Создание и использование базы данных
use university

// Создание коллекции
db.createCollection("students")

// Проверка существующих коллекций
show collections
```

Встраивание vs Ссылки

```
// Встраивание (Embedding)
db.orders.insertOne({
  customer: {
    name: "Иван Петров",
    email: "ivan@example.com"
  },
  items: [
    { product: "Телефон", price: 15000 },
    { product: "Чехол", price: 1000 }
  ]
})

// Ссылки (References)
db.customers.insertOne({
  _id: ObjectId("60a1e2d3f889cb45e37b6a2c"),
  name: "Иван Петров",
  email: "ivan@example.com"
})

db.orders.insertOne({
  customer_id: ObjectId("60a1e2d3f889cb45e37b6a2c"),
  items: [/* ... */]
})
```

1. Встраивание предпочтительнее, когда:
   1. часто запрашиваются вместе
   2. обновляются атомарно
   3. связь "один-ко-многим"

2. Ссылки предпочтительны, когда:
   1. большие иерархии данных
   2. много изменяемых элементов
   3. данные должны быть доступны самостоятельно

Индексирование и производительность

Типы индексов

Тип индекса	Ключевые свойства	Когда использовать	Синтаксис создания (`db.collection.createIndex()`)
Однополевой (single-field)	Сортирует/фильтрует по одному полю; направление `1` (ASC) или `-1` (DESC).	Любой частый поиск или сортировка по одному столбцу.	`{ age: 1 }`
Составной (compound)	Один индекс на несколько полей (порядок *важен*).	Запросы, где фильтр начинается с первых ключей индекса.	`{ status: 1, createdAt: -1 }`
Multikey	Создаётся на поле-массив; индексирует каждое значение массива.	Фильтры по массивам (`tags: "mongo"`).	`{ tags: 1 }` *(если* `tags` *— массив)*
Уникальный (unique)	Гарантирует отсутствия дубликатов.	E-mail, логины, любые «однозначные» поля.	`{ email: 1 }`, `{ unique: true }`
Sparse	Индексируются **только** документы, где поле существует.	Поле необязательно у всех; держим индекс компактным.	`{ promoCode: 1 }`, `{ sparse: true }`
TTL (Time-To-Live)	Документы автоматически удаляются через *N* секунд/по дате.	Сессии, временные логи, кэш.	`{ createdAt: 1 }`, `{ expireAfterSeconds: 3600 }`
Text	Полнотекстовый поиск, поддержка стемминга и веса полей.	Поиск по названиям, описаниям, блогам.	`{ title: "text", body: "text" }`
Geospatial 2d	Плоская поверхность (старый формат координат `[x, y]`).	Устаревшие датасеты, простые «на карте».	`{ loc: "2d" }`

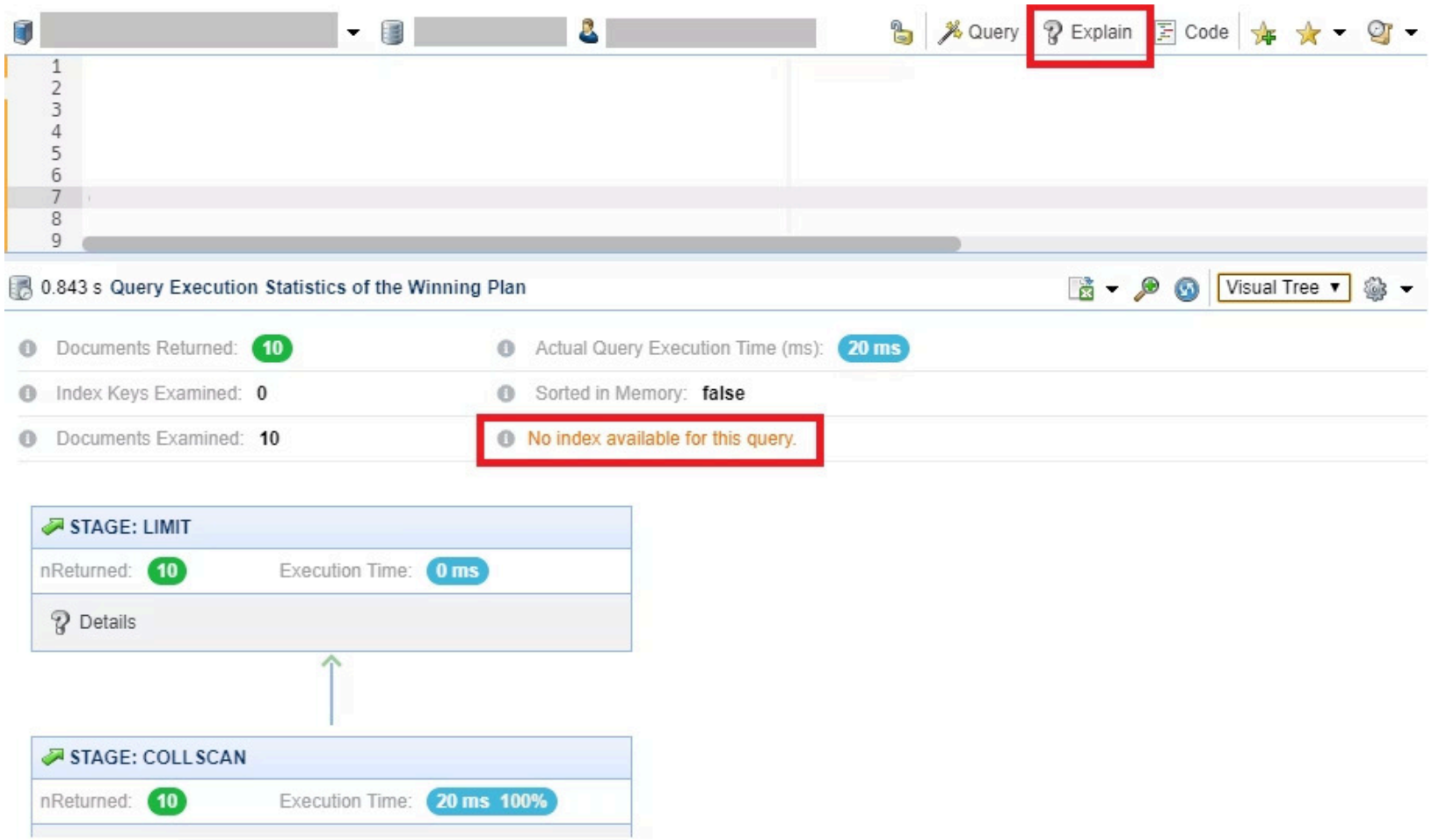
Анализ производительности запросов

```
// Базовый анализ плана запроса
db.users.find({ age: { $gt: 21 } }).explain()

// Подробный анализ с метриками производительности
db.users.find({ age: { $gt: 21 } }).explain("executionStats")

// Анализ запроса с сортировкой и лимитом
db.users.find({ status: "active" })
  .sort({ lastLogin: -1 })
  .limit(10)
  .explain("allPlansExecution")
```

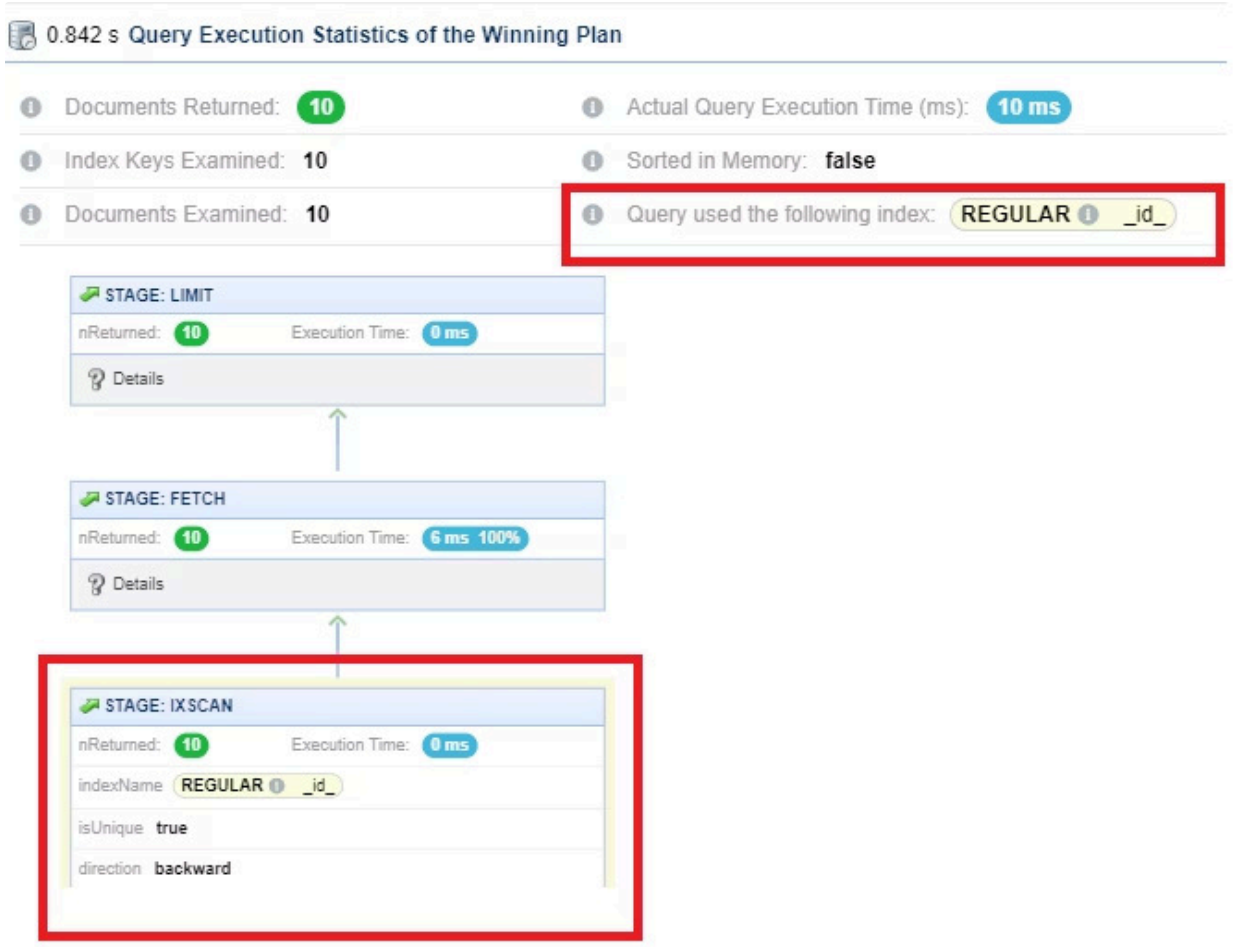
Ключевые метрики explain()

- winningPlan - выбранный план исполнения запроса
- COLLSCAN vs IXSCAN - сканирование коллекции или использование индекса
- nReturned - количество возвращенных документов
- totalKeysExamined - количество просмотренных ключей индекса
- totalDocsExamined - количество просмотренных документов
- executionTimeMillis - время выполнения Рассказать о признаках неоптимального запроса: totalDocsExamined >> nReturned или COLLSCAN для больших коллекций.

Без индекса


Query
Explain
Code
0.843 s Query Execution Statistics of the Winning Plan
Visual Tree
Documents Returned: 10
Actual Query Execution Time (ms): 20 ms
Index Keys Examined: 0
Sorted in Memory: false
Documents Examined: 10
No index available for this query.
STAGE: LIMIT
nReturned: 10
Execution Time: 0 ms
Details
STAGE: COLLSCAN
nReturned: 10
Execution Time: 20 ms 100%

С индекса


0.842 s Query Execution Statistics of the Winning Plan
Documents Returned: 10
Index Keys Examined: 10
Documents Examined: 10
Actual Query Execution Time (ms): 10 ms
Sorted in Memory: false
Query used the following index: REGULAR _id_
STAGE: LIMIT
nReturned: 10
Execution Time: 0 ms
Details
STAGE: FETCH
nReturned: 10
Execution Time: 6 ms 100%
Details
STAGE: IXSCAN
nReturned: 10
Execution Time: 0 ms
indexName REGULAR _id_
isUnique true
direction backward

Агрегационный фреймворк

Pipeline-концепция

- «Unix-like» конвейер: каждый stage обрабатывает поток MongoDB
- Позволяет трансформировать, фильтровать, группировать
- `$facet` для параллельных под-pipeline

Ключевые стадии

- `$match` — фильтр (использует индексы) MongoDB
- `$project` — переименование/выбор полей
- `$group` — агрегаторы `$sum`, `$avg`, `$push`
- `$sort` + `$limit` — пагинация без `skip` (cursor-based)
- etc.

Простая аггрегация

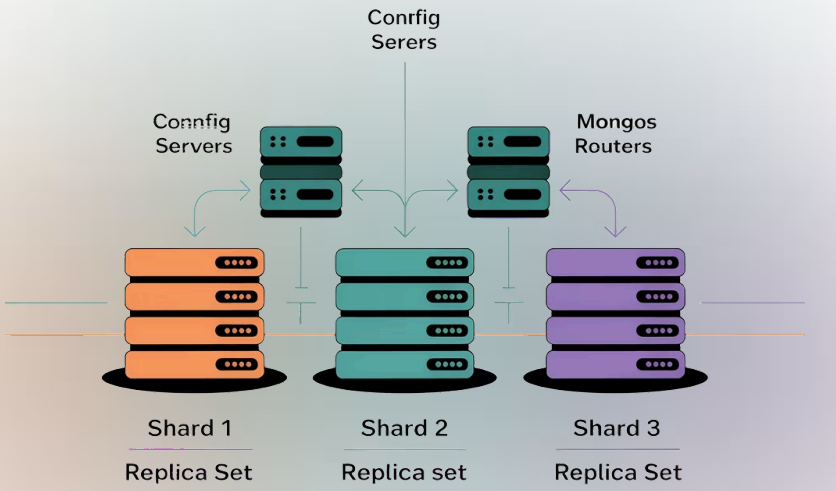
```
// Простой агрегационный конвейер
db.sales.aggregate([
  // Этап 1: фильтрация документов
  { $match: { status: "completed" } },

  // Этап 2: группировка по категории
  { $group: {
      _id: "$category",
      totalSales: { $sum: "$amount" },
      count: { $sum: 1 }
  }},

  // Этап 3: сортировка по общим продажам
  { $sort: { totalSales: -1 } }
])
```

Массивы в аггрегациях

```
db.products.aggregate([
  // Разворачивание массива тегов в отдельные документы
  { $unwind: "$tags" },

  // Группировка по тегам и подсчет продуктов
  { $group: {
      _id: "$tags",
      productCount: { $sum: 1 },
      products: { $push: "$name" }
  }},

  // Фильтрация, чтобы оставить только популярные теги
  { $match: { productCount: { $gt: 5 } } }
])
```

Продвинутые агрегации с $facet и $bucket

```
db.sales.aggregate([
  // Параллельные агрегации с разными группировками
  { $facet: {
      // Агрегация 1: продажи по категориям
      "byCategory": [
        { $group: { _id: "$category", total: { $sum: "$amount" } } },
        { $sort: { total: -1 } },
        { $limit: 3 }
      ],
      // Агрегация 2: распределение по ценовым диапазонам
      "byPriceRange": [
        { $bucket: {
            groupBy: "$amount",
            boundaries: [0, 1000, 5000, 10000, 50000],
            default: "50000+",
            output: { count: { $sum: 1 }, total: { $sum: "$amount" } }
        }}
      ]
  }}
])
```



Распределенные возможности

Консистентность и уровни гарантий

MongoDB позволяет настраивать компромисс между консистентностью и производительностью через параметры writeConcern и readConcern.

```
// Запись с разными уровнями гарантий
db.orders.insertOne(
  { item: "laptop", price: 50000 },
  { writeConcern: { w: 1 } }  // Подтверждение от одного узла
)

db.orders.insertOne(
  { item: "phone", price: 20000 },
  { writeConcern: { w: "majority", j: true } }  // Подтверждение от большинства с журналированием
)

// Чтение с разными уровнями гарантий
db.orders.find({}, { readConcern: { level: "local" } })  // Быстрое чтение
db.orders.find({}, { readConcern: { level: "majority" } })  // Согласованное чтение
```

Спасибо за внимание!

Беседа курса в Telegram