Тема 15. Язык запросов MongoDB. Агрегация данных. Индексация и поиск.

Цель занятия:

Изучить основные аспекты использования языка запросов MongoDB, включая агрегацию данных, индексацию и методы поиска.

Учебные вопросы:

1. Агрегация данных.
2. Индексация
3. Поиск

1. Агрегация данных.

Агрегация в MongoDB — это процесс обработки данных и возврата вычисленных результатов.

Она позволяет выполнять сложные манипуляции с данными, такие как фильтрация, сортировка и группировка.

Этапы агрегации (Pipeline)

Агрегация строится с использованием конвейера (pipeline), который состоит из нескольких этапов. Каждый этап обрабатывает документы и передает их следующему.

Основные этапы:

- **$match**: Фильтрует документы, аналогично оператору find. Используется для предварительной фильтрации данных.
- **$group**: Группирует документы по указанному полю и позволяет выполнять операции над каждой группой, такие как подсчет количества, суммирование и др.
- **$sort**: Сортирует документы по указанным полям.
- **$project**: Изменяет структуру документов, позволяет выбрать, какие поля будут возвращены.
- **$limit** и **$skip**: Ограничивают количество возвращаемых документов и пропускают определенное количество документов соответственно.

1. $match

Назначение: Фильтрация документов на основе заданных условий.

Пример:

```
{ $match: { status: "active" } }
```

Используется для удаления ненужных данных на ранних этапах обработки.

2. $group

Назначение: Группировка документов по определенному полю и выполнение агрегатных операций над каждой группой.

Пример:

```
{ $group: { _id: "$category", totalSales: { $sum: "$sales" } } }
```

Здесь документы группируются по категории, и подсчитывается общая сумма продаж.

3. $sort

Назначение: Сортировка документов по указанным полям.

Пример:

```
{ $sort: { createdAt: -1 } }
```

Сортирует документы по дате создания в порядке убывания.

4. $project

Назначение: Изменение структуры документов. Позволяет выбрать, какие поля должны быть включены в результаты.

Пример:

```
{ $project: { name: 1, totalPrice: { $multiply: ["$price", "$quantity"] } } }
```

Включает только поля name и totalPrice, который рассчитывается на основе других полей.

5. $limit и $skip

$limit: Ограничивает количество возвращаемых документов.

Пример:

```
{ $limit: 10 }
```

Возвращает только 10 документов.

$skip: Пропускает указанное количество документов.

Пример:

```
{ $skip: 5 }
```

Пропускает первые 5 документов.

Дополнительные этапы

$unwind

Назначение: Разворачивает массивы, создавая отдельный документ для каждого элемента массива.

Пример:

```
{ $unwind: "$tags" }
```

Создает отдельный документ для каждого элемента массива tags.

$lookup

Назначение: Выполняет операцию соединения (join) с другой коллекцией.

Пример:

```
{
  $lookup: {
    from: "orders",
    localField: "customerId",
    foreignField: "_id",
    as: "customerOrders"
  }
}
```

Соединяет документы с коллекцией orders.

$out

Назначение: Записывает результаты агрегации в новую коллекцию.

Пример:

```
{ $out: "aggregatedResults" }
```

Переносит результаты в коллекцию aggregatedResults.

Пример 1: Подсчет количества документов в каждой категории:

```
db.collection.aggregate([
  { $match: { status: "active" } },
  { $group: { _id: "$category", count: { $sum: 1 } } }
])
```

Пример 2: Средняя цена товаров по категориям:

```
db.collection.aggregate([
  { $group: { _id: "$category", avgPrice: { $avg: "$price" } } }
])
```

Пример 3: Топ-5 самых дорогих товаров:

```
db.collection.aggregate([
  { $sort: { price: -1 } },
  { $limit: 5 }
])
```

Агрегационные функции:

- **$sum**: Вычисляет сумму значений.
- **$avg**: Вычисляет среднее значение.
- **$min** и **$max**: Находит минимальное и максимальное значения соответственно.

$sum

Назначение: Подсчитывает сумму значений поля.

Пример:

```
db.sales.aggregate([
  {
    $group: {
      _id: "$product",
      totalSales: { $sum: "$amount" }
    }
  }
])
```

Подсчитывает общие продажи для каждой категории.

$avg

Назначение: Вычисляет среднее значение поля.

Пример:

```
db.sales.aggregate([
  {
    $group: {
      _id: "$category",
      averagePrice: { $avg: "$price" }
    }
  }
])
```

Вычисляет среднюю цену для каждой категории.

$min

Назначение: Находит минимальное значение поля.

Пример:

```
db.products.aggregate([
  {
    $group: {
      _id: null,
      minPrice: { $min: "$price" }
    }
  }
])
```

Находит минимальную цену среди всех продуктов.

$max

Назначение: Находит максимальное значение поля.

Пример:

```
db.products.aggregate([
  {
    $group: {
      _id: null,
      maxPrice: { $max: "$price" }
    }
  }
])
```

Находит максимальную цену среди всех продуктов..

$push

Назначение: Создает массив значений из группы.

Пример:

```
db.orders.aggregate([
  {
    $group: {
      _id: "$customerId",
      orders: { $push: "$orderId" }
    }
  }
])
```

Создает массив с идентификаторами заказов для каждого клиента.

$addToSet

Назначение: Создает массив уникальных значений из группы.

Пример:

```
db.orders.aggregate([
  {
    $group: {
      _id: "$customerId",
      uniqueProducts: { $addToSet: "$productId" }
    }
  }
])
```

Создает массив уникальных товаров для каждого клиента.

$first

Назначение: Возвращает первое значение из группы.

Пример:

```
db.sales.aggregate([
  {
    $sort: { date: 1 } // Сортировка по дате, чтобы $first работал корректно
  },
  {
    $group: {
      _id: "$category",
      firstProduct: { $first: "$productName" }
    }
  }
])
```

Возвращает первый продукт в каждой категории.

$sort: Сначала сортируем документы по дате. Это важно, чтобы $first возвращал первый продукт по времени в каждой категории.

$group: Группировка по полю category, извлечение первого продукта в каждой категории с помощью $first.

$last

Назначение: Возвращает последнее значение из группы.

Пример:

```
db.sales.aggregate([
  {
    $sort: { date: 1 } // Сортировка по дате, чтобы $last работал корректно
  },
  {
    $group: {
      _id: "$category",
      lastProduct: { $last: "$productName" }
    }
  }
])
```

Возвращает последний продукт в каждой категории.

$sort: Сортируем документы по дате, чтобы $last возвращал последний продукт по времени в каждой категории.

$group: Группировка по полю category, извлечение последнего продукта в каждой категории с помощью $last.

Пример сложного запроса:

```
db.orders.aggregate([
  // 1. Фильтрация заказов по статусу
  { $match: { status: "completed" } },

  // 2. Разворачивание массива товаров
  { $unwind: "$items" },

  // 3. Группировка по идентификатору товара и подсчет количества продаж
  {
    $group: {
      _id: "$items.productId",
      totalQuantity: { $sum: "$items.quantity" },
      totalRevenue: { $sum: { $multiply: ["$items.price", "$items.quantity"] } }
    }
  },
```

```
// 4. Сортировка по общему доходу
{ $sort: { totalRevenue: -1 } },

// 5. Ограничение до топ-10 самых прибыльных товаров
{ $limit: 10 },

// 6. Соединение с коллекцией продуктов для получения информации о товаре
{
  $lookup: {
    from: "products",
    localField: "_id",
    foreignField: "_id",
    as: "productDetails"
  }
},
```

```
  // 7. Разворачивание массива с информацией о товаре
  { $unwind: "$productDetails" },

  // 8. Проекция нужных полей
  {
    $project: {
      _id: 0,
      productId: "$_id",
      productName: "$productDetails.name",
      totalQuantity: 1,
      totalRevenue: 1
    }
  }
])
```

Этот конвейер выполняет следующие действия:

- Фильтрация заказов с завершенным статусом.
- Разворачивание массива товаров в заказах.
- Группировка по идентификатору товара для подсчета общего количества и выручки.
- Сортировка по общей выручке в порядке убывания.
- Ограничение результатов до топ-10 товаров.
- Соединение с коллекцией products для получения деталей о каждом товаре.
- Разворачивание массива с информацией о товаре.
- Проекция для выбора необходимых полей в итоговом результате.

2. Индексация

Зачем нужны индексы?

- Ускорение поиска: Индексы позволяют MongoDB находить нужные документы быстрее, избегая полного сканирования коллекции (что особенно полезно при больших объемах данных).
- Оптимизация сортировки: Сортировка документов также ускоряется при наличии индексов.
- Уникальность: Индексы могут гарантировать уникальность значений в полях (например, для email-адресов).
- Уменьшение нагрузки на систему: Индексы снижают затраты на ресурсы при выполнении сложных запросов.

Типы индексов в MongoDB

Однопольные индексы (Single Field Index)

Индекс создается на одном поле, чтобы ускорить поиск по этому полю.

Пример:

```
db.books.createIndex({ author: 1 })
```

Здесь индекс на поле author сортируется по возрастанию (1). Использование -1 означало бы сортировку по убыванию.

Составные индексы (Compound Index)

Индекс создается на нескольких полях для оптимизации запросов, использующих сразу несколько условий.

Пример:

```
db.books.createIndex({ author: 1, year: -1 })
```

Этот индекс ускоряет запросы, которые фильтруют книги по автору и году.

Текстовые индексы (Text Index)

Позволяют выполнять полнотекстовый поиск по строковым полям.

Пример:

```
db.books.createIndex({ title: "text", description: "text" })
```

Запрос для поиска по этим полям:

```
db.books.find({ $text: { $search: "Fantasy" } })
```

Геопространственные индексы (Geospatial Index)

Используются для работы с координатами и геоданными. MongoDB поддерживает индексы 2dsphere для данных в формате GeoJSON.

Пример:

```
db.locations.createIndex({ location: "2dsphere" })
```

Запрос для поиска ближайших точек:

```
db.locations.find({
  location: {
    $near: {
      $geometry: { type: "Point", coordinates: [50, 50] },
      $maxDistance: 10000  // В метрах
    }
  }
})
```

Просмотр всех индексов в коллекции:

```
db.books.getIndexes()
```

Удаление индекса:

По имени индекса:

```
db.books.dropIndex("author_1")
```

Удаление всех индексов в коллекции:

```
db.books.dropIndexes()
```

3. Поиск

MongoDB предлагает множество инструментов для выполнения поиска в коллекциях, включая полнотекстовый поиск, поиск с учетом локали, регулярные выражения и фильтрацию по диапазонам. Далее рассмотрим каждый из этих инструментов.

1. Поиск по тексту

Для выполнения полнотекстового поиска необходимо создать текстовый индекс на одном или нескольких строковых полях.

Создание текстового индекса:

```
db.books.createIndex({ title: "text", description: "text" })
```

Пример текстового поиска:

```
db.books.find({ $text: { $search: "Fantasy" } })
```

Использование оператора $text:

$search: Поиск слова или фразы.

$caseSensitive: Можно настроить регистрозависимый поиск.

```
db.books.find({ $text: { $search: "Fantasy", $caseSensitive: true } })
```

Поиск фразы:

```
db.books.find({ $text: { $search: "\"high fantasy\"" } })
```

Ищет точное совпадение фразы

Поиск с использованием регулярных выражений

MongoDB поддерживает регулярные выражения для поиска строковых совпадений.

Пример поиска с регулярными выражениями:

```
db.books.find({ title: { $regex: /^Hobbit/, $options: "i" } })
```

Hobbit: Ищет строки, которые начинаются с "Hobbit".

$options: "i": Указывает, что поиск нечувствителен к регистру.

Поиск по диапазонам и условиям

Для поиска документов по числовым значениям или диапазонам используются операторы, такие как $gt, $lt, $gte, $lte, и $in.

Пример поиска по диапазону:

Найти все книги, выпущенные между 1950 и 2000 годами:

```
db.books.find({ year: { $gte: 1950, $lte: 2000 } })
```

Поиск с несколькими условиями:

```
db.books.find({
  $and: [
    { price: { $gt: 10 } },
    { available: true }
  ]
})
```

Сочетание поиска по диапазону и регулярным выражениям

Пример комбинированного запроса, который ищет книги с ценой выше 20 и названием, содержащим слово "Magic":

```
db.books.find({
  $and: [
    { price: { $gt: 20 } },
    { title: { $regex: /Magic/, $options: "i" } }
  ]
})
```

Заключение

MongoDB предлагает широкий спектр возможностей для поиска: от простого полнотекстового поиска до локализованных запросов и регулярных выражений. Эти инструменты позволяют гибко обрабатывать данные и находить нужную информацию. Использование текстовых индексов и правильных фильтров также способствует повышению производительности запросов.

Домашнее задание:

1. Повторить материал лекции.

Список литературы:

1. В. Ю. Кара-ушанов SQL — язык реляционных баз данных
2. А. Б. ГРАДУСОВ. Введение в технологию баз данных
3. А.Мотеев. Уроки MySQL

Материалы лекций:

https://github.com/ShViktor72/Education

Обратная связь:

colledge20education23@gmail.com