MongoDB ч.3

Индексы

- Индексы — это отдельная структура данных, которую поддерживает база данных для быстрого поиска.
- Компромисс здесь заключается в том, что индексы могут привести к тому, что вставки, обновления и удаления будут немного медленнее, потому что они также должны обновлять индексы, а также занимать больше места на диске. Но взамен вы получаете очень быстрые запросы.

Простой индекс поля

```
{
  "_id":
ObjectId("5f9e83f16d4e88cc16dbd7a1"),
  "name": "Kevin",
  "age": 32,
  "color": "yellow",
  "hobbies": ["banana eating",
"mischief"],
  "address": {
    "city": "Minionville",
    "country": "Despicable Land"
  }
}
```

```
db.minions.createIndex({ "name": 1 })
```

Простой индекс поля с явным именем

```
{
  "_id":
ObjectId("5f9e83f16d4e88cc16dbd7a1"),
  "name": "Kevin",
  "age": 32,
  "color": "yellow",
  "hobbies": ["banana eating",
"mischief"],
  "address": {
    "city": "Minionville",
    "country": "Despicable Land"
  }
}
```

```
db.minions.createIndex({ "name": 1 }, { name: "index_name" })
```

Индекс поля вложенного документа

```
{
  "_id":
ObjectId("5f9e83f16d4e88cc16dbd7a1"),
  "name": "Kevin",
  "age": 32,
  "color": "yellow",
  "hobbies": ["banana eating",
"mischief"],
  "address": {
    "city": "Minionville",
    "country": "Despicable Land"
  }
}
```

```
db.minions.createIndex({ «address.city": 1 })
```

Удаление индекса

```
{
  "_id":
ObjectId("5f9e83f16d4e88cc16dbd7a1"),
  "name": "Kevin",
  "age": 32,
  "color": "yellow",
  "hobbies": ["banana eating",
"mischief"],
  "address": {
    "city": "Minionville",
    "country": "Despicable Land"
  }
}
```

```
db.minions.dropIndex("index_name")
```

Рассмотрим простой запрос

```
db.pets.find({ name: "Fido" });
```

- Найдите всех питомцев по имени Fido.
- Проблема в том, что этот запрос делает много работы: он фактически заставляет базу данных просматривать каждую запись в базе данных.

Рассмотрим простой запрос

- Рассмотрим запрос подробнее:

```
db.pets.find({ name: "Fido" }).explain("executionStats");
```

```
winningPlan: {
  stage: 'COLLSCAN',
  filter: {
    name: {
      '$eq': 'Luna'
    }
  },
  direction: 'forward'
```

```
executionStats: {
  executionSuccess: true,
  nReturned: 1112,
  executionTimeMillis: 3,
  totalKeysExamined: 0,
  totalDocsExamined: 10000,
  executionStages: {
    stage: 'COLLSCAN',
    filter: {
      name: {
        '$eq': 'Luna'
      }
    },
    nReturned: 1112,
    executionTimeMillisEstimate: 0,
    works: 10002,
    advanced: 1112,
    needTime: 8889,
    needYield: 0,
```

Рассмотрим простой запрос

В базовом случае просматривается каждая запись в нашей базе данных и используется COLLSCAN стратегия, аналогичная линейному поиску, со сложностью O(n).

Создадим индексы и проверим

```
winningPlan: {
  stage: 'FETCH',
  inputStage: {
    stage: 'IXSCAN',
    keyPattern: {
      name: 1
    },
    indexName: 'name_1',
    isMultiKey: false,
    multiKeyPaths: {
      name: []
    },
```

```
db.pets.createIndex({ name: 1 });
db.pets.find({ name: "Fido" }).explain("executionStats");
db.pets.find({ name: "Fido" }).count();
db.pets.getIndexes();
```

```
executionTimeMillis: 1,
```

Составные индексы

Если вы часто для поиска используете два ключа вместе, например тип и породу собаки, вы можете использования составной индекс. Поиск по данным индексам вместе будет работать быстрее, нежели если делать два индекса отдельно.

Составной индекс

```
{
  "_id":
ObjectId("5f9e83f16d4e88cc16dbd7a1"),
  "name": "Kevin",
  "age": 32,
  "color": "yellow",
  "hobbies": ["banana eating",
"mischief"],
  "address": {
    "city": "Minionville",
    "country": "Despicable Land"
  }
}
```

```
db.minions.createIndex({ "name": 1, "color": 1 })
```

Уникальные индексы

Индексы, сделанные по уникальным полям.

```
db.pets.createIndex({ index: 1 }, { unique: true });

# Попробуем добавить новую запись с повторным индексом
db.pets.insertOne({ name: "Doggo", index: 10 });
```

```
▸ MongoServerError: E11000 duplicate key error collection: pets.pets index: index_1 dup key: { index: 10 }
```

Запросим запись по индексу

```
db.pets.find({ index: 1337 }).explain("executionStats");
```

Отметим, что благодаря индексу, смотрится только 1 запись!

Полнотекстовый поиск

Итак, в MongoDB можно создать текстовый индекс. Каждая коллекция может иметь только один текстовый индекс, поэтому убедитесь, что вы индексируете все нужные поля. В нашем случае давайте индексировать тип, породу и имя.

```
db.pets.createIndex({
  type: "text",
  breed: "text",
  name: "text",
});
```

Специальный оператор $text

Для поиска по текстовым индексам используется специальный оператор **$text**.
Простое совпадение, без сортировки по точности совпадения:

```
db.pets.find({ $text: { $search: "dog Havanese Luna" } });
```

Данный запрос вернет все документы, которые содержат хотя бы одно из указанных слов в поле, проиндексированное текстовым индексом. Результат будет содержать документы, которые соответствуют хотя бы одному из критериев поиска.

Специальный оператор $text

Для поиска по текстовым индексам используется специальный оператор **$text**.
Точное совпадение ближе к началу результата:

```
db.pets
  .find({ $text: { $search: "dog Havanese Luna" } })
  .sort({ score: { $meta: "textScore" } });
```

Сортирует результаты по рейтингу (textScore) в порядке убывания. Это позволяет получить документы, которые наиболее соответствуют запросу.

Маркеры для результата

В MongoDB работают операторы маркеры как в поисковой системе Google. Если вы хотите найти всех Luna, которые не являются кошками:

```
db.pets
  .find({ $text: { $search: "-cat Luna" } })
  .sort({ score: { $meta: "textScore" } });
```

Подробнее тут:
https://www.mongodb.com/docs/manual/reference/operator/query/text/

Агрегации

- Операции агрегации обрабатывают записи данных и возвращают вычисленные результаты.
- Операции агрегации группируют значения из нескольких документов вместе и могут выполнять различные операции над сгруппированными данными для возврата одного результата.
- В SQL count (*) и с group by является эквивалентом агрегации mongodb.

Агрегации

- MongoDB предоставляет три способа выполнения агрегации: pipeline, Map-Reduce и одноцелевые методы агрегирования.

  - pipeline - фреймворк для агрегации в MongoDB моделирует концепцию обработку данных с помощью pipeline. Документы вводят многоэтапный конвейер, который преобразует документы в агрегированный результат.
  - map-reduce — это алгоритм предложенный гугл для обработки больших данных. Тут все довольно просто есть две функции одна map, которая удаляет поля в документах по определенным признакам и группирует их и функция reduce, которая может свернуть значение сгруппированных документов.

Агрегации-pipeline

```
db.minions.aggregate([
    { $match: { age: { $gte: 18 } } },
    { $group: { _id: "$color", count: { $sum: 1 } } },
    { $sort: { count: -1 } },
]);
```

Агрегации-Map-Reduce:

```
db.minions.mapReduce(
    functionMap () {
        if (this.age >= 18) {
            emit(this.color, 1);
        }
    },
    functionReduce (key, values) {
        return Array.sum(values);
    },
    {
        query: { age: { $gte: 18 } },
        out: "minions_color_count"
    }
);
```

Функция map: Принимает каждый документ из коллекции, фильтрует по возрасту и эмиттирует пары ключ-значение, где ключ - цвет, а значение – 1

Функция reduce: Агрегирует значения, суммируя их для каждого ключа

Агрегации

- Одноцелевая агрегация это агрегация одной коллекции по определенному ключу.
- Рекомендованный вариант – pipeline.

Одноцелевой метод count и метод агрегации:

```
db.minions.count({ age: { $gte: 18 } });

db.minions.aggregate([
    { $group: { _id: null, totalMinions: { $sum: 1 } } }
]);
```

Агрегации

- Что, если бы мы захотели узнать, сколько щенков, взрослых и пожилых собак есть в нашей коллекции питомцев?

```
db.pets.aggregate([
  {
    $bucket: {
      groupBy: "$age",
      boundaries: [0, 3, 9, 15],
      default: "very senior",
      output: {
        count: { $sum: 1 },
      },
    },
  },
]);
```

Агрегации

- Что, если бы мы захотели узнать, сколько щенков, взрослых и пожилых собак есть в нашей коллекции питомцев?

```
db.pets.aggregate([
  {
    $bucket: {
      groupBy: "$age",
      boundaries: [0, 3, 9, 15],
      default: "very senior",
      output: {
        count: { $sum: 1 },
      },
    },
  },
]);
```

Агрегации

- К примеру нам нужны только собаки. Добавим еще один этап.

```
db.pets.aggregate([
  {
    $match: {
      type: "dog",
    },
  },
  {
    $bucket: {
      groupBy: "$age",
      boundaries: [0, 3, 9, 15],
      default: "very senior",
      output: {
        count: { $sum: 1 },
      },
    },
  },
]);
```

```
{
  _id: 0,
  count: 278
}
{
  _id: 3,
  count: 834
}
{
  _id: 9,
  count: 833
}
{
  _id: 'very senior',
  count: 555
}
```

Задание

- Создать коллекцию документов по выбранной вами
- предметной области
- Сгенерировать 10000 документов
- Осуществить запросы на поиск данных
- Осуществить запросы на создание и изменение данных
- Осуществить запросы на удаление данных
- Создать индексы и сверить трудоемкость запросов после индексов
- Осуществить агрегацию

Есть вопросы?