Тема 8. Язык запросов MongoDB.

Цель занятия:

Ознакомиться с основами языка запросов MongoDB и научиться выполнять базовые CRUD операции (создание, чтение, обновление, удаление) с документами в коллекциях.

Учебные вопросы:

1. Введение в язык запросов MongoDB.
2. Операции CRUD.
3. Выполнение простых запросов.

1. Введение в язык запросов MongoDB.

MongoDB использует свой язык запросов (MongoDB Query Language, MQL), основанный на JSON, который позволяет взаимодействовать с базой данных для выполнения операций CRUD (создание, чтение, обновление, удаление).

MongoDB предоставляет удобные инструменты для работы с коллекциями и документами, где запросы выглядят как объекты JSON, а команды и операторы позволяют гибко и эффективно управлять данными.

В MongoDB данные хранятся в виде документов, которые объединяются в коллекции.

Документ: это основная единица данных, представляющая собой JSON-объект. Документ MongoDB хранит данные в формате BSON, который представляет собой двоичный формат JSON.

Коллекция: это набор документов. В реляционных базах данных аналогом коллекции является таблица, а документ можно сравнить со строкой в таблице.

В примере документ содержит поля различного типа (строки, числа, массивы, вложенные документы)

```
{
  "_id": "507f1f77bcf86cd799439011",
  "name": "John Doe",
  "age": 29,
  "email": "john.doe@example.com",
  "address": {
    "street": "123 Main St",
    "city": "Somewhere",
    "state": "NY"
  },
  "hobbies": ["reading", "hiking", "coding"]
}
```

Особенности работы с BSON-форматом

BSON (Binary JSON) — это двоичный формат, используемый MongoDB для хранения данных. BSON оптимизирован для быстрого хранения и обработки данных, при этом сохраняя гибкость JSON.

Ключевые особенности BSON:

- Типизация данных: BSON поддерживает больше типов данных, чем JSON, такие как int, long, date, binary data.
- Эффективное хранение: данные хранятся в бинарном формате, что улучшает производительность запросов и операций над документами.
- Полная совместимость с JSON: BSON поддерживает все структуры данных, которые есть в JSON, что делает его простым для использования.

2. Операции CRUD.

Операции CRUD — это базовые операции для работы с данными в MongoDB, которые позволяют добавлять, читать, изменять и удалять документы в коллекциях.

1. Создание (Create): Вставка документов в коллекцию

MongoDB поддерживает два основных метода для добавления данных в коллекцию:

- insertOne() — для добавления одного документа.
- insertMany() — для добавления нескольких документов сразу.

Пример использования insertOne():

```
db.users.insertOne({
  name: "Alice",
  age: 30,
  email: "alice@example.com"
});
```

Пример использования insertMany():

```
db.users.insertMany([
  { name: "Bob", age: 25, email: "bob@example.com" },
  { name: "Charlie", age: 35, email: "charlie@example.com" }
]);
```

2. Чтение (Read): Выполнение запросов для получения данных

Для чтения данных используются методы:

- find() — для получения всех документов, соответствующих условиям запроса.
- findOne() — для получения одного документа.

Пример использования find(), запрос возвращает всех пользователей, чей возраст больше 30:

```
db.users.find({ age: { $gt: 30 } });
```

Пример использования findOne(), запрос вернет первый документ с именем "Alice":

```
db.users.findOne({ name: "Alice" });
```

3. Обновление (Update): Изменение существующих документов.

MongoDB предоставляет два метода для обновления документов:

- updateOne() — обновляет первый найденный документ, соответствующий условиям.
- updateMany() — обновляет все документы, соответствующие условиям.

Для изменения полей документа используются операторы:

- $set — для изменения значений полей.
- $inc — для увеличения (или уменьшения) значения числового поля.

Пример использования updateOne() с оператором $set.

Этот запрос обновит возраст пользователя "Alice" до 31:

```
db.users.updateOne(
  { name: "Alice" },
  { $set: { age: 31 } }
);
```

Пример использования updateMany() с оператором $inc.

Этот запрос увеличит возраст всех пользователей младше 30 на 1:

```
db.users.updateMany(
  { age: { $lt: 30 } },
  { $inc: { age: 1 } }
);
```

4. Удаление (Delete): Удаление документов.

Для удаления документов MongoDB предоставляет методы:

- deleteOne() — удаляет первый документ, соответствующий условиям.
- deleteMany() — удаляет все документы, соответствующие условиям.

Пример использования deleteOne().

Этот запрос удалит первого пользователя с именем "Bob":

```
db.users.deleteOne({ name: "Bob" });
```

Пример использования deleteMany().

Этот запрос удалит всех пользователей старше 35 лет:

```
db.users.deleteMany({ age: { $gt: 35 } });
```

3. Выполнение простых запросов.

1. Основы фильтрации: использование операторов сравнения:

$eq — равенство.

$gt — больше, чем.

$lt — меньше, чем.

$gte — больше или равно.

$lte — меньше или равно.

$ne — не равно.

$in — соответствие одному из элементов списка.

$nin — не входит в список.

Примеры фильтрации с операторами сравнения:

Поиск всех пользователей старше 30 лет:

```
db.users.find({ age: { $gt: 30 } });
```

Поиск пользователей с возрастом 25 или 30:

```
db.users.find({ age: { $in: [25, 30] } });
```

Поиск пользователей с именем "Alice":

```
db.users.find({ name: { $eq: "Alice" } });
```

Фильтрация по вложенным документам и массивам

MongoDB поддерживает фильтрацию по полям, которые содержат вложенные документы и массивы.

Фильтрация по вложенным документам:

Для фильтрации по вложенным полям используется синтаксис с точкой (.).

Пример фильтрации по адресу в документе. В данном случае address — это вложенный документ, а city — одно из его полей.

```
db.users.find({ "address.city": "New York" });
```

Фильтрация по массивам:

MongoDB позволяет фильтровать документы по элементам массивов.

Пример фильтрации пользователей, у которых в массиве хобби есть "hiking":

```
db.users.find({ hobbies: "hiking" });
```

Поиск документов, где массив содержит все указанные значения:

```
db.users.find({ hobbies: { $all: ["reading", "coding"] } });
```

Примеры сложных условий с использованием логических операторов

MongoDB поддерживает логические операторы для выполнения сложных условий в запросах.

Основные логические операторы:

- $or — выполняет запрос, если хотя бы одно из условий верно.
- $and — выполняет запрос, если все условия верны.
- $not — выполняет запрос, если условие ложно.
- $nor — выполняет запрос, если ни одно из условий не верно.

Примеры использования логических операторов:

Поиск пользователей, у которых возраст меньше 25 или больше 30:

```
db.users.find({
  $or: [
    { age: { $lt: 25 } },
    { age: { $gt: 30 } }
  ]
});
```

Поиск пользователей, чье имя "Alice" и возраст больше 25:

```
db.users.find({
  $and: [
    { name: "Alice" },
    { age: { $gt: 25 } }
  ]
});
```

Поиск пользователей, чей возраст не равен 30:

```
db.users.find({
  age: { $not: { $eq: 30 } }
});
```

Поиск пользователей, которые живут в "New York" и чье имя не "Bob", или у которых возраст больше 40:

```
db.users.find({
  $or: [
    { $and: [
        { "address.city": "New York" },
        { name: { $ne: "Bob" } }
      ]
    },
    { age: { $gt: 40 } }
  ]
});
```

Сортировка данных в MongoDB.

MongoDB предоставляет возможность сортировки данных с помощью метода sort(), который используется для упорядочивания документов в соответствии с одним или несколькими полями. Сортировка может быть выполнена по числовым, строковым или другим типам данных.

Метод **sort()** используется для упорядочивания результатов запроса по значениям полей.

Синтаксис:

```
db.collection.find().sort({ field: order });
```

field — это поле, по которому выполняется сортировка.

order — это порядок сортировки:

1 — по возрастанию (от меньшего к большему).

-1 — по убыванию (от большего к меньшему).

Пример сортировки по одному полю

Пример сортировки пользователей по возрасту (по возрастанию):

```
db.users.find().sort({ age: 1 });
```

Пример сортировки пользователей по имени (по убыванию):

```
db.users.find().sort({ name: -1 });
```

Пример сортировки по нескольким полям

MongoDB также поддерживает сортировку по нескольким полям. В этом случае при сортировке сначала будет учитываться первое указанное поле, а затем — второе.

Пример сортировки пользователей сначала по возрасту (по убыванию), затем по имени (по возрастанию):

```
db.users.find().sort({ age: -1, name: 1 });
```

Проекция данных в MongoDB.

Проекция — это механизм, который позволяет выбирать, какие поля документа будут отображены в результатах запроса. Это полезно, когда нужно получить только часть данных из документа, что сокращает объем передаваемых данных и повышает производительность запроса.

Использование метода projection()

Для управления выводом полей используется метод projection() или просто дополнительный аргумент в методе find(). По умолчанию, если проекция не указана, MongoDB возвращает все поля документа.

Синтаксис:

```
db.collection.find(query, { field1: 1, field2: 0 });
```

field1: 1 — поле включается в результат (отображается).

field2: 0 — поле исключается из результата (скрывается).

Примеры скрытия и отображения полей документа

Пример отображения определенных полей:

Отображение только полей name и age у пользователей:

```
db.users.find({}, { name: 1, age: 1, _id: 0 });
```

В данном примере отображаются только поля name и age. Поле _id исключено (скрыто) с помощью _id: 0.

Пример скрытия определенных полей:

Скрытие поля email при отображении всех остальных полей:

```
db.users.find({}, { email: 0 });
```

В этом запросе все поля документа будут отображены, кроме поля email.

Проекция по вложенным полям:

Отображение только вложенного поля address.city:

```
db.users.find({}, { "address.city": 1, _id: 0 });
```

Этот запрос вернет только поле city из вложенного документа address.

Практика:

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