Лабораторная работа № 2.

В данной работе представлена пошаговая инструкция по работе с MongoDB. Все предварительные замечания, настройки системы и ввод данных были сделаны в работе № 1.

1. Введение.

В этой лабораторной работе будет показано, как изменять структуру и содержание документов в коллекции MongoDB, а также рассмотрены способы работы с датами в документах MongoDB.

2. Обновление одного документа в коллекции.

Будем использовать уже созданную вами ранее коллекцию *books*. Давайте уточним жанр книги «Убить пересмешника» Харпер Ли. Пусть это будет детектив.

Внесём изменения следующим образом:

```
db.books.updateOne(
  { title: "To Kill a Mockingbird" },
  { $set: { genres: ["Detective"] } }
);
```

Ищем документ по названию произведения (оно у нас теперь не повторяется). Конструкция **$set** устанавливает новое значение поля. Мы оставляем значение в виде массива (на всякий случай). В результате получился следующий документ:

Однако для этого произведения было бы логично оставить и тэг «классика». Поэтому добавим соответствующий элемент в массив *genres*:

```
db.books.updateOne(
  { title: "To Kill a Mockingbird" },
  { $push: { genres: "Classic" } }
);
```

После этого массив genres будет иметь два значения: ['Detective', 'Classic'].

Теперь заменим в массиве жанров романа Джорджа Оруэлла «1984» значение «Science Fiction» на «Classic». Для этого обратимся к элементу массива 1 (нумерация массивов начинается с нуля):

```
db.books.updateOne(
  { title: "1984" },
  { $set: { "genres.1": "Classic" } }
);
```

Посмотрите, всё ли получилось в обновлённом документе так, как было задумано.

Теперь удалим в документе о книге «Великий Гэтсби» из массива *genres* элемент 'Fiction'. Сделаем это так:

```
db.books.updateOne(
  { title: "The Great Gatsby" },
  { $pull: { genres: "Fiction" } }
);
```

! Замените значение в поле stock для документа о книге 'Pride and Prejudice' на 8. Внесите допустимую правку в те документы, которые вы в предыдущей работе записали в коллекцию books.

3. Обновление нескольких документов в коллекции.

Для одновременной правки нескольких документов MongoDB предоставляет метод **updateMany()**. Для того, чтобы изменить названия полей и добавить новые поля, сначала отключим схему валидации и индекс, привязанный к полю *title*:

```
db.runCommand({
    collMod: "books",
    validator: {}
});
```

Индекс, привязанный к *_id*, удалять не надо. Имя индексного файла у вас может оказаться другим:

```
> db.books.getIndexes()
< [
    { v: 2, key: { _id: 1 }, name: '_id_' },
    { v: 2, unique: true, key: { title: 1 }, name: 'title_1' }
  ]
> db.books.dropIndex("title_1")
< { nIndexesWas: 2, ok: 1 }
```

Теперь можно менять структуру документов в коллекции. Предлагается название поля «title» изменить на «title_eng», а в каждый документ коллекции *books* добавить текстовое поле с именем «title_rus», куда занести русское название произведения.

Изменяем имя поля (обязательно контролируйте результат применения этой и следующей команд):

```
db.books.updateMany(
  {}, // Фильтр {} означает "все документы в коллекции"
  {
    $rename: {
      "title": "title_eng"
    }
  }
);
```

Теперь добавляем новое поле:

```
db.books.updateMany({ }, { $set: { title_rus: null } });
```

Поскольку русских названий ещё не введено, по умолчанию в соответствующее поле записываем значение null.

! Используя updateOne(), добавьте в созданную таблицу русские переводы названий произведений. Если в введённых вами книгах были русские заголовки, переместите их в поле *title_rus* и дайте им английский перевод в поле *title_eng*. После этого вновь проиндексируйте коллекцию по полю *title_eng*. Постарайтесь не прибегать к помощи интерактивных инструментов MongoDB Compass — сделайте всё с помощью командной строки.

4. Использование дополнительных операторов обновления.

Будем использовать коллекцию *employees* из предыдущей работы. Увеличим зарплаты (поле *salary*) всем сотрудникам на 10% (т. е. умножим каждую зарплату на 1.1). Решение:

```
db.employees.updateMany({}, {$mul: {salary: 1.1} } );
```

! Используя ту же коллекцию, уменьшите зарплату сотрудникам отдела HR на 5000. Используйте вместо $mul ключевое слово $inc для увеличения значения в поле на указанное число.

Опция "Upsert" (обновление или вставка) — это особый тип операции обновления, который либо обновляет документ, если он существует, либо вставляет новый, если его нет. Это полезно для того, чтобы избежать раздельной логики "проверить, затем вставить" и обеспечивает *атомарность операций* с базой данных.

Добавим вновь безвременно ушедшего в первой работе сотрудника в коллекцию *employees*. Для этого поступаем так:

```
db.employees.updateOne(
        { name: "John Doe"},
        {$set: {age: 25, department: "Sales", salary: 55000, status: "returned" }},
        {upsert: true}
);
```

Поскольку ни один документ не соответствовал фильтру, был создан новый. Если бы этот документ существовал, значения полей просто изменились бы в соответствии с указанными.

Присмотритесь: новый документ с информацией о Джоне Доe содержит одно лишнее поле "status", которого нет у других сотрудников. Исправим эту особенность, удалив лишнее поле:

```
db.employees.updateOne(
  { name: "John Doe" },
  { $unset: { status: "Здесь может быть любое значение" } }
);
```

5. Работа с датами.

! Создайте новую коллекцию с именем *events*. Каждый документ будет представлять некое событие, которое произошло в определённое время. MongoDB хранит даты как объекты BSON Date, которые вы можете создать с помощью конструктора **new Date()**.

Примерный вид документов для коллекции *events*:

1. Первый документ использует строку конкретной даты-времени в формате ISO-8601

```
{
    event_name: "Conference",
    date: new Date("2024-06-15T10:30:00Z")
}
```

2. Второй документ использует текущую дату-время.

```
{
    event_name: "System Maintenance",
    timestamp: new Date()
}
```

3. Третий документ использует временную метку эры Unix, представляющую собой количество миллисекунд от начала 1970 года.

```
{
    event_name: "Project Deadline",
    timestamp: new Date(1718476800000)
}
```

4. Ещё несколько сходных по виду документов:

```
{
    event_name: "Summer Conference",
    date: new Date("2024-07-15T09:00:00Z")
}
```

5.

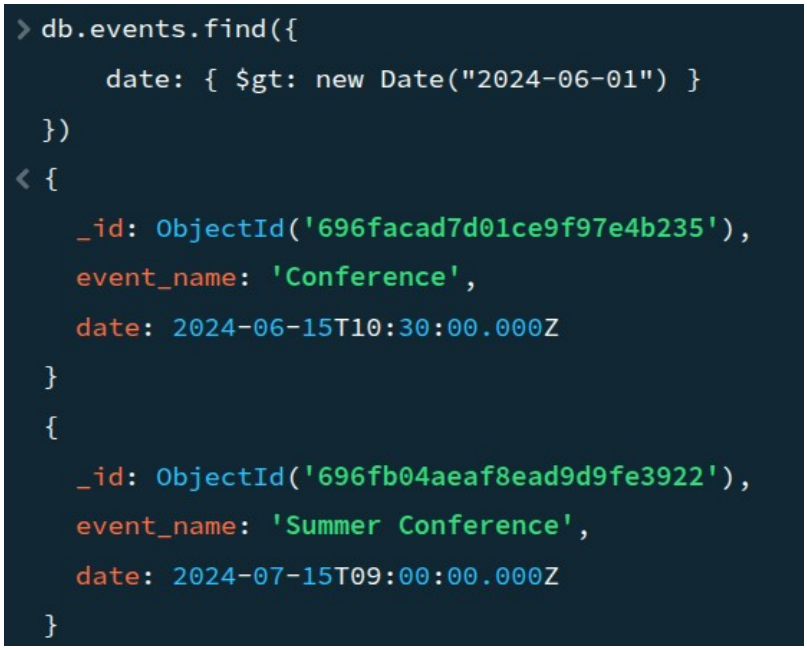
```
{
    event_name: "Winter Workshop",
    date: new Date("2024-01-20T14:30:00Z")
}
```

6.

```
{
    event_name: "Spring Meetup",
    date: new Date("2024-04-10T11:15:00Z")
}
```

Посмотрим, как будет выглядеть вывод первых трёх документов этой коллекции с помощью цепочки методов **find().limit(3)**:

Найдем все события, которые произошли после 1 июня 2024 года:



Обратите внимание, документ с полем *timestamp* в выборку не попал, поскольку условие ставилось на поле с именем *date*. Если бы мы захотели расширить условие по выбору дат на два этих поля, то написали бы так:

```
db.events.find( {$or:
    [
      {date: {$gt: new Date("2024-06-01") }},
      {timestamp: {$gt: new Date("2024-06-01") }}
    ]
});
```

Здесь мы применили синтаксическую конструкцию с условным оператором **$or**, банально предполагая, что в одном документе не будут встречаться сразу два поля *date* и *timestamp*, поскольку в противном случае в предметной области базы данных необходимо задать какую-то дополнительную логику обработки этих полей.

С помощью ключевых слов **$and**, **$or**, **$nor**, **$not** с использованием представленного выше специфического синтаксиса можно создавать достаточно сложные логические конструкции.

! Напишите запрос, который находит события, которые произошли между 31 января и 15 июня 2024 года включительно. Попробуйте сделать 2 варианта запроса: с $and и без $and.

6. Форматирование дат при выводе.

JavaScript имеет большое количество инструментов для отображения даты и времени в различных форматах. Воспользуемся лишь некоторыми возможностями, типичными для MongoDB. Давайте отформатируем поле *date* наших событий в формат "день-месяц-год" без указания времени:

```
db.events.aggregate([
  {
    $project: {
      _id: 0,
      event_name: 1,
      'дата': {$dateToString: {format: "%d-%m-%Y", date: "$date" }}
    }
  }
]);
```

Здесь **aggregate()** создаёт контейнер для размещения набора операций, **$project** задаёт структуру формируемого документа, **date: "$date"** указывает входное поле даты из исходного документа, при этом префикс **$** указывает на путь к полю. При этом поля "timestamp" этим запросом, естественно, не обрабатываются.

Вы также можете извлекать отдельные компоненты даты, такие как год или месяц, используя такие операторы как **$year**, **$month**, **$dayOfMonth** и др.

```
db.events.aggregate([
    {
        $project: {
            event_name: 1,
            год: { $year: "$date" },
            месяц: { $month: "$date" },
            день: { $dayOfMonth: "$date" }
        }
    }
]);
```

! Используя знания основ языка JavaScript, напишите запрос, который будет форматировать вывод даты и времени, обрабатывая как поле date, так и поле timestamp, и вывод которого будет аналогичен представленному на следующем скриншоте. Для облегчения написания такого запроса рекомендуется пользоваться текстовыми редакторами, поддерживающими комфортную работу с JavaScript, например, Microsoft VS Code.

Сортировка по дате ничем не будет отличаться от сортировки по другим типам данных. Чтобы отсортировать документы, добавьте метод **sort()** к запросу **find()**. В аргументе метода **sort()** ключевое поле — поле для сортировки, поле значения — порядок сортировки (значение 1 указывает на порядок от ранних к поздним, а -1 — от поздних к ранним).

Следующий запрос отсортирует коллекцию *events* по возрастанию даты, при этом не будут выведены документы, в которых нет поля *date*:

```
db.events.find({date: {$exists: true}}).sort({date: 1})
```

7. Итоговое задание.

! Напишите один либо несколько запросов, которые введут для всех документов коллекции *events* новое поле *status* и поместят в это поле значение "archived", если дата документа старше 1 января 2025 года, и значение "actual" в противном случае. Анализировать нужно как поле *data*, так и поле *timestamp*. Ну и, конечно же, запросы надо писать так, как будто у вас не 6, а 6 миллионов документов.