Лабораторная работа № 3.

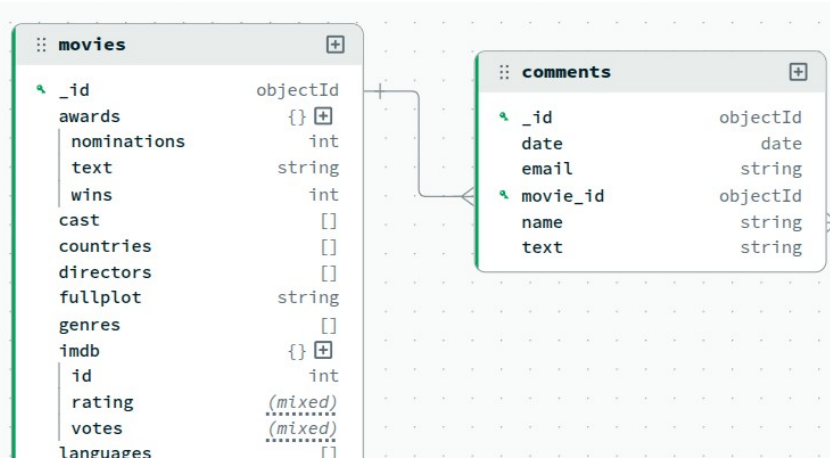
Ниже представлена пошаговая инструкция (с элементами самостоятельной работы) по изучению MongoDB. Вся информация о настройке системы и работе с данными была представлена в лабораторных работах № 1 и 2.

1. Введение.

В этой лабораторной работе будем иметь дело с коллекциями, содержащими значительное количество связанных документов. Мы изучим структуру таких документов, методы управления вложенными документами, создание документов со ссылками на другие документы, выполнение целевых запросов к вложенным данным, извлечение и объединение связанных данных.

2. Коллекции со связанными документами.

! Скопируйте из базы данных *sample_mflix* в вашу базу данных полностью две коллекции: *movies* и *comments*. Подсчитайте количество документов в каждой из этих коллекций.



Коллекция *movies* содержит подробное описание кинофильмов, представленных в некоем онлайн-кинотеатре, коллекция *comments* — якобы комментарии пользователей сайта о фильмах. Документы в коллекциях *movies* и *comments* связаны через уникальный идентификатор фильма.

Выведите содержание одного из документов (для вас — порядковый номер которого равен номеру вашего варианта, умноженному на 100, либо же найдите ваш любимый фильм) в коллекции *movies*:

В коллекции *comments* найдём все комментарии, оставленные к данному фильму:

```
db.comments.find({movie_id: ObjectId('573a1391f29313caabcd8c6b')});
```

Комментариев к выбранному вами фильму может не оказаться вовсе. В любом случае создайте собственный комментарий (можно несколько) примерно так, как показано ниже.

Проверим, что ваш комментарий действительно соответствует выбранному фильму. Для этого выведем на экран информацию о фильме вместе с комментариями к нему. Нам потребуется написать достаточно длинный запрос, поскольку будем использовать *агрегат*. Секция **$lookup** в нём отвечает за связь между коллекциями. В нашем случае связь «один ко многим» строится так, что на один фильм может быть дано любое количество комментариев от любых авторов. Секция **$match** отвечает за условия, которые накладываются на выводимую информацию. В нашем случае вы ищете результаты только для одного фильма, выбранного по его названию. Секцию **$project** мы уже использовали в предыдущих работах — это описание того, как должна выглядеть выводимая информация.

```
db.movies.aggregate([
  {
    $lookup: {
      from: "comments",
      localField: "_id", // для коллекции movies
      foreignField: "movie_id", // для коллекции comments
      as: "views" // имя, через которое поля comments будут доступны
    }
  },
  {
    $match: {
      title: 'Storm Over Asia'
    }
  },
  {
    $project: {
      _id: 0,
      title: 1,
      released: 1,
      views: 1
    }
  },
]);
```

Решим следующую задачу — попробуем применить наш агрегат для того чтобы найти, к каким фильмам пользователь *Cameron Duran* оставил свои комментарии. Пусть набор данных на выходе должен содержать название фильма, режиссёра, первые два значения жанра фильма и количество комментариев к фильму, которые оставил выбранный нами пользователь.

Перед выполнением запроса следует проиндексировать коллекцию *comments* по внешнему ключу *movie_id*, иначе поиск займёт слишком много времени.

```
db.comments.createIndex({ movie_id: 1 }, { unique: false });
```

Вот пример запроса, решающего эту задачу:

```
db.movies.aggregate([
  {
    $lookup: {
      from: "comments",
      localField: "_id",
      foreignField: "movie_id",
      as: "views"
    }
  },
  {
    $match: {
```

```
      'views.name': 'Cameron Duran'
    }
  },
  {
    $project: {
      _id: 0,
      title: 1,
      directors: 1,
      'жанр': {$slice: ["$genres", 0, 2]},
      'количество комментариев': {$size: "$views.text"}
    }
  },
]);
```

В представленном скрипте не потребовалось какой-либо внешней группировки, поскольку число комментариев можно найти, просто узнав размер массива (**$size**), состоящего из полей *text* дочерней коллекции *comments*. Первые два значения жанра фильма выводятся с помощью конструкции **$slice**, логика работы которой совпадает с оператором *slice* в языках Python и JavaScript.

! Напишите запрос, который выведет самый свежий комментарий к фильму «Один дома» вместе с названием фильма, режиссёром и списком актёров. Обратите внимание, если вы берёте за основу предыдущий скрипт, то имеет смысл добавить секцию $sort для сортировки внутри *aggregate* и ключевое слово $first для вывода первого элемента массива. Либо вы можете написать собственный вариант запроса.

3. Делаем запросы к документам со сложной структурой.

Найдём в коллекции *movies* фильмы Стивена Спилберга. Очевидно, нужно анализировать на совпадение с фамилией режиссёра поле *directors*, содержащее массив строковых значений. Поскольку мы должны проанализировать все элементы массива в этом поле, используем сравнение вида `{directors: /Spielberg/}`, где символы косой черты являются признаком регулярного выражения. В данном контексте это будет просто подстрока, которую нужно найти в массиве строк.

Поскольку каждый документ с информацией о фильме очень обширен, выведем только название фильма и год выпуска, а также отсортируем фильмы от новых к старым. Текст запроса в этом случае будет выглядеть так:

```
db.movies.find(
    {directors: /Spielberg/},
    {_id: 0, title: 1, year:{$year: "$released"}}
).sort({year: -1});
```

В коллекции, если внимательно посмотреть, уже есть готовое поле *year*, поэтому выбор года из поля *released* — лишь повод ещё раз повторить работу с датами.

Теперь попробуем найти, кто был режиссёром фильма «Назад в будущее». Если при поиске по образцу текста (название фильма) мы сомневаемся в регистре символов, можно сделать поиск нечувствительным к регистру, используя исключительно синтаксис регулярных выражений (/i). В представленном ниже запросе дано решение этой простой задачи:

```
db.movies.find(
    {title: /back to the future/i},
    {_id: 0, title: 1, year:{$year: "$released"},
      director: {$first: "$directors"}}
);
```

Для разнообразия, из поля *directors* было решено выводить только первый элемент массива (конструкция **$first**).

Далее посмотрим, есть ли в коллекции фильмы совместного производства четырёх стран:

```
db.movies.find({countries: {$size: 4}}, {title: 1, countries: 1, _id: 0});
```

И таких фильмов оказалось 497 (для подсчёта используйте метод **countDocuments**).

Найдём, сколько среди таких фильмов тех, у которых в списке стран обязательно присутствуют Советский Союз и Польша. Очевидно, что их позиции в массиве *countries* для каждого документа могут быть произвольными. Однако конструкция **$all** прекрасно справится с анализом такого массива.

```
db.movies.countDocuments(
  {$and: [
    {countries: {$size: 4}},
    {countries: {$all: ['Soviet Union', 'Poland']}}
  ]},
  {title: 1, countries: 1, _id: 0}
);
```

Сколько получилось?

! Напишите запросы, которые найдут год выпуска самого старого и самого нового фильмов из числа представленных в коллекции *movies*.

4. Группировка и сортировка документов.

Создадим небольшую коллекцию, в которой удобно проводить группировку.

```
db.products.insertMany([
  { category: "Electronics", brand: "Apple", price: 1200 },
  { category: "Electronics", brand: "Samsung", price: 800 },
  { category: "Electronics", brand: "Sony", price: 950 },
  { category: "Apparel", brand: "Nike", price: 150 },
  { category: "Apparel", brand: "Adidas", price: 120 },
  { category: "Books", brand: "Penguin", price: 25 },
  { category: "Books", brand: "Penguin", price: 35 }
]);
```

Пусть нам нужно сгруппировать документы по полю *category* с целью вычислить общую цену для каждой группы товаров. Будем использовать *агрегацию* для получения результата:

- **aggregate()** — метод, используемый для выполнения агрегации. Он принимает массив этапов, формирующих конвейер.
- **$group** — оператор этапа, который группирует входные документы.
- _id: "$category" — выражение, определяющее ключ для группировки. Префикс $ указывает на путь к полю.
- totalPrice: { $sum: "$price" } — аккумулятор. Он определяет новое поле в выходном документе под названием *totalPrice*. Оператор **$sum** вычисляет сумму поля *price* для всех документов в группе.

Этап $group может одновременно вычислять несколько аккумуляторов. Вы можете рассчитывать средние значения, находить минимальные или максимальные значения и подсчитывать элементы в каждой группе. Вот пример:

```
db.products.aggregate([
  {
    $group: {
      _id: "$category",
      totalPrice: { $sum: "$price" },
      averagePrice: { $avg: "$price" },
      productCount: { $sum: 1 }
    }
  }
]);
```

После группировки данных часто требуется отфильтровать группы на основе вычисленных значений. Например, вы можете захотеть увидеть только те категории, где общий объем продаж превышает определенную сумму. Для этой цели используется этап **$match**, который следует разместить после этапа $group.

Выведем только те категории, в которых средняя цена продуктов больше 500:

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

```
  },
  {
    $match: {
      averagePrice: { $gt: 500 }
    }
  }
]);
```

И, наконец, отсортируем группы документы по убыванию общей цены.

```
db.products.aggregate([
  {
    $group: {
      _id: "$category",
      totalPrice: { $sum: "$price" }
    }
  },
  {
    $sort: {
      totalPrice: -1
    }
  }
]);
```

! Сгруппируйте коллекцию *movies* по полю *year* и узнайте, сколько фильмов приходится на каждый год. Отсортируйте список по возрастанию лет.

5. Итоговое задание.

! Выведите из коллекции *movies* топ-10 фильмов с наибольшим рейтингом "свежести" по (якобы) версии сайта Rotten Tomatoes (поле *tomatoes*).