

ЛАБОРАТОРНАЯ РАБОТА № 8

Работа с БД в СУБД MongoDB

Цель: овладеть практическими навыками работы с CRUD-операциями, с вложенными объектами в коллекции базы данных MongoDB, агрегации и изменения данных, со ссылками и индексами в базе данных MongoDB.

Оборудование: компьютерный класс.

Программное обеспечение: СУБД MongoDB не ниже 5.0.8.

1 БАЗА ДАННЫХ ДОКУМЕНТОВ

База данных состоит из коллекций, состоящих из документов. Имя коллекции – произвольный идентификатор, состоящий из не более чем 128 различных алфавитно-цифровых символов и знака подчеркивания. В то же время имя коллекции не должно начинаться с префикса `system.`, так как он зарезервирован для внутренних коллекций (например, коллекция `system.users` содержит всех пользователей базы данных). Имя не должно содержать знака доллара `$`.

Максимальный размер BSON документа составляет 16 мегабайт.

Максимальный размер документа помогает гарантировать, что один документ не может использовать чрезмерное количество памяти или, во время передачи, чрезмерное количество трафика. Для хранения документов больше, чем максимальный размер, MongoDB обеспечивает GridFS API.

Запись в MongoDB является документом, который представляет собой структуру данных, состоящий из пар «ключ: значение»:

```
{
  field1: value1,
  field2: value2,
  field3: value3,
  ...
  fieldN: valueN
}
```

Значения полей могут включать в себя другие документы, массивы и массивы документов (рисунки 1-3).

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

← field: value
← field: value
← field: value
← field: value

Рисунок 1 – Пример структуры простого документа с использованием массива



Рисунок 2 – Пример структуры документа с вложенными документами (денормализованная модель)

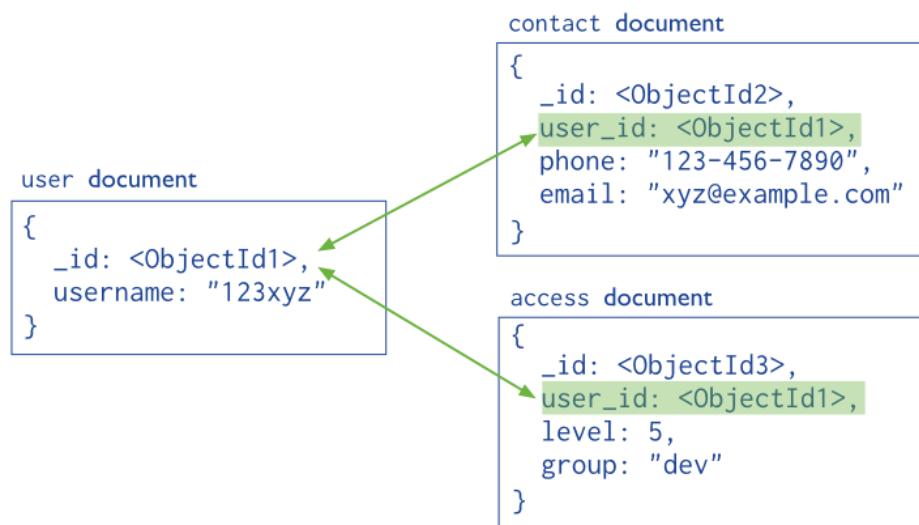


Рисунок 3 – Пример нормализованной модели с использованием ссылок между документами

В примере 3 приведена нормализованная модель БД, но это не означает наличие логических связей между коллекциями БД.

Ниже приведен пример документа, содержащий значения различных типов.

```

var mydoc = {
  _id: ObjectId("5099803df3f4948bd2f98391"),
  name: { first: "Alan", last: "Turing" },
  birth: new Date('Jun 23, 1912'),
  death: new Date('Jun 07, 1954'),
  contribs: [ "Turing machine", "Turing test", "Turingery" ],
  views : NumberLong(1250000)
}

```

2 CRUD-ОПЕРАЦИИ В СУБД MONGODB. ВСТАВКА ДАННЫХ. ВЫБОРКА ДАННЫХ

2.1 ВСТАВКА ДОКУМЕНТОВ В КОЛЛЕКЦИЮ

Для вставки в коллекцию используется функция (метод) `insert` (`insertOne`, `insertMany`). Простой пример ее использования:

```
> db.users.insert({"name": "Tom", "age": 28, languages: ["english", "spanish"]})
> db.users.find()
```

В данном случае в коллекцию `users` добавляется простой объект.

Формат JSON допускает создание как единичных объектов, так и массивов.

```
> db.users.insert([{"name": "Tom", "age": 28, languages: ["english", "spanish"]}, {"name": "Bill", "age": 32, languages: ["english", "french"]}])
```

Наиболее простой способом получения содержимого БД представляет использование метода `find`. Действие этой функции во многом аналогично обычному запросу `SELECT * FROM Table`, который извлекает все строки.

Второй способ добавления в бд документа включает два этапа: определение документа (`document = ({ ... })`) и собственно его добавление:

```
> document={"name": "Bill", "age": 32, languages: ["english", "french"]}
> db.users.insert(document)
```

2.2 ВЫБОРКА ДАННЫХ ИЗ БД

Как было показано выше, наиболее простой способ получения содержимого БД представляет использование метода `find`. В большинстве запросов возникает необходимость извлечения только тех документов, которые удовлетворяют заданным критериям.

Пусть в БД добавлены документы:

```
> db.users.insert({"name": "Tom", "age": 28, languages: ["english", "spanish"]})
> db.users.insert({"name": "Bill", "age": 32, languages: ["english", "french"]})
> db.users.insert({"name": "Tom", "age": 32, languages: ["english", "german"]})
```

Выведем все документы, имеющие `name=Tom`:

```
> db.users.find({name: "Tom"})
```

Такой запрос выведет два документа с именем `name=Tom`.

Для поиска документа по элементу в массиве используем следующий запрос: вывести все документы, у которых в массиве `languages` есть `german`.

```
> db.users.find({languages: "german"})
```

Более сложный запрос: вывести те объекты, у которых `name=Tom` и одновременно `age=32`. На языке SQL это могло бы выглядеть так: `SELECT * FROM Table WHERE Name='Tom' AND Age=32`. Данному критерию соответствует последний добавленный объект. Тогда использовать написать следующий запрос:

```
> db.users.find({name: "Tom", age: 32})
```

Для того, чтобы извлечь лишь одно значение из полученной выборки можно воспользоваться методом `findOne()`.

```
> db.users.findOne({name: "Tom"})
```

Однако, к методу `findOne()` прибегают чаще в тех ситуациях, когда ожидается, что результирующая коллекция будет содержать лишь один документ. В тех же случаях, когда следует ограничить выборку несколькими документами предпочитают использовать метод `limit()`, который принимает в качестве аргумента количество извлекаемых документов. Она задает максимально допустимое количество получаемых документов. Количество передается в виде числового параметра. Например, ограничим выборку тремя документами:

```
> db.users.find().limit(3)
```

В данном случае получены первые три документа (если в коллекции 3 и больше документов). Но что, если нужно произвести выборку не сначала, а пропустив какое-то количество документов? В этом используется функция **skip**. Например, пропустить первые три записи:

```
> db.users.find().skip(3)
```

MongoDB предоставляет возможности отсортировать полученный из бд набор данных с помощью функции **sort**. Передавая в эту функцию значения 1 или -1, можно указать в каком порядке сортировать: по возрастанию (1) или по убыванию (-1). Во многом эта функция по действию аналогична выражению `ORDER BY` в SQL. Например, сортировка по возрастанию по полю `name`:

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

Можно совмещать все эти функции в одной цепочке:

```
> db.users.find().sort({name: 1}).skip(3).limit(3)
```

Практическое задание 2.2:

1) Сформируйте запросы для вывода списков каких либо объектов коллекции вашей БД. Ограничьте список. Отсортируйте списки по имени.

2) Ограничьте этот список с помощью функций `findOne` и `limit`.

По умолчанию выборка содержит все поля документа, однако, в том случае, если требуется выбрать лишь конкретные поля, методам `find()` и `findOne()` можно передавать второй аргумент в виде JSON-структуры, с ключами, совпадающими с названиями столбцов и значениями 1, если поле должно попадать в выборку и 0, если его необходимо исключить из выборки.

Вывести только значения полей "age" у все документов, в которых `name=Tom`:

```
> db.users.find({name: "Tom"}, {age: 1})
```

Использование единицы в качестве параметра `{age: 1}` указывает, что запрос должен вернуть только содержание свойства `age`.

Обратная ситуация: нужно найти все параметры документа, кроме свойства `age`. В этом случае в качестве параметра указать 0:

```
> db.persons.find({name: "Tom"}, {age: 0})
```

При этом надо учитывать, что даже если мы отметим, что мы хотим получить только поле `name`, поле `_id` также будет включено в результирующую выборку. Поэтому, если не нужно видеть данное поле в выборке, то надо явным образом указать: `{"_id": 0}`

Альтернативно вместо 1 и 0 можно использовать `true` и `false`:

```
> db.users.find({name: "Tom"}, {age: true, _id: false})
```

Практическое задание 2.2.3:

3) Модифицируйте запрос 2.2.1 для вывода списков объектов, исключив из результата какую-либо информацию и поле.

Если нужно отсортировать ограниченную коллекцию, то можно воспользоваться параметром `$natural`. Этот параметр позволяет задать сортировку: документы передаются в том порядке, в каком они были добавлены в коллекцию, либо в обратном порядке.

Например, отобразить последние пять документов:

```
> db.users.find().sort({ $natural: -1 }).limit(5)
```

Практическое задание 2.2.4:

4) Вывести список объектов в коллекции в обратном порядке добавления.

Для работы с массивами используется оператор `$slice`. Он является в некотором роде комбинацией функций `limit` и `skip`.

Оператор `$slice` принимает два параметра. Первый параметр указывает на общее количество возвращаемых документов. Второй параметр необязательный, но если он используется, тогда первый параметр указывает на смещение относительно начала (как функция `skip`), а второй - на ограничение количества извлекаемых документов.

Например, в каждом документе определен массив `languages` для хранения языков, на которых говорит человек. Их может быть и 1, и 2, и 3 и более. И допустим, ранее мы добавили следующий объект:

```
> db.users.insert({"name": "Tom", "age": "32", "languages": ["english", "german"]})
```

Если необходимо при выводе документов сделать так, чтобы в выборку попадал только один язык из массива `languages`, а не весь массив, использовать запрос:

```
> db.users.find ({name: "Tom"}, {languages: {$slice : 1}})
```

Данный запрос при извлечении документа оставит в результате только первый язык из массива `languages`, то есть в данном случае `english`.

Обратная ситуация: нужно оставить в массиве также один элемент, но не с начала, а с конца. В этом случае необходимо передать в параметр отрицательное значение:

```
> db.users.find ({name: "Tom"}, {languages: {$slice : -1}});
```

Теперь в массиве окажется `german`, так как он первый с конца в добавленном элементе.

Использовать сразу два параметра:

```
> db.users.find ({name: "Tom"}, {languages: {$slice : [-1, 1]}});
```

Первый параметр говорит пропустить один элемент с конца (так как отрицательное значение), а второй параметр указывает на количество возвращаемых элементов массива. В итоге в массиве `language` окажется `"german"`.

В качестве селектора могут выступать не только строки, но и регулярные выражения, Например, найти все документы, в которых значение ключа `name` начинается с буквы `T`:

```
> db.users.find({name:/T\w+/i})
```

2.3 ЛОГИЧЕСКИЕ ОПЕРАТОРЫ

В таблице ниже представлены логические операторы сравнения и логические операторы MongoDB.

Оператор SQL	MongoDB	Описание
<	\$lt	меньше
<=	\$lte	меньше или равно
>	\$gt	больше
>=	\$gte	больше или равно
<>	\$ne	не равно
NOT	\$not	отрицание
EXISTS	\$exists	проверка существования поля
OR	\$or	или
NOT OR	\$nor	не или
RLIKE, REGEXP	\$regex	соответствие регулярному выражению
LIKE	\$elemMatch	соответствие всех полей вложенного документа
-	\$size	соответствие размеру массива
-	\$type	соответствие, если поле имеет указанный тип
IN	\$in	входит в список
NOT IN	\$nin	не входит в список
ALL	\$all	одновременное совпадение набора элементов

Операторы из таблицы выше можно использовать как по отдельности, так и в комбинации.

Например, найти все документы, у которых значение ключа age меньше 30:

```
> db.users.find ({age: {$lt : 30}})
```

Аналогично будет использование других операторов сравнения. Например, тот же ключ, только больше 30:

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

Внимание: сравнение здесь проводится над целочисленными типами, а не строками. Если ключ `age` представляет строковые значения, то надо соответственно проводить сравнение над строками: `db.users.find ({age: {$gt : "30"}})`, однако результат будет тем же.

Если нужно найти все объекты со значением поля `age` больше 30, но меньше 50, можно скомбинировать два оператора:

```
> db.users.find ({age: {$gt : 30, $lt: 50}})
```

Практическое задание 2.3:

5) Вывести список объектов коллекции вашей БД и использованием логических операторов, исключив вывод идентификатора.

Оператор `$ne` извлекает все документы, **не** соответствующие некоторому условию:

```
> db.users.find ({age: {$ne : 22}})
```

Оператор `$in` определяет массив возможных выражений и ищет те ключи, значение которых имеется в массиве:

```
> db.users.find ({age: {$in : [22, 32]}})
```

Противоположным образом действует оператор `$nin`: он определяет массив возможных выражений и ищет те ключи, значение которых отсутствует в этом массиве:

```
> db.users.find ({age: {$nin : [22, 32]}})
```

Оператор `$all` похож на `$in`: он также определяет массив возможных выражений, но требует, чтобы документы имели весь определяемый набор выражений. Например, следующий запрос не вернет нам ни одного результата:

```
> db.users.find ({age: {$all : [22, 32]}})
```

Так как в документе, который представляет человека, может быть только одно значение поля `age` (не может же быть у человека два возраста), то естественно ни в одном документе не будет найдено сразу 22 и 32. Если мы сократим до одного элемента массива, тогда можем получить результат:

```
> db.users.find ({age: {$all : [22]}})
```

Другая ситуация: человек может знать несколько языков, которые присваиваются ключу `languages` в форме массива. Тогда, например, чтобы найти всех людей, говорящих одновременно и по-английски, и по-французски, можно использовать следующее выражение:

```
> db.users.find ({languages: {$all : ["english", "french"]}})
```

Оператор `$or` определяет набор пар ключ-значение, которые должны иметься в документе. И если документ имеет хоть одну такую пару ключ-значение, то он соответствует данному запросу и извлекается из бд:

```
> db.users.find ({ $or : [{name: "Tom"}, {age: 22}]})
```

Это выражение вернет все документы, в которых либо `name=Tom`, либо `age=22`.

Другой пример вернет все документы, в которых `name=Tom`, а `age` равно либо 22, либо среди значений `languages` есть "german":

```
> db.users.find ({name: "Tom", $or : [{age: 22}, {languages: "german"}]})
```

Оператор `$exists` позволяет извлечь только те документы, в которых определенный ключ присутствует или отсутствует. Например, вернуть все документы, в которых есть ключ `company`:

```
> db.users.find ({company: {$exists:true}})
```

Если указать у оператора `$exists` в качестве параметра `false`, то запрос вернет только те документы, в которых не определен ключ `company`.

Оператор `$regex` задает регулярное выражение, которому должно соответствовать значение поля. Например, пусть поле `name` обязательно имеет букву "b":

```
> db.users.find ({name: {$regex:"b"}})
```

Важно понимать, что `$regex` принимает не просто строки, а именно регулярные выражения, например: `name: {$regex:"om$"} - значение name должно оканчиваться на "om".`

Оператор `$slice` является в некотором роде комбинацией функций `limit` и `skip`. Но в отличие от них `$slice` может работать с массивами.

Оператор `$slice` принимает два параметра. Первый параметр указывает на общее количество возвращаемых документов. Второй параметр необязательный, но если он используется, тогда первый параметр указывает на смещение относительно начала (как функция `skip`), а второй - на ограничение количества извлекаемых документов.

Например, в каждом документе определен массив `languages` для хранения языков, на которых говорит человек. Их может быть и 1, и 2, и 3 и более. И допустим, ранее добавили следующий объект:

```
> db.users.insert({"name": "Tom", "age": "32", "languages": ["english", "german"]})
```

Если нужно выводе документов сделать так, чтобы в выборку попадал только один язык из массива `languages`, а не весь массив, то:

```
1 > db.users.find ({name: "Tom"}, {languages: {$slice : 1}})
```

Данный запрос при извлечении документа оставит в результате только первый язык из массива `languages`, то есть в данном случае `english`.

Обратная ситуация: нужно оставить в массиве также один элемент, но не с начала, а с конца. В этом случае необходимо передать в параметр отрицательное значение:

```
1 > db.users.find ({name: "Tom"}, {languages: {$slice : -1}});
```

Теперь в массиве окажется `german`, так как он первый с конца в добавленном элементе.

Использовать сразу два параметра:

```
1 > db.users.find ({name: "Tom"}, {languages: {$slice : [-1, 1]}});
```

Первый параметр говорит пропустить один элемент с конца (так как отрицательное значение), а второй параметр указывает на количество возвращаемых элементов массива. В итоге в массиве `language` окажется `"german"`.

Практическое задание 2.3:

б) Вывести список упорядоченный список имен объектов с информацией из коллекции вашей БД.

Список источников:

1. MongoDB CRUD Operations [Электронный ресурс] // mongoDB. Documentation: официальный сайт MongoDB. URL: <https://docs.mongodb.com/manual/crud/> (дата обращения: 02.05.2021).
2. MongoDB – Краткое руководство [Электронный ресурс] // CoderLessons.com. Уроки по программированию, DevOps и другим IT-технологиям: сайт, 2021. URL: <https://coderlessons.com/tutorials/bazy-dannykh/uchitsia-mongodb/mongodb-kratkoe-rukovodstvo> (дата обращения: 02.05.2021).
3. Кайл Б. MongoDB в действии [Электронный ресурс] // Доступ в ЭБС «Лань». Режим доступа: <https://e.lanbook.com/book/4156> (дата обращения: 05.05.2021).
4. Онлайн-руководство по MongoDB [Электронный ресурс] // METANIT.COM. Сайт о программировании. URL: <https://metanit.com/nosql/mongodb/> (дата обращения: 05.05.2021).

ЗАПРОСЫ К БАЗЕ ДАННЫХ MONGODB. ВЫБОРКА ДАННЫХ. ВЛОЖЕННЫЕ ОБЪЕКТЫ. ИСПОЛЬЗОВАНИЕ КУРСОРОВ. АГРЕГИРОВАННЫЕ ЗАПРОСЫ. ИЗМЕНЕНИЕ ДАННЫХ

2.4 ЗАПРОС К ВЛОЖЕННЫМ ОБЪЕКТАМ

Рассмотренные ранее запросы применялись к простым документам. Документы могут иметь сложную структуру и содержать вложенные документы.

Пусть коллекцию users добавлен следующий документ:

```
> db.users.insert({"name": "Alex", "age": 28, "company": {"name": "microsoft", "country": "USA"}})
```

Примечание. Содержание коллекции users:

```
> db.users.insert({"name": "Tom", "age": 28, "languages": ["english", "spanish"]})
> db.users.insert({"name": "Bill", "age": 32, "languages": ["english", "french"]})
> db.users.insert({"name": "Tom", "age": 32, "languages": ["english", "german"]})
```

Здесь определяется вложенный объект с ключом company. Чтобы найти все документы, у которых в ключе company вложенное свойство name=microsoft, нужно использовать оператор точку:

```
> db.users.find({"company.name": "microsoft"})
```

2.5 ИСПОЛЬЗОВАНИЕ JAVASCRIPT

MongoDB предоставляет возможность создавать запросы, используя язык JavaScript. Например, создать запрос, возвращающий те документы, в которых name=Tom. Для этого сначала объявляется функция:

```
> fn = function() { return this.name=="Tom"; }
> db.users.find(fn)
```

Этот запрос эквивалентен следующему:

```
> db.users.find("this.name=='Tom'")
```

Собственно только запросами область применения JavaScript в консоли mongo не

ограничена. Например, можно создать какую-нибудь функцию и применять ее:

```
> function sqrt(n) { return n*n; }
> sqrt(5)
25
```

2.6 КУРСОРЫ

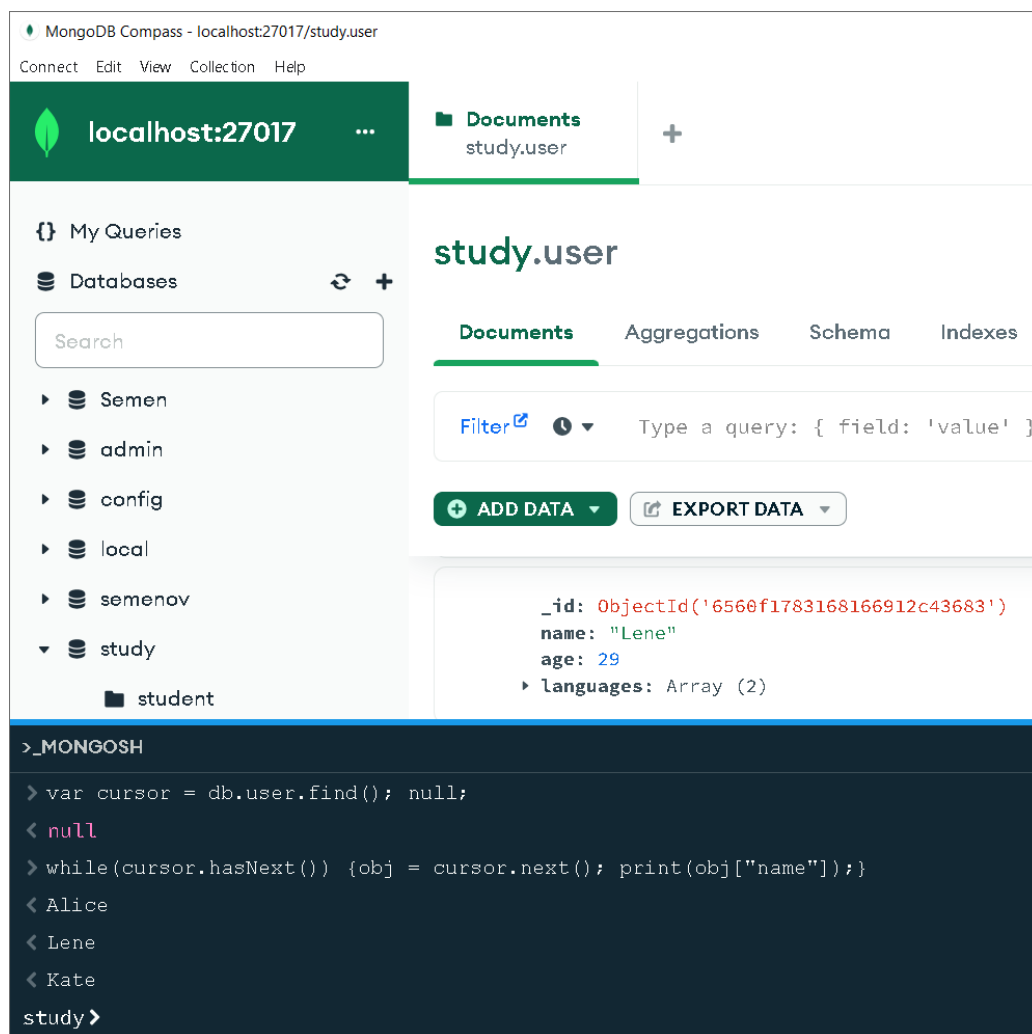
Результат выборки, получаемой с помощью функции `find`, называется курсором:

```
> var cursor = db.users.find(); null;
```

Чтобы получить курсор и сразу же не выводить все содержащиеся в нем данные, после метода `find()` через точку с запятой нужно задать выражение `null;`.

Курсоры инкапсулируют в себе наборы, получаемых из базы объектов. Используя синтаксис языка `javascript` и методы курсоров, можно вывести полученные документы на экран и обработать их. Например:

```
> var cursor = db.users.find();null;
> while(cursor.hasNext()){
...  obj = cursor.next();
...  print(obj["name"]);
... }
```



Курсор обладает методом `hasNext`, который показывает при переборе, имеется ли еще в наборе документ. Метод `next` извлекает текущий документ и перемещает курсор к следующему документу в наборе. В итоге в переменной `obj` оказывается документ, к полям которого можно получить доступ.

Для перебора документов в курсоре в качестве альтернативы можно использовать конструкцию итератора javascript - `forEach`:

```
> var cursor = db.users.find()
> cursor.forEach(function(obj) {
...   print(obj.name);
... })
```

Чтобы ограничить размер выборки, используется метод `limit`, принимающий количество документов:

```
> var cursor = db.users.find();null;
> cursor.limit(3);null;
> cursor.forEach(function(obj) {
...   print(obj.name);
... })
```

Используя метод `sort()`, можно отсортировать документы в курсоре:

```
> var cursor = db.users.find();null;
null
> cursor.sort({name:1});null;
null
> cursor.forEach(function(obj) {
...   print(obj.name);
... })
```

Выражение `cursor.sort({name:1})` сортирует документы в курсоре по полю `name` по возрастанию. Если нужно отсортировать по убыванию, то вместо `1` использовать `-1`: `cursor.sort({name:-1})`

Метод `skip()` позволяет пропустить при выборке определенное количество документов:

```
> var cursor = db.users.find();null;
> cursor.skip(2);null;
> cursor.forEach(function(obj) {
...   print(obj.name);
... })
```

В данном случае пропускается два документа.

Можно объединять все эти методы в цепочки:

```
> var cursor = db.users.find();null;
> cursor.sort({name:1}).limit(3).skip(2);null;
> cursor.forEach(function(obj) {
...   print(obj.name);
... })
```

Практическое задание 2.6:

7) Сформировать курсор для вывода списка каких-либо объектов вашего варианта и вывести этот список

2.7 АГРЕГИРОВАННЫЕ ЗАПРОСЫ

С помощью функции **count()** можно получить число элементов в коллекции:

```
> db.users.count()
```

Эта функция является самым простым агрегатом.

Можно группировать параметры поиска и функцию **count**, чтобы подсчитать количество документов, удовлетворяющих критерию, например, у которых **name=Alex**:

```
> db.users.find({name: "Alex"}).count()
```

Можно создавать цепочки функций, чтобы конкретизировать условия подсчета:

```
> db.users.find({name: "Tom"}).skip(2).count(true)
```

Здесь нужно отметить, что по умолчанию функция **count** не используется с функциями **limit** и **skip**. Чтобы их использовать, как в примере выше, в функцию **count** надо передать булево значение **true**.

Практическое задание 2.7.1:

8) . Вывести количество объектов вашей БД при определенном условии.

Функция **distinct()** позволяет найти уникальные различающиеся значения для одного или нескольких полей документа.

Например, в нескольких документах определено **name: "Tom"**. Нужно найти только уникальные различающиеся значения для одного из полей документа. Для этого можно воспользоваться функцией **distinct**:

```
> db.users.distinct("name")
["Tom", "Bill", "Alex"]
```

Использование метода **aggregate** аналогично применению выражения **GROUP BY** в SQL. Метод **group** принимает три параметра:

- **\$group**: агрегатор, который вернет новый документ
- **_id**: указывает на ключ, по которому надо проводить группировку (\$+название поля)
- **\$sum**: оператор для вычисления.

Например:

```
> db.users.aggregate({"$group":{"_id":"$name",count:{$sum:1}}})
```

Пояснение. Параметр **_id** указывает, что группировка будет проводиться по ключу **name: _id:"\$name"**.

Значение параметра **\$sum** инициализирует начальное значение поля **count**. Это поле будет представлять количество элементов для группы. На рисунке 1 приведен пример подсчет количества документов с разными именами **name**.

```
> db.users.aggregate({"$group":{"_id":"$name",count:{$sum:1}}})
{ "_id" : "Tom", "count" : 3 }
{ "_id" : "Kate", "count" : 1 }
```

Рисунок 1 – Пример использования метода aggregate

(<https://docs.mongodb.com/manual/aggregation/>).

9) Сформировать запрос на подсчет количества документов с разными именами name для коллекции вашей БД (Использование метода aggregate).

2.8 РЕДАКТИРОВАНИЕ ДАННЫХ

СУБД MongoDB предоставляет несколько возможностей для обновления данных.

Метод **save** является наиболее простым. В качестве фактического параметра метод принимает документ. В этот документ в качестве поля можно передать параметр `_id`. Если метод находит документ с таким значением `_id`, то документ обновляется. Если же с подобным `_id` нет документов, то документ вставляется.

Если параметр `_id` не указан, то документ вставляется, а параметр `_id` генерируется автоматически как при обычном добавлении через функцию `insert`:

```
> db.users.save({name: "Eugene", age : 29,
  languages: ["english", "german", "spanish"]})
```

В качестве результата функция возвращает объект `WriteResult`. Например, при успешном сохранении получится результат:

Функция принимает три параметра:

- `query`: принимает запрос на выборку документа, который нужно обновить;
- `objNew`: представляет документ с новой информацией, который заместит старый при обновлении;
- `options`: определяет дополнительные параметры при обновлении документов и может принимать два аргумента: `upsert` и `multi`.

Если параметр `upsert` имеет значение `true`, что `mongodb` будет обновлять документ, если он найден, и создавать новый, если такого документа нет. Если же он имеет значение `false`, то `mongodb` не будет создавать новый документ, если запрос на выборку не найдет ни одного документа.

Параметр `multi` указывает, должен ли обновляться первый элемент в выборке (используется по умолчанию, если данный параметр не указан) или же должны обновляться все документы в выборке.

Например:

```
> db.users.update({name : "Tom"}, {name: "Tom", age : 25, married : false},
  {upsert: true})
```

Теперь документ, найденный запросом `{name : "Tom"}`, будет перезаписан документом `{"name": "Tom", "age" : "25", "married" : false}`.

Функция `update()` также возвращает объект `WriteResult`. Например:

```
WriteResult({"nMatched" : 1, "nUpserted": 0, "nModified": 1})
```

В данном случае результат говорит о том, что найден один документ, удовлетворяющий условию, и один документ был обновлен.

Для обновления значения только одного из ключей используется оператор **\$set**.

Если документ не содержит обновляемое поле, то оно создается.

```
> db.users.update({name : "Eugene", age: 29}, {$set: {age : 30}})
```

В данном случае обновлялся только один документ, первый в выборке. Указав значение `multi:true`, можно обновить все документы выборки:

```
> db.users.update({name : "Tom"},  
  {$set: {name: "Tom", age : 25, married : false}}, {multi:true})
```

Для простого увеличения значения числового поля на определенное количество единиц применяется оператор **\$inc**. Если документ не содержит обновляемое поле, то оно создается. Данный оператор применим только к числовым значениям.

```
> db.users.update({name : "Tom"}, {$inc: {salary:100}})
```

Для удаления отдельного ключа используется оператор **\$unset**:

```
> db.users.update({name : "Tom"}, {$unset: {salary: 1}})
```

Если подобного ключа в документе не существует, то оператор не оказывает никакого влияния.

Можно удалять сразу несколько ключей:

```
> db.users.update({name : "Tom"}, {$unset: {salary: 1, age: 1}})
```

Оператор **\$push** позволяет **добавить еще одно значение к уже существующему**. Например, если ключ в качестве значения хранит массив:

```
> db.users.update({name : "Tom"}, {$push: {languages: "russian"}})
```

Если ключ, для которого нужно добавить значение, не представляет массив, то получится ошибка `Cannot apply $push/$pushAll modifier to non-array`.

Оператор **\$addToSet** подобно оператору **\$push** добавляет объекты в массив. Отличие состоит в том, что **\$addToSet** добавляет данные, если их еще нет в массиве:

```
> db.users.update({name : "Tom"}, {$addToSet: {languages: "russian"}})
```

Используя оператор **\$each**, можно **добавить сразу несколько значений**:

```
> db.users.update({name : "Tom"},  
  {$addToSet: {languages: {$each: ["russian", "spanish", "italian"]}}})
```

Оператор **\$pop** позволяет **удалять элемент из массива**:

```
> db.users.update({name : "Tom"}, {$pop: {languages: 1}})
```

Указывая для ключа `languages` значение `1`, можно удалить первый элемент с конца. Чтобы удалить первый элемент с начала массива, нужно передать отрицательное значение:

```
> db.users.update({name : "Tom"}, {$pop: {languages: -1}})
```

Оператор `$pull` удаляет каждое вхождение элемента в массив. Например, через оператор `$push` можно добавить одно и то же значение в массив несколько раз. Далее с помощью `$pull` удалить его:

```
> db.users.update({name : "Tom"}, {$pull: {languages: "english"}})
```

Если нужно удалить не одно значение, а сразу несколько, тогда можно применить оператор `$pullAll`:

```
> db.users.update({name : "Tom"},
{$pullAll: {languages: ["english", "german", "french"]}})
```

2.9 УДАЛЕНИЕ ДАННЫХ ИЗ КОЛЛЕКЦИИ

Для удаления документов в MongoDB предусмотрен метод `remove`:

```
> db.users.remove({name : "Tom"})
```

Метод `remove()` возвращает объект `WriteResult`. При успешном удалении одного документа результат будет следующим:

```
WriteResult({"nRemoved" : 1})
```

В итоге все найденные документы с `name=Tom` будут удалены. Причем, как и в случае с `find`, можно задавать условия выборки для удаления различными способами (в виде регулярных выражений, в виде условных конструкций и т.д.):

```
> db.users.remove({age: {$lt : 30}})
```

Метод `remove` также может принимать второй необязательный параметр булевого типа, который указывает, надо удалять один элемент или все элементы, соответствующие условию. Если этот параметр равен `true`, то удаляется только один элемент. По умолчанию он равен `false`:

```
> db.users.remove({name : "Tom"}, true)
```

Чтобы удалить все документы из коллекции, нужно оставить пустым параметр запроса:

```
> db.users.remove({})
```

Примечание. Содержание коллекции `users`:

```
> db.users.insert({"name": "Tom", "age": 28, languages: ["english",
"spanish"]})
```

```
> db.users.insert({"name": "Bill", "age": 32, languages: ["english",
"french"]})
```

```
> db.users.insert({"name": "Tom", "age": 32, languages: ["english",
"german"]})
```

Контрольные вопросы:

1. Как используется оператор точка?
2. Как можно использовать курсор?
3. Какие возможности агрегирования данных существуют в MongoDB?
4. Какая из функций `save` или `update` более детально позволит настроить редактирование документов коллекции?

5. Как происходит удаление документов из коллекции по умолчанию?

Список источников:

1. MongoDB CRUD Operations [Электронный ресурс] // mongoDB. Documentation: официальный сайт MongoDB. URL: <https://docs.mongodb.com/manual/crud/> (дата обращения: 02.05.2021).
2. MongoDB – Краткое руководство [Электронный ресурс] // CoderLessons.com. Уроки по программированию, DevOps и другим IT-технологиям: сайт, 2021. URL: <https://coderlessons.com/tutorials/bazy-dannykh/uchitsia-mongodb/mongodb-kratkoe-rukovodstvo> (дата обращения: 02.05.2021).
3. Кайл Б. MongoDB в действии [Электронный ресурс] // Доступ в ЭБС «Лань». Режим доступа: <https://e.lanbook.com/book/4156> (дата обращения: 05.05.2021).
4. Онлайн-руководство по MongoDB [Электронный ресурс] // METANIT.COM. Сайт о программировании. URL: <https://metanit.com/nosql/mongodb/> (дата обращения: 05.05.2021).
5. Эрик Р. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL. [Электронный ресурс]/Р. Эрик, Р.У. Джим. Электрон. дан. М.: ДМК Пресс, 2013. 384с. Доступ из ЭБС «Лань». URL: <http://e.lanbook.com/book/58690>

2.10 ССЫЛКИ В БАЗЕ ДАННЫХ MONGODB

ССЫЛКИ В БД

Ручная установка ссылок

Ручная установка ссылок сводится к присвоению значения поля `_id` одного документа полю другого документа. Пусть есть коллекции `companies` и `persons`, представляющие компании и работников, работающих в этих компаниях. В коллекции `companies` есть документ, представляющий компанию:

```
> db.companies.insert({"_id": "microsoft", year: 1974})
```

Пусть в коллекции `persons` есть документ, представляющий работника. В этом документе будет поле `company`, представляющее компанию, где работает работник. И очень важно, что в качестве значения для этого поля устанавливается не объект `company`, а значение ключа `_id` добавленного выше документа:

```
> db.users.insert({name: "Tom", age: 28, company: "microsoft"})
```

Вывести документ из коллекции `users`:

```
> user = db.users.findOne()
```

В данном случае имеется в виду, что выше добавленный элемент будет единственным в коллекции.

После этого консоль выводит полученный документ. Далее найти ссылку на его компанию в коллекции `companies`:

```
> db.companies.findOne({_id: user.company})
```

Если документ с таким идентификатором обнаружен, он отображается на консоли:

A screenshot of a MongoDB shell window titled "C:\mongodb\bin\mongo.exe". The shell version is 3.0.3. The user connects to a database named 'test'. They insert a document into the 'companies' collection with '_id' as 'microsoft' and 'year' as 1974. Then they insert a document into the 'users' collection with 'name' as 'Tom', 'age' as 28, and 'company' as 'microsoft'. They find the user document and then find the company document using the user's company name.

```
MongoDB shell version: 3.0.3
connecting to: test
> db.companies.insert<<{"_id": "microsoft", year: 1974}>>
WriteResult<< "nInserted" : 1 >>
> db.users.insert<<{name: "Tom", age: 28, company: "microsoft"}>>
WriteResult<< "nInserted" : 1 >>
> user = db.users.findOne<>
{
  "_id" : ObjectId<"556abb9aa6fc9ffd908901ff">,
  "name" : "Tom",
  "age" : 28,
  "company" : "microsoft"
}
> db.companies.findOne<<{"_id": user.company}>>
{ "_id" : "microsoft", "year" : 1974 }
>
```

Автоматическое связывание

1 способ. Используя функциональность DBRef, можно установить автоматическое связывание между документами.

Пример применение данной функциональности: добавить новый документ в коллекцию companies:

```
> apple=({"name": "apple", "year": 1976})
> db.companies.save(apple)
```

В данном случае сохранение идет с помощью метода save, не insert. Метод save при добавлении нового документа генерирует _id. После сохранения можно вывести документ на консоль: > apple

Далее создать новый документ для коллекции person, у которого ключ company свяжем с только что добавленным документом apple:

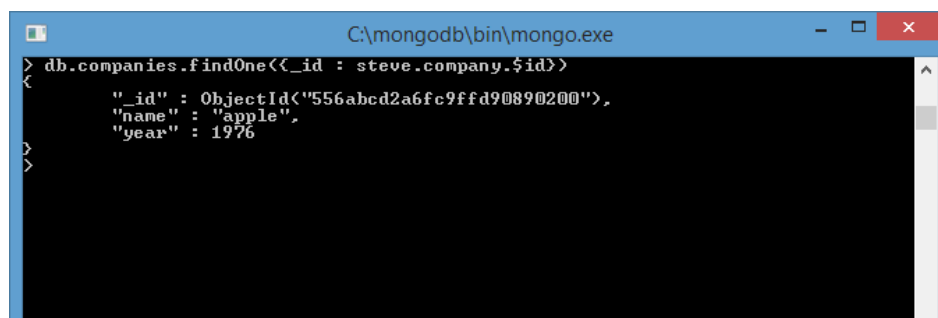
```
> steve = ({ "name": "Steve", "age": 25, company: new DBRef('companies',
apple._id) })
> db.users.save(steve)
```

A screenshot of a MongoDB shell window titled "C:\mongodb\bin\mongo.exe". The user creates an 'apple' document and saves it. Then they create a 'steve' document where the 'company' field is a DBRef pointing to the 'apple' document. Finally, they save the 'steve' document.

```
> apple=({"name": "apple", year: 1976})
{ "name" : "apple", "year" : 1976 }
> db.companies.save(apple)
WriteResult<< "nInserted" : 1 >>
> apple
{
  "name" : "apple",
  "year" : 1976,
  "_id" : ObjectId<"556abcd2a6fc9ffd90890200">
}
> steve= (<{name: "Steve", age: 25, company: new DBRef<'companies', apple._id>>>
{
  "name" : "Steve",
  "age" : 25,
  "company" : DBRef<"companies", ObjectId<"556abcd2a6fc9ffd90890200">>
}
> db.users.save(steve)
WriteResult<< "nInserted" : 1 >>
>
```

Протестировать:

```
> db.companies.findOne({_id: steve.company.$id})
```

A screenshot of a Windows command prompt window titled "C:\mongodb\bin\mongo.exe". The prompt shows a MongoDB query: > db.companies.findOne(<_id : steve.company.\$id>). The result is a JSON object: {"_id" : ObjectId("556abcd2a6fc9ffd90890200"), "name" : "apple", "year" : 1976}.

```
> db.companies.findOne(<_id : steve.company.$id>)
{
  "_id" : ObjectId("556abcd2a6fc9ffd90890200"),
  "name" : "apple",
  "year" : 1976
}
```

Посмотрев на примере, можно разобрать организацию ссылок между документами. Для связывания с документом apple использовалось следующее выражение company: new DBRef('companies', apple._id). Формальный синтаксис DBRef следующий:

```
{ "$ref" : название_коллекции, "$id": значение [, "$db" : название_бд ] }
```

При тестировании в качестве запроса на выборку указывается выражение _id: steve.company.\$id. Так как person.company представляет теперь объект new DBRef('companies', apple._id), то нужно конкретизировать параметр steve.company.\$id.

2 способ. Пусть пользователя нужно связать со страной.

Используется коллекция countries, хранящая в качестве идентификатора аббревиатуру и наименование, например:

```
> db.countries.insert({_id:"us", name:"US"})
```

Необходимо добавить в коллекцию пользователей ссылку на страну:

```
>db.users.update({_id:ObjectId("573d7468bfe2c1a9875562e6")},{ $set:
{country:{$ref:"countries", $id: "us"}}})
```

Теперь можно извлечь из коллекции данные о Томе:

```
>var tom=db.users.findOne({_id:ObjectId("573d7468bfe2c1a9875562e6")})
```

Можно также получить сведения о стране, в которой находится пользователь, опросив коллекцию стран, передав сохраненный ранее идентификатор \$id:

```
> db.countries.findOne({_id:tom.country.$id})
```

Можно непосредственно запросить у документа о пользователе имя коллекции, хранящейся в поле ссылки:

```
db[tom.country.$ref].findOne({_id:tom.country.$id})
```

Последние два запроса эквивалентны, но второй в большей степени управляется данными.

Практическое задание 2.10.:

10) Создайте новую коллекцию (например, person) для часто встречающихся названий объектов в исходной коллекции, указав в качестве идентификатора кратко название объекта, далее включив полное название и описание.

11) Включите в документы исходной коллекции ссылку на документы новой коллекции(например, person), используя второй способ автоматического связывания.

12) Выведете все данные исходной и новой коллекции.

Список источников:

1. MongoDB CRUD Operations [Электронный ресурс] // mongoDB. Documentation: официальный сайт MongoDB. URL: <https://docs.mongodb.com/manual/> (дата обращения: 02.05.2022).
2. MongoDB – Краткое руководство [Электронный ресурс] // CoderLessons.com. Уроки по программированию, DevOps и другим IT-технологиям: сайт, 2019. URL: <https://coderlessons.com/tutorials/bazy-dannykh/uchitsia-mongodb/mongodb-kratkoe-rukovodstvo> (дата обращения: 02.05.2022).
3. Кайл Б. MongoDB в действии [Электронный ресурс] // Доступ в ЭБС «Лань». Режим доступа: <https://e.lanbook.com/book/4156> (дата обращения: 05.05.2022).
4. Онлайн-руководство по MongoDB [Электронный ресурс] // METANIT.COM. Сайт о программировании. URL: <https://metanit.com/nosql/mongodb/> (дата обращения: 05.05.2022).
5. Эрик Р. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL. [Электронный ресурс]/Р. Эрик, Р.У. Джим. Электрон. дан. М.: ДМК Пресс, 2013. 384с. Доступ из ЭБС «Лань». URL: <http://e.lanbook.com/book/58690> (дата обращения: 05.05.2022).

Практическое задание

13) В приложении MongoDB Compass сделайте экспорт коллекций вашей БД в файлы *.json. Файл отчета в формате WORD и файлы коллекций в формате *.json загрузите на учебный портал в курса «Управление данными» в раздел «Сдать Лаб.8»

Структура отчета:

1. Наименование работы.
2. Цель работы.
3. Практическое задание.
4. Выполнение.

Указание: привести результаты выполнения практических заданий (номер задания, формулировка, команда, скриншот результата, вывод (при необходимости)).

5. Выводы.