

## MongoDB. Часть 3.

### Обновление данных

#### Метод save

Как и другие системы управления базами данных MongoDB предоставляет возможность обновления данных. Наиболее простым для использования является метод `save`. В качестве параметра этот 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`:

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

В качестве результата функция возвращает объект `WriteResult`. Например, при успешном сохранении мы получим:

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

#### update

Более детальную настройку при обновлении предлагает функция `update`. Она принимает три параметра:

- `query`: принимает запрос на выборку документа, который надо обновить
- `objNew`: представляет документ с новой информацией, который заместит старый при обновлении
- `options`: определяет дополнительные параметры при обновлении документов. 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:

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

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

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

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

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

### Обновление отдельного поля

Часто не требуется обновлять весь документ, а только значение одного из его ключей. Для этого применяется оператор `$set`. Если документ не содержит обновляемое поле, то оно создается.

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

Если обновляемого поля в документе нет, то оно добавляется:

```
1> db.users.update({name : "Tom", age: 29}, {$set: {salary :  
1300}})
```

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

```
1> db.users.update({name : "Tom"}, {$set: {name: "Tom", age :  
125}}, {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.

```
1> db.users.update({name : "Tom"}, {$inc: {age:2}})
```

### Удаление поля

Для удаления отдельного ключа используется оператор `$unset`:

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

Если вдруг подобного ключа в документе не существует, то оператор не оказывает никакого влияния. Также можно удалять сразу несколько полей:

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

### updateOne и updateMany

Метод `updateOne` похож на метод `update` за тем исключением, что он обновляет только один документ.

```
1> db.users.updateOne({name : "Tom", age: 29}, {$set: {salary :  
1360}})
```

Если необходимо обновить все документы, соответствующие некоторому критерию, то применяется метод `updateMany()`:

```
1> db.users.updateMany({name : "Tom"}, {$set: {salary : 560}})
```

### Обновление массивов

#### Оператор \$push

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

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

Если ключ, для которого мы хотим добавить значение, не представляет массив, то мы получим ошибку `Cannot apply $push/$pushAll modifier to non-array`.

Используя оператор `$each`, можно добавить сразу несколько значений:

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

Еще пара операторов позволяет настроить вставку. Оператор `$position` задает позицию в массиве для вставки элементов, а оператор `$slice` указывает, сколько элементов оставить в массиве после вставки.

```
1> db.users.update({name : "Tom"}, {$push: {languages: {$each:
1 ["german", "spanish", "italian"], $position:1, $slice:5}}})
```

В данном случае элементы `["german", "spanish", "italian"]` будут вставляться в массив `languages` с 1-го индекса, и после вставки, в массиве останутся только 5 первых элементов.

## Оператор `$addToSet`

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

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

## Удаление элемента из массива

Оператор `$pop` позволяет удалять элемент из массива:

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

Указывая для ключа `languages` значение 1, мы удаляем первый элемент с конца. Чтобы удалить первый элемент сначала массива, надо передать отрицательное значение:

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

Несколько иное действие предполагает оператор `$pull`. Он удаляет каждое вхождение элемента в массив. Например, через оператор `$push` мы можем добавить одно и то же значение в массив несколько раз. И теперь с помощью `$pull` удалим его:

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

А если мы хотим удалить не одно значение, а сразу несколько, тогда мы можем применить оператор `$pullAll`:

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

## Удаление данных

Для удаления документов в MongoDB предусмотрен метод `remove`:

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

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

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

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

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

```
2> 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`:

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

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

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

## Удаление коллекций и баз данных

Мы можем удалять не только документы, но и коллекции и базы данных. Для удаления коллекций используется функция `drop`:

```
1> db.users.drop()
```

И если удаление коллекции пройдет успешно, то консоль выведет:

```
true
```

Чтобы удалить всю базу данных, надо воспользоваться функцией `dropDatabase()`:

```
1> db.dropDatabase()
```



```
C:\mongodb\bin\mongo.exe
> db.users.drop()
true
> db.dropDatabase()
{ "dropped" : "test", "ok" : 1 }
>
```

## Установка ссылок в БД

в реляционных базах данных можно устанавливать внешние ключи, когда поля из одной таблицы ссылаются на поля в другой таблице. В MongoDB также можно устанавливать ссылки.

### Ручная установка ссылок

Ручная установка ссылок сводится к присвоению значения поля `_id` одного документа полю другого документа. Допустим, у нас могут быть коллекции, представляющие компании и работников, работающих в этих компаниях. Итак, сначала добавим в коллекцию `companies` документ представляющий компанию:

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

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

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

Теперь получим документ из коллекции `users`:

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

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

После этого консоль выводит полученный документ. И теперь найдем ссылку на его компанию в коллекции `companies`:

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

И если документ с таким идентификатором обнаружен, он отображается на консоли:



```
C:\mongodb\bin\mongo.exe
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

Используя функциональность **DBRef**, мы можем установить автоматическое связывание между документами. Посмотрим на примере применение данной функциональности. Вначале добавим новый документ в коллекцию `companies`:

```
1> apple=({"name": "apple", "year": 1976})
```

```
2> db.companies.save(apple)
```

Обратите внимание, что в данном случае сохранение идет с помощью метода `save`, не `insert`. Метод `save` при добавлении нового документа генерирует `_id`. И после сохранения мы можем вывести документ на консоль: `> apple`

Теперь создадим новый документ для коллекции `person`, у которого ключ `company` свяжем с только что добавленным документом `apple`:

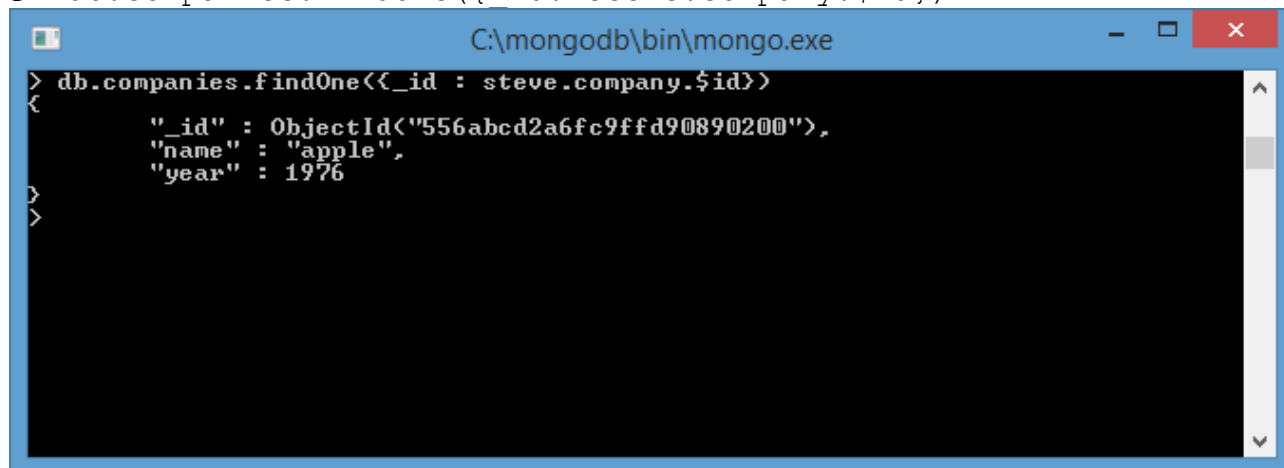
```
1> steve = ({ "name": "Steve", "age": 25, company: new
2DBRef('companies', apple._id)})
> db.users.save(steve)
```



```
C:\mongodb\bin\mongo.exe
> 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 }>
>
```

И мы можем протестировать:

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



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

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

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

Первый параметр `$ref` указывает на коллекцию, где хранится связанный документ. Второй параметр указывает на значение, которое и будет представлять что-то типа внешнего ключа. Третий необязательный параметр указывает на базу данных.

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

## Работа с индексами

При поиске документов в небольших коллекциях мы не испытаем особых проблем. Однако когда коллекции содержат миллионы документов, а нам надо сделать выборку по определенному полю, то поиск нужных данных может занять некоторое время, которое может оказаться критичным для нашей задачи. В этом случае нам могут помочь индексы.

Индексы позволяют упорядочить данные по определенному полю, что впоследствии ускорит поиск. Например, если мы в своем приложении или задаче, как правило, выполняем поиск по полю `name`, то мы можем индексировать коллекцию по этому полю:

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

Таким образом с помощью метода `createIndex` устанавливается индекс по полю `name`. MongoDB позволяет установить до 64 индексов на одну коллекцию.

## Настройка индексов

Если мы просто определим индекс для коллекции, например, `db.users.createIndex({"name" : 1})`, то мы все еще сможем добавлять в коллекцию документы с одинаковым значением ключа `name`. Однако, если нам потребуется, чтобы в коллекцию можно было добавлять документ с одним и тем же значением ключа только один раз, мы можем установить флаг `unique`:

```
> db.users.createIndex({"name" : 1}, {"unique" : true})
```

Теперь, если мы попытаемся добавить в коллекцию два документа с одним и тем же значением `name`, то мы получим ошибку.

В тоже время тут есть свои тонкости. Так, документ может не иметь ключа `name`. В этом случае для добавляемого документа автоматически создается ключ `name` со значением `null`. Поэтому при добавлении второго документа, в котором не определен ключ `name`, будет выброшено исключение, так как ключ `name` со значением `null` уже присутствует в коллекции.

Также можно задать уникальный индекс сразу для двух полей:

```
> db.users.createIndex({"name" : 1, "age" : 1}, {"unique" : true})
```

Однако в этом случае все добавляемые документы должны иметь уникальные значения для обоих полей.

Кроме того, тут есть свои ограничения. Например, значение поля, по которому идет индексация, не должно быть больше 1024 байт.

## Управление индексами

Все индексы базы данных хранятся в системной коллекции *system.indexes*. Обратившись к ней, мы можем получить все индексы и связанную с ними информацию:

```
> db.system.indexes.find()
```

Также мы можем воспользоваться методом `getIndexes` и вывести всю информацию об индексах для конкретной коллекции:

```
> db.users.getIndexes()
```

Данная команда вернет вывод наподобие следующего:

```
[
  {
    "v" : 1,
    "key" : {
      "_id" : 1
    },
    "ns" : "test.users",
    "name" : "_id_"
  },
  {
    "v" : 1,
    "key" : {
      "name" : 1
    },
    "ns" : "test.users",
    "name" : "name_1"
  }
]
```

Как мы видим, здесь для коллекции `users` (из бд `test`) определено 2 индекса: `id` и `name`. Поле `key` используется для поиска максимального и минимального значений, для различных операций, где надо применять данный индекс. Поле `name` применяется в качестве идентификатора для операций администрирования, например, для удаления индекса:

```
> db.users.dropIndex("name_1")
```

## Управление коллекцией

### Явное создание коллекции

В предыдущих темах коллекция создавалась неявно автоматически при добавлении в нее первых данных. Но мы также можем создать ее явным образом, применив метод `db.createCollection(name, options)`, где `name` - название коллекции, а `options` - необязательный объект с дополнительными настройками инициализации. Например:



```
> db.createCollection("accounts")
{"ok" : 1}
```

Таким образом, создается коллекция accounts.

### Переименование коллекции

В процессе работы, возможно, потребуется изменить название коллекции. Например, если при первом добавлении данных в ее названии была опечатка. И чтобы не удалять и затем пересоздавать коллекцию, следует использовать функцию `renameCollection`:

```
> db.users.renameCollection("новое_название")
```

И если переименование пройдет удачно, то консоль отобразит:

```
{"ok" : 1}
```

### Ограниченные коллекции

Когда мы отправляем запрос к бд на выборку, то MongoDB возвращает нам документы в том порядке, как правило, в котором они были добавлены. Однако такой порядок не всегда гарантируется, так как данные могут быть удалены, перемещены, изменены. Поэтому в MongoDB существует понятие ограниченной коллекции (capped collection). Подобная коллекция гарантирует, что документы будут располагаться в том же порядке, в котором они добавлялись в коллекцию. Ограниченные коллекции имеют фиксированный размер. И когда в коллекции уже нет места, наиболее старые документы удаляются, и в конец добавляются новые данные.

В отличие от обычных коллекций ограниченные мы можем задать явным образом. Например, создадим ограниченную коллекцию с названием `profile` и зададим для нее размер в 9500 байт:

```
> db.createCollection("profile", {capped:true, size:9500})
```

И после удачного создания коллекции консоль выведет:

```
{"ok":1}
```

Также можно ограничить количество документов в коллекции, указав его в параметре `max`:

```
> db.createCollection("profile", {capped:true, size:9500, max:150})
```

Однако при таком способе создания коллекции следует учитывать, что если все место под коллекцию заполнено (например, выделенные нами 9500 байтов), а количество документов еще не достигло максимума, в данном случае 150, то в этом случае при добавлении нового документа самый старый документ будет удаляться, а на его место будет вставляться новый документ.

При обновлении документов в таких коллекциях надо помнить, что документы не должны расти в размерах, иначе обновление не удастся произвести.

Также нельзя удалять документы из подобных коллекций, можно только удалить всю коллекцию.

### Подколлекции

Для упрощения организации данных в коллекциях мы можем использовать подколлекции. Например, данные по коллекции `users` надо разграничить на профили и учетные данные. И мы можем использовать создать коллекции `db.users.profiles` и `db.users.accounts`. При этом они не будут никак связаны с коллекцией `users`. То есть в итоге будут три разные коллекции, однако в плане логической организации хранения данных, возможно, для кого-то так будет проще.

## Задания

1. Выполнить обновление описания фотоаппарата, добавив характеристики объектива
2. Изменить во всех записях информацию о производителе с «Китай» на «КНДР»
3. Увеличить стоимость всех фотоаппаратов Nikon на 2000 р.
4. Удалить фотоаппарат по определенному критерию
5. Добавить информацию в виде массива о режимах съемки (авто, портрет...)
6. Удалить поле, описывающее производителя, у всех фотоаппаратов Nikon
7. Создать резервную копию коллекции
8. Добавить связанную коллекцию
9. Удалить коллекцию
10. Удалить базу данных