

# Введение в MongoDB.

MongoDB реализует новый подход к построению баз данных, где нет таблиц, схем, запросов SQL, внешних ключей и многих других вещей, которые присущи объектно-реляционным базам данных.

Со времен динозавров было обычным делом хранить все данные в реляционных базах данных (MS SQL, MySQL, Oracle, PostgreSQL). При этом было не столь важно, а подходят ли реляционные базы данных для хранения данного типа данных или нет.

В отличие от реляционных баз данных MongoDB предлагает документо-ориентированную модель данных, благодаря чему MongoDB работает быстрее, обладает лучшей масштабируемостью, ее легче использовать.

Но, даже учитывая все недостатки традиционных баз данных и достоинства MongoDB, важно понимать, что задачи бывают разные и методы их решения бывают разные. В какой-то ситуации MongoDB действительно улучшит производительность вашего приложения, например, если надо хранить сложные по структуре данные. В другой же ситуации лучше будет использовать традиционные реляционные базы данных. Кроме того, можно использовать смешанный подход: хранить один тип данных в MongoDB, а другой тип данных - в традиционных БД.

Вся система MongoDB может представлять не только одну базу данных, находящуюся на одном физическом сервере. Функциональность MongoDB позволяет расположить несколько баз данных на нескольких физических серверах, и эти базы данных смогут легко обмениваться данными и сохранять целостность.

## Формат данных в MongoDB

Одним из популярных стандартов обмена данными и их хранения является JSON (JavaScript Object Notation). JSON эффективно описывает сложные по структуре данные. Способ хранения данных в MongoDB в этом плане похож на JSON, хотя формально JSON не используется. Для хранения в MongoDB применяется формат, который называется **BSON** (БиСон) или сокращение от binary JSON.

BSON позволяет работать с данными быстрее: быстрее выполняется поиск и обработка. Хотя надо отметить, что BSON в отличие от хранения данных в формате JSON имеет небольшой недостаток: в целом данные в JSON-формате занимают меньше места, чем в формате BSON, с другой стороны, данный недостаток с лихвой окупается скоростью.

## Кроссплатформенность

MongoDB написана на C++, поэтому ее легко портировать на самые разные платформы. MongoDB может быть развернута на платформах Windows, Linux, MacOS, Solaris. Можно также загрузить исходный код и самому скомпилировать MongoDB, но рекомендуется использовать библиотеки с офсайта.

## Документы вместо строк

Если реляционные базы данных хранят строки, то MongoDB хранит документы. В отличие от строк документы могут хранить сложную по структуре информацию. Документ можно представить как хранилище ключей и значений.

Ключ представляет простую метку, с которым ассоциировано определенный кусок данных.

Однако при всех различиях есть одна особенность, которая сближает MongoDB и реляционные базы данных. В реляционных СУБД встречается такое понятие как первичный ключ. Это понятие описывает некий столбец, который имеет уникальные значения. В MongoDB для каждого документа имеется уникальный идентификатор, который называется `_id`. И если явным образом не указать его значение, то MongoDB автоматически сгенерирует для него значение.

Каждому ключу сопоставляется определенное значение. Но здесь также надо учитывать одну особенность: если в реляционных базах есть четко очерченная структура, где есть поля, и если какое-то поле не имеет значение, ему (в зависимости от настроек конкретной бд) можно присвоить значение `NULL`. В MongoDB все иначе. Если какому-то ключу не сопоставлено значение, то этот ключ просто опускается в документе и не употребляется.

## Коллекции

Если в традиционном мире SQL есть таблицы, то в мире MongoDB есть коллекции. И если в реляционных БД таблицы хранят однотипные жестко структурированные объекты, то в коллекции могут содержать самые разные объекты, имеющие различную структуру и различный набор свойств.

## Репликация

Система хранения данных в MongoDB представляет набор реплик. В этом наборе есть основной узел, а также может быть набор вторичных узлов. Все вторичные узлы сохраняют целостность и автоматически обновляются вместе с обновлением главного узла. И если основной узел по каким-то причинам выходит из строя, то один из вторичных узлов становится главным.

## Простота в использовании

Отсутствие жесткой схемы базы данных и в связи с этим потребности при малейшем изменении концепции хранения данных пересоздавать эту схему значительно облегчают работу с базами данных MongoDB и дальнейшим их масштабированием. Кроме того, экономится время разработчиков. Им больше не надо думать о пересоздании базы данных и тратить время на построение сложных запросов.

## GridFS

Одной из проблем при работе с любыми системами баз данных является сохранение данных большого размера. Можно сохранять данные в файлах, используя различные языки

программирования. Некоторые СУБД предлагают специальные типы данных для хранения бинарных данных в БД (например, BLOB в MySQL).

В отличие от реляционных СУБД MongoDB позволяет сохранять различные документы с различным набором данных, однако при этом размер документа ограничивается 16 мб. Но MongoDB предлагает решение - специальную технологию **GridFS**, которая позволяет хранить данные по размеру больше, чем 16 мб.

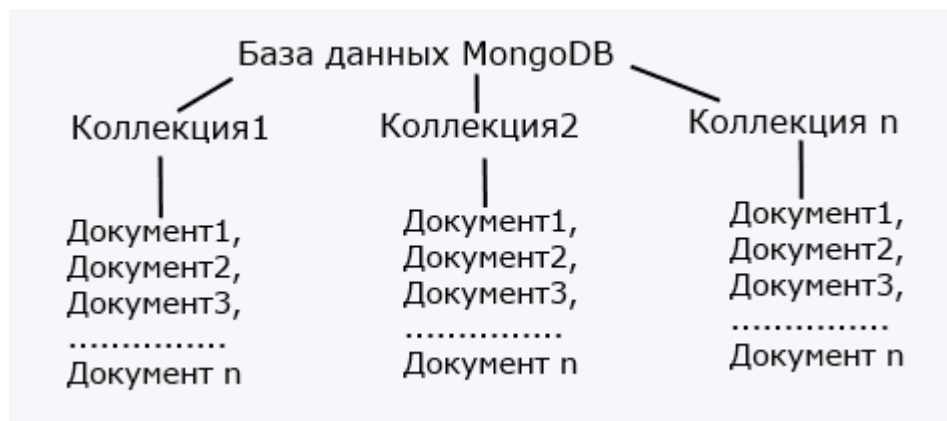
Система GridFS состоит из двух коллекций. В первой коллекции, которая называется *files*, хранятся имена файлов, а также их метаданные, например, размер. А в другой коллекции, которая называется *chunks*, в виде небольших сегментов хранятся данные файлов, обычно сегментами по 256 кб.

Для тестирования GridFS можно использовать специальную утилиту **mongofiles**, которая идет в пакете *mongodb*.

## Работа с базой данных

### Устройство базы данных. Документы

Всю модель устройства базы данных в MongoDB можно представить следующим образом:



Если в реляционных бд содержимое составляют таблицы, то в *mongodb* база данных состоит из коллекций.

Каждая коллекция имеет свое уникальное имя - произвольный идентификатор, состоящий из не более чем 128 различных алфавитно-цифровых символов и знака подчеркивания.

В отличие от реляционных баз данных MongoDB не использует табличное устройство с четко заданным количеством столбцов и типов данных. MongoDB является документо-ориентированной системой, в которой центральным понятием является документ.

Документ можно представить как объект, хранящий некоторую информацию. В некотором смысле он подобен строкам в реляционных субд, где строки хранят информацию об отдельном элементе. Например, типичный документ:

```

1 {
2   "name": "Bill",
3   "surname": "Gates",
4   "age": "48",
5   "company": {
6     "name" : "microsoft",
7     "year" : "1974",
8     "price" : "300000"
9   }
10}

```

Документ представляет набор пар ключ-значение. Например, в выражении "name": "Bill" name представляет ключ, а Bill - значение.

Ключи представляют строки. Значения же могут различаться по типу данных. В данном случае у нас почти все значения также представляют строковый тип, и лишь один ключ (company) ссылается на отдельный объект. Всего имеется следующие типы значений:

- **String:** строковый тип данных, как в приведенном выше примере (для строк используется кодировка UTF-8)
- **Array (массив):** тип данных для хранения массивов элементов
- **Binary data (двоичные данные):** тип для хранения данных в бинарном формате
- **Boolean:** булевый тип данных, хранящий логические значения TRUE или FALSE, например, {"married": FALSE}
- **Date:** хранит дату в формате времени Unix
- **Double:** числовой тип данных для хранения чисел с плавающей точкой
- **Integer:** используется для хранения целочисленных значений, например, {"age": 29}
- **JavaScript:** тип данных для хранения кода javascript
- **Min key/Max key:** используются для сравнения значений с наименьшим/наибольшим элементов BSON
- **Null:** тип данных для хранения значения Null
- **Object:** строковый тип данных, как в приведенном выше примере
- **ObjectID:** тип данных для хранения id документа
- **Regular expression:** применяется для хранения регулярных выражений
- **Symbol:** тип данных, идентичный строковому. Используется преимущественно для тех языков, в которых есть специальные символы.
- **Timestamp:** применяется для хранения времени

В отличие от строк документы могут содержать разнородную информацию. Так, рядом с документом, описанным выше, в одной коллекции может находиться другой объект, например

```
1 {  
2     "name": "Tom",  
3     "birthday": "1985.06.28",  
4     "place" : "Vashington",  
5     "languages" : [  
6         "english",  
7         "german",  
8         "spanish"  
9     ]  
10 }
```

Казалось бы разные объекты за исключением отдельных свойств, но все они могут находиться в одной коллекции.

Еще пара важных замечаний: в MongoDB запросы обладают регистрозависимостью и строгой типизацией. То есть следующие два документа не будут идентичны:

```
1 {"age" : "28"}  
2 {"age" : 28}
```

Если в первом случае для ключа age определена в качестве значения строка, то во втором случае значением является число.

## Идентификатор документа

Для каждого документа в MongoDB определен уникальный идентификатор, который называется `_id`. При добавлении документа в коллекцию данный идентификатор создается автоматически. Однако разработчик может сам явным образом задать идентификатор, а не полагаться на автоматически генерируемые, указав соответствующий ключ и его значение в документе.

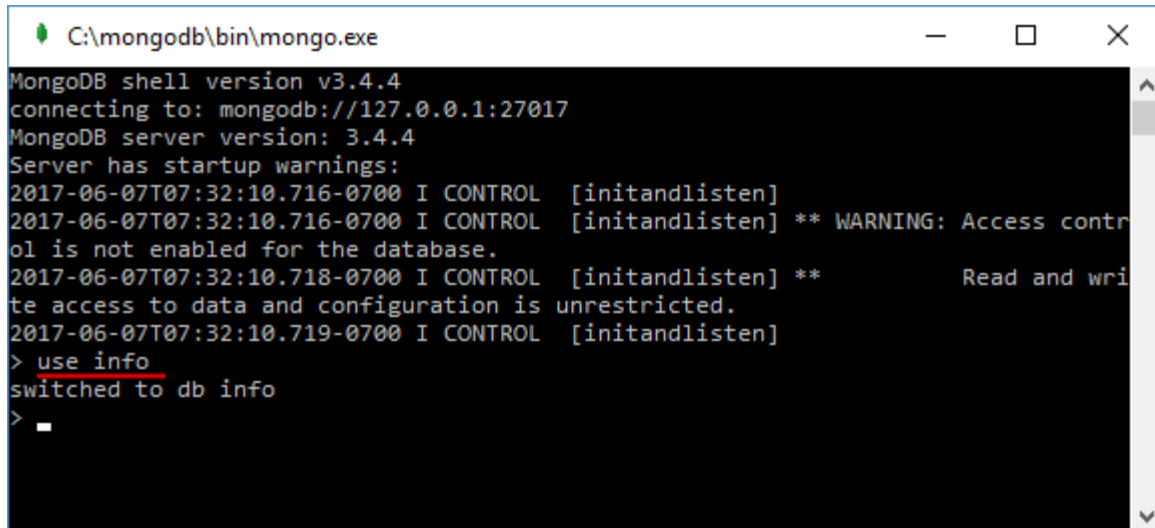
Данное поле должно иметь уникальное значение в рамках коллекции. И если мы попробуем добавить в коллекцию два документа с одинаковым идентификатором, то добавится только один из них, а при добавлении второго мы получим ошибку.

Если идентификатор не задан явно, то MongoDB создает специальное бинарное значение размером 12 байт. Это значение состоит из нескольких сегментов: значение типа `timestamp` размером 4 байта, идентификатор машины из 3 байт, идентификатор процесса из 2 байт и счетчик из 3 байт. Таким образом, первые 9 байт гарантируют уникальность среди других машин, на которых могут быть реплики базы данных. А следующие 3 байта гарантируют уникальность в течение одной секунды для одного процесса. Такая модель построения идентификатора гарантирует с высокой долей вероятности, что он будет иметь уникальное значение, ведь она позволяет создавать до 16 777 216 уникальных объектов `ObjectId` в секунду для одного процесса.

## Установка и администрирование базы данных

Начиная работать с MongoDB, первым делом надо установить нужную нам базу данных в качестве текущей, чтобы затем ее использовать. Для этого надо использовать команду `use`, после которой идет название базы данных. При этом не важно, существует ли такая бд или нет. Если ее нет, то MongoDB автоматически создаст ее при добавлении в нее данных. Например, запустим `mongo.exe` и введем там следующую команду:

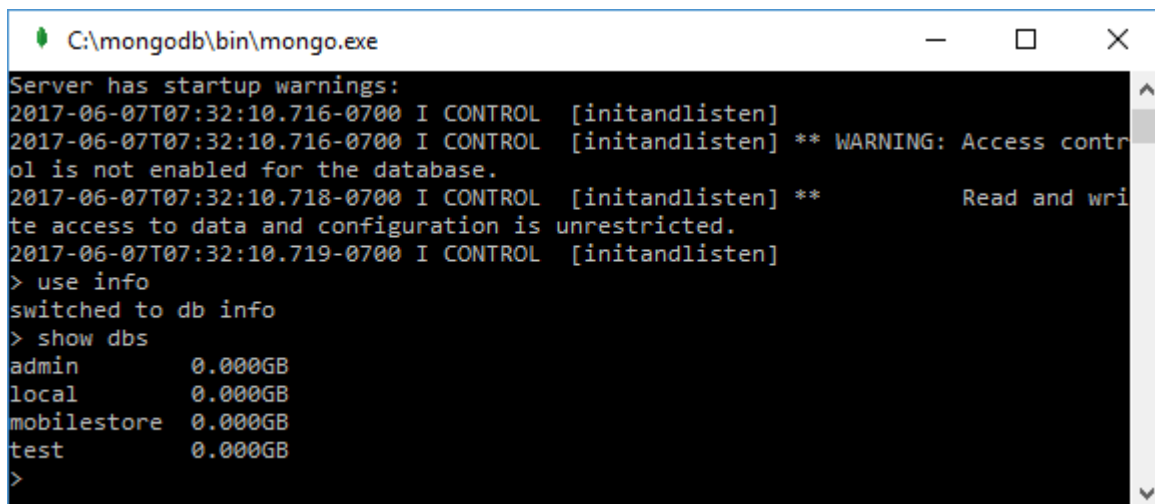
```
1> use info
```



```
C:\mongodb\bin\mongo.exe
MongoDB shell version v3.4.4
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.4
Server has startup warnings:
2017-06-07T07:32:10.716-0700 I CONTROL [initandlisten]
2017-06-07T07:32:10.716-0700 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-06-07T07:32:10.718-0700 I CONTROL [initandlisten] **           Read and write access to data and configuration is unrestricted.
2017-06-07T07:32:10.719-0700 I CONTROL [initandlisten]
> use info
switched to db info
>
```

Теперь в качестве текущей будет установлена БД `info`.

Если вы вдруг не уверены, а существует ли уже база данных с таким названием, то с помощью команды `show dbs` можно вывести названия всех имеющихся бд на консоль:



```
C:\mongodb\bin\mongo.exe
Server has startup warnings:
2017-06-07T07:32:10.716-0700 I CONTROL [initandlisten]
2017-06-07T07:32:10.716-0700 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-06-07T07:32:10.718-0700 I CONTROL [initandlisten] **           Read and write access to data and configuration is unrestricted.
2017-06-07T07:32:10.719-0700 I CONTROL [initandlisten]
> use info
switched to db info
> show dbs
admin          0.000GB
local          0.000GB
mobilestore    0.000GB
test           0.000GB
>
```

Для базы данных можно задать любое имя, однако есть некоторые ограничения. Например, в имени не должно быть символов `/`, `\`, `.`, `"`, `*`, `<`, `>`, `:`, `|`, `?`, `$`. Кроме того, имена баз данных ограничены 64 байтами.

Также есть зарезервированные имена, которые нельзя использовать: `local`, `admin`, `config`.

Причем как вы видите, бд `info` в данном списке нет, так как я в нее еще не добавил данные.

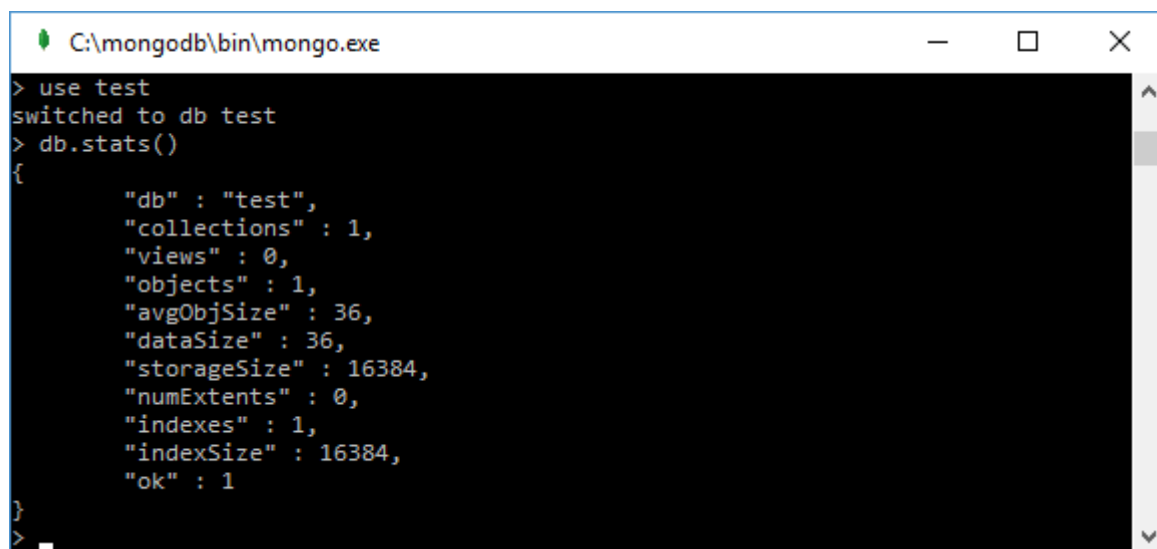
Если мы хотим узнать, какая бд используется в текущий момент, то мы можем воспользоваться командой db:

```
1> db
2info
```

Кроме баз данных мы можем посмотреть список всех коллекций в текущей бд с помощью команды `show collections`. Но так как бд `info`, которая указана текущей, еще не существует, то и коллекций она пока не содержит.

## Получение статистики

Используя команду `db.stats()`, можно получить статистику по текущей базе данных. Например, у нас в качестве текущей установлена база данных `test`:



```
C:\mongodb\bin\mongo.exe
> use test
switched to db test
> db.stats()
{
  "db" : "test",
  "collections" : 1,
  "views" : 0,
  "objects" : 1,
  "avgObjSize" : 36,
  "dataSize" : 36,
  "storageSize" : 16384,
  "numExtents" : 0,
  "indexes" : 1,
  "indexSize" : 16384,
  "ok" : 1
}
```

Похожим образом мы можем узнать всю статистику по отдельной коллекции. Например, узнаем статистику по коллекции `users`: `db.users.stats()`

## Добавление данных

Установив бд, теперь мы можем добавить в нее данные. Все данные хранятся в бд в формате BSON, который близок к JSON, поэтому нам надо также вводить данные в этом формате. И хотя у нас, возможно, на данный момент нет ни одной коллекции, но при добавлении в нее данных она автоматически создается.

Как ранее говорилось, 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). И также имя не должно содержать знака доллара - `$`.

Для добавления в коллекцию могут использоваться три ее метода:

- `insertOne()`: добавляет один документ

- `insertMany()`: добавляет несколько документов
- `insert()`: может добавлять как один, так и несколько документов

Итак, добавим один документ:

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

Документ представляет набор пар ключ-значение. В данном случае добавляемый документ имеет три ключа: `name`, `age`, `languages`, и каждому из них сопоставляет определенное значение. Например, ключу `languages` в качестве значения сопоставляется массив.

Некоторые ограничения при использовании имен ключей:

- Символ `$` не может быть первым символом в имени ключа
- Имя ключа не может содержать символ точки `.`

При добавлении данных, если мы явным образом не предоставили значение для поля `"_id"` (то есть уникального идентификатора документа), то оно генерируется автоматически. Но в принципе мы можем сами установить этот идентификатор при добавлении данных:

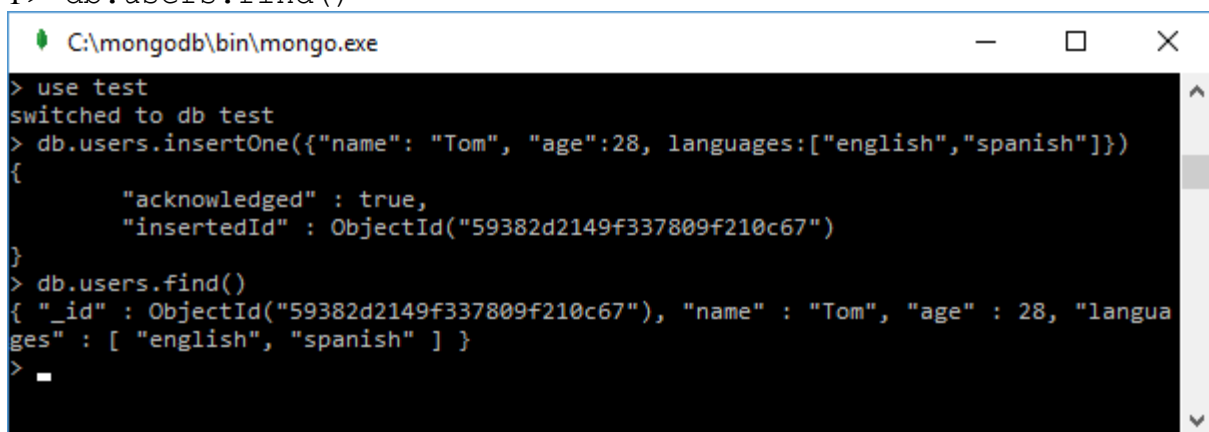
```
1> db.users.insertOne({"_id": 123457, "name": "Tom", "age": 28,
1 languages: ["english", "spanish"]})
```

Стоит отметить, что названия ключей могут использоваться в кавычках, а могут и без кавычек.

В случае удачного добавления на консоль будет выведен идентификатор добавленного документа.

И чтобы убедиться, что документ в бд, мы его выводим функцией `find`.

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



```
C:\mongodb\bin\mongo.exe
> use test
switched to db test
> db.users.insertOne({"name": "Tom", "age":28, languages:["english","spanish"]})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("59382d2149f337809f210c67")
}
> db.users.find()
{ "_id" : ObjectId("59382d2149f337809f210c67"), "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
>
```

Чтобы вывести в более читабельном виде добавим метод `pretty()`:

```
1> db.users.find().pretty()
```

Если надо добавить ряд документов, то мы можем воспользоваться методом `insertMany()`:

```
1> db.users.insertMany([{"name": "Bob", "age": 26, languages:
1 ["english", "frensh"]},
2 {"name": "Alice", "age": 31, languages:["german", "english"]}])
```

После добавления консоль выводит идентификаторы добавленных документов:



```
C:\mongodb\bin\mongo.exe
> db.users.insertMany([{"name":"Bob", "age":26, languages:["english", "french"]}
, {"name": "Alice", "age": 31, languages:["german", "english"]}])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("59383270032ed06deaffacd7"),
    ObjectId("59383270032ed06deaffacd8")
  ]
}
>
```

И третий метод - `insert()` демонстрирует более универсальный способ добавления документов. При его вызове в него также передается добавляемый документ:

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

После его вызова на консоль выводится количество добавленных записей:

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

Есть еще один способ добавления в бд документа, который включает два этапа: определение документа (`document = ( { ... } )`) и собственно его добавление:

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

При желании опять же можно с помощью функции `db.users.find()` убедиться, что документ попал в бд.

```
C:\mongodb\bin\mongo.exe
> document=({"name": "Bill", "age": 32, languages:["english", "french"]})
{ "name": "Bill", "age": 32, "languages": [ "english", "french" ] }
> db.users.insert(document)
WriteResult({ "nInserted" : 1 })
> db.users.find()
{ "_id" : ObjectId("59382d2149f337809f210c67"), "name" : "Tom", "age" : 28, "lan
guages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("59383270032ed06deaffacd7"), "name" : "Bob", "age" : 26, "lan
guages" : [ "english", "french" ] }
{ "_id" : ObjectId("59383270032ed06deaffacd8"), "name" : "Alice", "age" : 31, "l
anguages" : [ "german", "english" ] }
{ "_id" : ObjectId("5938344d032ed06deaffacd9"), "name" : "Bill", "age" : 32, "la
nguages" : [ "english", "french" ] }
>
```

Возможно, не всем будет удобно вводить в одну строку все пары ключей и свойств. Но интеллектуальный интерпретатор MongoDB на основе javascript позволяет также вводить и многострочные команды. Если выражение не закончено (с точки зрения языка JavaScript), и вы нажимаете Enter, то ввод следующей части выражения автоматически переносится на следующую строку:

A screenshot of a Windows command prompt window titled "C:\mongodb\bin\mongo.exe". The prompt shows a document being inserted into a database. The document is a JSON object with fields: "name" (Bill), "age" (32), and "languages" (an array containing "english" and "french"). The prompt shows the document being inserted and the resulting JSON representation of the document.

```
> doc=<{
...  "name": "Bill",
...  "age" : 32,
...  "languages": [
...    "english",
...    "french" ]
... }>
{ "name" : "Bill", "age" : 32, "languages" : [ "english", "french" ] }
```

## Загрузка данных из файла

Данные для базы данных mongodb можно определять в обычном текстовом файле, что довольно удобно, поскольку мы можем переносить или пересылать этот файл независимо от базы данных mongodb. Например, определим где-нибудь на жестком диске файл users.js со следующим содержанием:

```
1 db.users.insertMany([
2 {"name": "Alice", "age": 31, languages: ["english", "french"]},
3 {"name": "Lene", "age": 29, languages: ["english", "spanish"]},
4 {"name": "Kate", "age": 30, languages: ["german", "russian"]}
5 ])
```

То есть здесь с помощью метода insertMany добавляются три документа в коллекцию users.

Для загрузки файла в текущую базу данных применяется функция load(), в которую в качестве параметра передается путь к файлу:

```
1> load("D:/users.js")
```

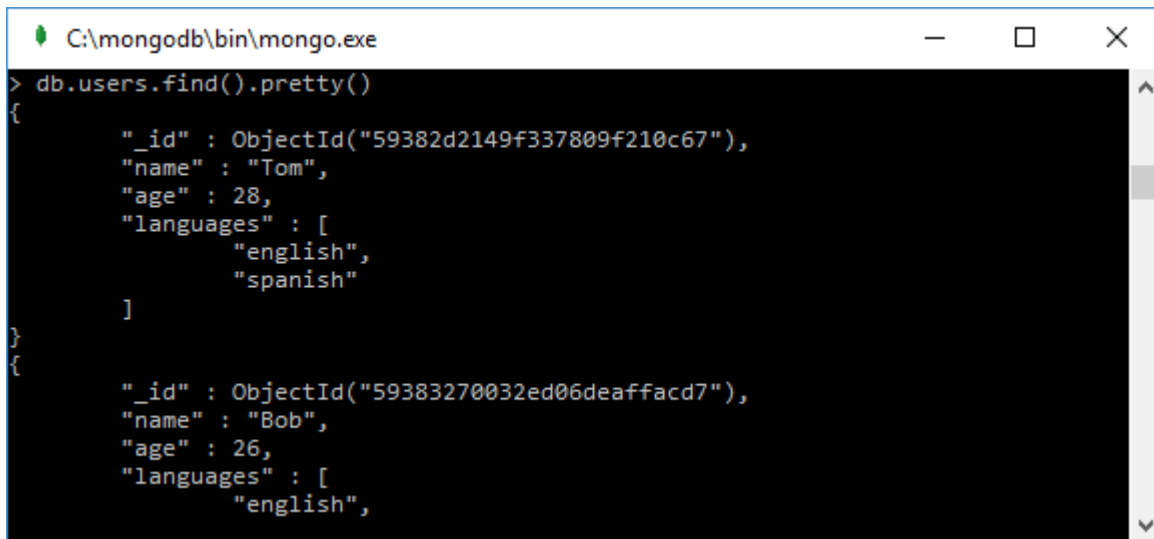
В данном случае предполагается, что файл располагается по пути "D:/users.js".

## Выборка из БД

Наиболее простой способом получения содержимого БД представляет использование функции find. Действие этой функции во многом аналогично обычному запросу SELECT \* FROM Table, который извлекает все строки. Например, чтобы извлечь все документы из коллекции users, созданной в прошлой теме, мы можем использовать команду db.users.find().

Для вывода документов в более удобном наглядном представлении мы можем добавить вызов метода pretty():

```
1> db.users.find().pretty()
```

A screenshot of a Windows command prompt window titled "C:\mongodb\bin\mongo.exe". The prompt shows the command `db.users.find().pretty()` and its output. The output displays two JSON documents. The first document has an `_id` of "59382d2149f337809f210c67", a `name` of "Tom", an `age` of 28, and a `languages` array containing "english" and "spanish". The second document has an `_id` of "59383270032ed06deaffacd7", a `name` of "Bob", an `age` of 26, and a `languages` array containing "english".

```
> db.users.find().pretty()
{
  "_id" : ObjectId("59382d2149f337809f210c67"),
  "name" : "Tom",
  "age" : 28,
  "languages" : [
    "english",
    "spanish"
  ]
}
{
  "_id" : ObjectId("59383270032ed06deaffacd7"),
  "name" : "Bob",
  "age" : 26,
  "languages" : [
    "english",
  ]
}
```

Однако что, если нам надо получить не все документы, а только те, которые удовлетворяют определенному требованию. Например, мы ранее в базу добавили следующие документы:

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

Выведем все документы, в которых `name=Tom`:

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

Такой запрос выведет нам два документа, в которых `name=Tom`.

Теперь более сложный запрос: нам надо вывести те объекты, у которых `name=Tom` и одновременно `age=32`. То есть на языке SQL это могло бы выглядеть так: `SELECT * FROM Table WHERE Name='Tom' AND Age=32`. Данному критерию у нас соответствует последний добавленный объект. Тогда мы можем написать следующий запрос:

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

Также несложно отыскать по элементу в массиве. Например, следующий запрос выводит все документы, у которых в массиве `languages` есть `english`:

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

Усложним запрос и получим те документы, у которых в массиве `languages` одновременно два языка: `"english"` и `"german"`:

```
1> db.users.find({languages: ["english", "german"]})
```

Теперь выведем все документы, в которых `"english"` в массиве `languages` находится на первом месте:

```
1> db.users.find({"languages.0": "english"})
```

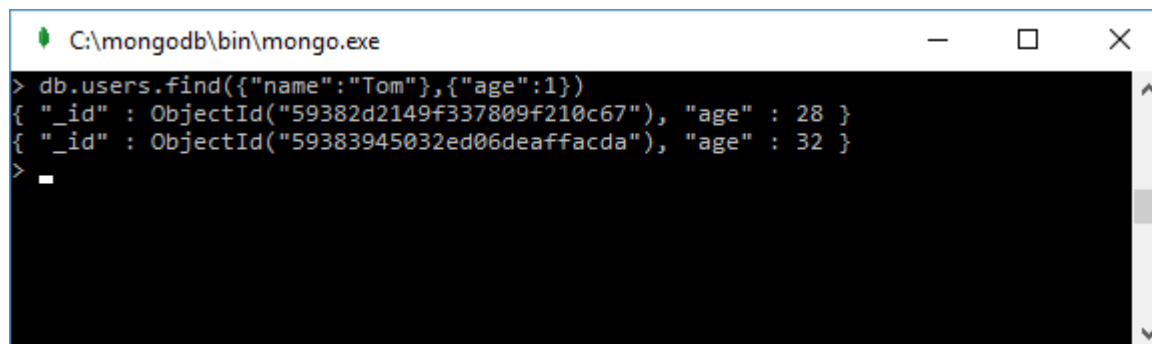
Соответственно если нам надо вывести документы, где `english` на втором месте (например, `["german", "english"]`), то вместо нуля ставим единицу: `languages.1`.

## Проекция

Документ может иметь множество полей, но не все эти поля нам могут быть нужны и важны при запросе. И в этом случае мы можем включить в выборку только нужные поля, используя проекцию. Например, выведем только порцию информации, например, значения полей "age" у всех документов, в которых name=Tom:

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

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



```
C:\mongodb\bin\mongo.exe
> db.users.find({name:"Tom"}, {"age":1})
{ "_id" : ObjectId("59382d2149f337809f210c67"), "age" : 28 }
{ "_id" : ObjectId("59383945032ed06deaffacda"), "age" : 32 }
>
```

И обратная ситуация: мы хотим найти все поля документа кроме свойства age. В этом случае в качестве параметра указываем 0:

```
1> 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 мы не хотим видеть данное поле в выборке, то надо явным образом указать: {"\_id":0}

Альтернативно вместо 1 и 0 можно использовать true и false:

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

Если мы не хотим при этом конкретизировать выборку, а хотим вывести все документы, то можно оставить первые фигурные скобки пустыми:

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

## Запрос к вложенным объектам

Предыдущие запросы применялись к простым объектам. Но документы могут быть очень сложными по структуре. Например, добавим в коллекцию persons следующий документ:

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

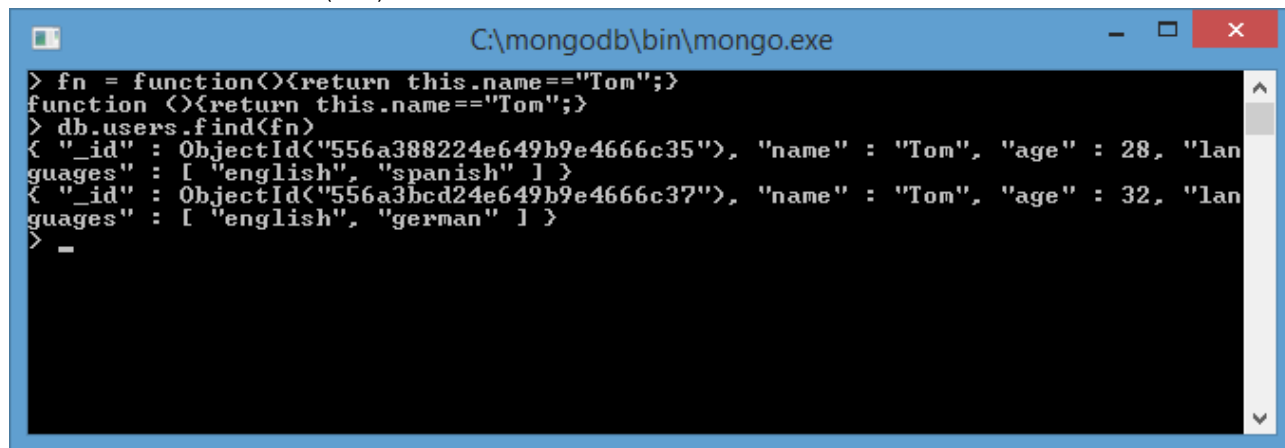
Здесь определяется вложенный объект с ключом company. И чтобы найти все документы, у которых в ключе company вложенное свойство name=microsoft, нам надо использовать оператор точку:

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

## Использование JavaScript

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

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



```
C:\mongodb\bin\mongo.exe
> fn = function() { return this.name=="Tom"; }
function () { return this.name=="Tom"; }
> db.users.find(fn)
{ "_id" : ObjectId<"556a388224e649b9e4666c35">, "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId<"556a3bcd24e649b9e4666c37">, "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }
>
```

Этот запрос эквивалентен следующему:

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

Собственно только запросами область применения JavaScript в консоли mongo не ограничена.

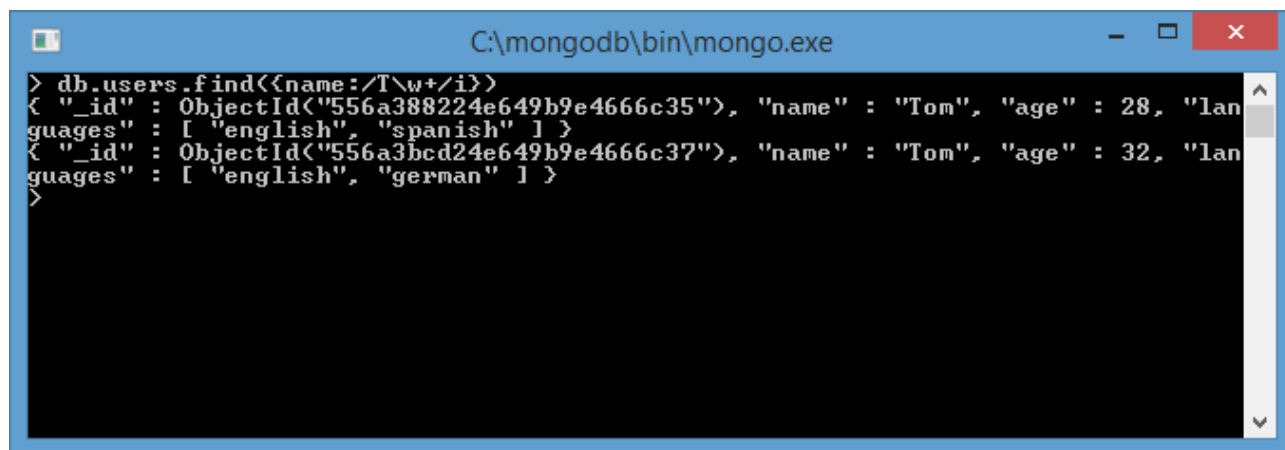
Например, мы можем создать какую-нибудь функцию и применять ее:

```
1> function sqrt(n) { return n*n; }
2> sqrt(5)
325
```

## Использование регулярных выражений

Еще одной замечательной возможностью при построении запросов является использование регулярных выражений. Например, найдем все документы, в которых значение ключа name начинается с буквы T:

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



```
C:\mongodb\bin\mongo.exe
> db.users.find({name:/T\w+/i})
{ "_id" : ObjectId<"556a388224e649b9e4666c35">, "name" : "Tom", "age" : 28, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId<"556a3bcd24e649b9e4666c37">, "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }
>
```

## Настройка запросов и сортировка

MongoDB представляет ряд функций, которые помогают управлять выборкой из бд. Одна из них - функция `limit`. Она задает максимально допустимое количество получаемых документов. Количество передается в виде числового параметра. Например, ограничим выборку тремя документами:

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

В данном случае мы получим первые три документа (если в коллекции 3 и больше документов). Но что, если мы хотим произвести выборку не сначала, а пропустив какое-то количество документов? В этом нам поможет функция `skip`. Например, пропустим первые три записи:

```
1> 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, мы можем 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`:

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

Ну и в конце надо отметить, что мы можем совмещать все эти функции в одной цепочке:

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

## Поиск одиночного документа

Если все документы извлекаются функцией `find`, то одиночный документ извлекается функцией `findOne`. Ее действие аналогично тому, как если бы мы использовали функцию `limit(1)`, которая также извлекает первый документ коллекции. А комбинация функций `skip` и `limit` извлечет документ по нужному местоположению.

## Параметр `$natural`

Если вдруг нам надо отсортировать ограниченную коллекцию, то мы можем 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, отберем последние пять документов:

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

## Оператор `$slice`

`$slice` является в некотором роде комбинацией функций `limit` и `skip`. Но в отличие от них `$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 мы добавили следующий объект:

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

И мы хотим при выводе документов сделать так, чтобы в выборку попадал только один язык из массива `languages`, а не весь массив:

```
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: нам надо 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.

Используем сразу два параметра:

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

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

## Курсоры

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

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

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

Курсоры инкапсулируют в себе наборы получаемых из бд объектов. Используя синтаксис языка `javascript` и методы курсоров, мы можем вывести полученные документы на экран и как-то их обработать. Например:

```
1> var cursor = db.users.find();null;
2> while(cursor.hasNext()) {
3... obj = cursor.next();
4... print(obj["name"]);
5... }
```

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

Также для перебора документов в курсоре в качестве альтернативы мы можем использовать конструкцию итератора `javascript` - `forEach`:

```
1> var cursor = db.users.find()
2> cursor.forEach(function(obj) {
```

```
3... print(obj.name);  
4... })
```

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

```
1> var cursor = db.users.find();null;  
2null  
3> cursor.limit(5);null;  
4null  
5> cursor.forEach(function(obj) {  
6... print(obj.name);  
7... })
```

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

```
1> var cursor = db.users.find();null;  
2null  
3> cursor.sort({name:1});null;  
4null  
5> cursor.forEach(function(obj) {  
6... print(obj.name);  
7... })
```

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

И еще один метод `skip()` позволяет пропустить при выборке определенное количество документов:

```
1> var cursor = db.users.find();null;  
2null  
3> cursor.skip(2);null;  
4null  
5> cursor.forEach(function(obj) {  
6... print(obj.name);  
7... })
```

В данном случае пропускаем два документа.

И кроме того, можно объединять все эти методы в цепочки:

```
1> var cursor = db.users.find();null;  
2null  
3> cursor.sort({name:1}).limit(3).skip(2);null;  
4null  
5> cursor.forEach(function(obj) {  
6... print(obj.name);  
7... })
```



## Задание для самоподготовки

1. Создайте базу данных интернет-магазина по продаже фотоаппаратов.
2. Добавьте в базу данных информацию о товаре. Производитель (Canon, Nikon ...), Модель, Тип камеры (Зеркальная, беззеркальная, компактная ...), Формат матрицы, Число мегапикселей, Объектив (Body или описание объектива (Фокусное расстояние, байонет ...), Максимальное разрешение записи видео, Цвет, Страна производитель, Оптический зум, Цена
3. Вывести полный список всех товаров в интернет-магазине
4. Вывести список фотоаппаратов от компании Canon
5. Вывести список зеркалок от компании Nikon
6. Вывести все фотоаппараты, у которых объектив имеет фокусное расстояние 50
7. Вывести все фотоаппараты компании Nikon, у которых объектив имеет фокусное расстояние 50 и светосилу – 1.2
8. Вывести названия моделей фотоаппаратов, позволяющих снимать видео с качеством 1920x1080
9. Вывести только наименования и цены товара
10. Вывести три зеркальных фотоаппарата с наименьшей ценой
11. Вывести информацию о последнем товаре, добавленном в базу данных