
mongoDB®

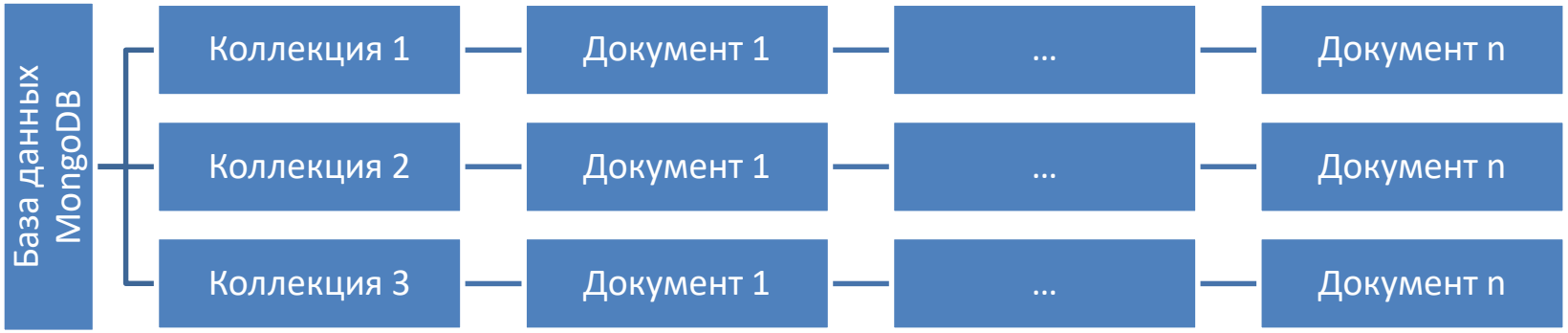
УСТАНОВКА

Ссылка на страницы скачивания community версии mongoDB с официального сайта: https://www.mongodb.com/download-center/community

Инструкция по установке: https://docs.mongodb.com/v4.0/installation/

ОПРЕДЕНИЕ

MongoDB - документно-ориентированная система управления базами данных с открытым исходным кодом, не требующая описания схемы таблиц.



Способ хранения данных в MongoDB называется BSON (БиСон) или сокращение от binary JSON.

ОПРЕДЕНИЕ

Документ можно представить как хранилище ключей и значений.

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.

Если какому-то ключу не сопоставлено значение, то этот ключ просто опускается в документе и не употребляется.

В коллекции могут содержать самые разные объекты, имеющие различную структуру и различный набор свойств.

ПРИМЕР ДОКУМЕНТА

Пример одного документа:

```
{
   "name": "Bill",
   "surname": "Gates",
   "age": "48",
   "company": {
      "name" : "microsoft",
      "year" : "1974",
      "price" : "300000"
      }
}
```

В MongoDB запросы обладают регистрозависимостью и строгой типизацией. То есть следующие два документа не будут идентичны:

```
{"age" : "28"}
{"age" : 28}
```

ТИПЫ ЗНАЧЕНИЙ В ДОКУМЕНТАХ

Всего имеется следующие типы значений:
- **String**: строковый тип данных, как в приведенном ранее 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

ОБОЛОЧКА MONGO

Команда показывает список доступных баз данных:
> show dbs

Команда выбора рабочей базы данных (даже не существующей):
> use [database_name]

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.

Команда напечатает название текущей базы данных:
> db

ОБОЛОЧКА MONGO

При добавлении данных в базу данных и коллекцию если она до этого не существовала они автоматически создадутся.

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

Примеры (добавление в коллекцию users):

```
> db.users.insertOne({"name": "Tom", "age": 28, languages:
["english", "spanish"]})
> db.users.insertMany([
{"name": "Bob", "age": 26, languages: ["english", "frensh"]},
{"name": "Alice", "age": 31, languages:["german", "english"]}])
```

ОБОЛОЧКА MONGO

Метод переименования коллекции:

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

Метод явного создания коллекции:

```
> db.createCollection("accounts")
```

Создание ограниченной коллекции (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. Также нельзя удалять документы из подобных коллекций, можно только удалить всю коллекцию.):

```
> db.createCollection("profile", {capped:true, size:9500})
//9500 байт
> db.createCollection("profile", {capped:true, size:9500, max: 150})
//150 – максимальное количество документов в коллекции
```

ОБОЛОЧКА MONGO

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.

ОБОЛОЧКА MONGO

Чтобы извлечь все документы из коллекции users, созданной в прошлой теме, мы можем использовать команду

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

Поиск с условиями:

```
> db.users.find({name: "Tom"})
> db.users.find({name: "Tom", age: 32})
> db.users.find({languages: ["english", "german"]})
> db.users.find({languages.0: "english"}) //вернет первый документ
```

ОБОЛОЧКА MONGO

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, использовав проекцию:

```
> db.users.find({name: "Tom"}, {age: 1}) //второй аргумент {age: 1} указывает, что запрос должен вернуть только содержание свойства age.
> db.persons.find({name: "Tom"}, {age: 0}) // 0 – скроет свойства из результата
```

Запрос к вложенным объектам:

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

ОБОЛОЧКА MONGO

MongoDB предоставляет замечательную возможность, создавать запросы, используя язык JavaScript:

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

Использование регулярных выражений

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

Настройка запросов и сортировка

```
> db.users.find().limit(3) //3 - количество получаемых документов
> db.users.find().skip(3) // 3 - пропустим первые три записи
> db.users.find().sort({name: 1}) //сортировка по возрастанию (1) или по убыванию (-1)
> db.users.find().sort({name: 1}).skip(3).limit(3)
```

ОБОЛОЧКА MONGO

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

```
> var cursor = db.users.find(); null; // null – чтобы не выводить сразу содержимое курсора
```

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:

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

ОБОЛОЧКА MONGO

Альтернатива перебору курсора:

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

Число элементов в коллекции

```
> db.users.count()
> db.users.find({name: "Tom"}).count()
> db.users.find({name: "Tom"}).skip(2).count(true) //учёт влияние метода skip или limit
```

Получение уникальных различающихся значений для одного из полей документа:

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

ОБОЛОЧКА MONGO

Группировка:

```
> db.users.group ({key: {name : true}, initial: {total : 0},
reduce : function (items,prev){prev.total += 1}})
// key: указывает на ключ, по которому надо проводить группировку
// initial: представляет базовое значение для сгруппированного результата
//reduce: представляет функцию, возвращающую количество элементов. Эта функция принимает в качестве аргументов два параметра: items и prev
```

ОБОЛОЧКА MONGO

В MongoDB в запросах можно использовать условные конструкции с помощью операторов сравнения:

```
//$eq (равно); $gt (больше чем); $lt (меньше чем); $gte (больше или равно); $lte (меньше или равно); $ne (не)
> db.users.find ({age: {$lt : 30}})
> db.users.find ({age: {$gt : 30}})
> db.users.find ({age: {$gt : 30, $lt: 50}})
> db.users.find ({age: {$ne : 22}})
> db.users.find ({age: {$eq : 22}})
```

ОБОЛОЧКА MONGO

Поиск по массивам:

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

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

Оператор $nin - 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:

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

ОБОЛОЧКА MONGO

Оператор $elemMatch позволяет выбрать документы, в которых массивы содержат элементы, попадающие под определенные условия:

```
> db.grades.find({courses: {$elemMatch: {name: "MongoDB", grade: {$gt: 3}}}})
```

Оператор $size используется для нахождения документов, в которых массивы имеют число элементов, равным значению $size:

```
> db.users.find ({languages: {$size:2}})
```

Оператор $exists позволяет извлечь только те документы, в которых определенный ключ присутствует или отсутствует:

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

ОБОЛОЧКА MONGO

Оператор $type извлекает только те документы, в которых определенный ключ имеет значение определенного типа:

```
> db.users.find ({age: {$type:"string"}})
> db.users.find ({age: {$type:"number"}})
```

Оператор $regex задает регулярное выражение, которому должно соответствовать значение поля:

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

Оператор $or представляет логическую операцию ИЛИ и определяет набор пар ключ-значение, из которые кто-то должен иметься в документе:

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

Оператор $and представляет логическую операцию И (логическое умножение) и определяет набор критериев, которым обязательно должен соответствовать документ:

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

ОБОЛОЧКА MONGO

MongoDB предоставляет возможность обновления данных:

```
> db.users.save({_id:123, name: "Eugene", age : 29, languages: ["english", "german", "spanish"]})
// 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.
```

ОБОЛОЧКА MONGO

Метод update: 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.
//upsert = true, то mongodb будет обновлять документ, если он найден, и создавать новый, если такого документа нет. Если = 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.

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

ОБОЛОЧКА MONGO

Обновление отдельного поля (оператор $set):

```
> db.users.update({name : "Tom", age: 29}, {$set: {age : 30}})
// Если обновляемого поля в документе нет, то оно добавляется
```

Для простого увеличения значения числового поля на определенное количество единиц применяется оператор $inc:

```
> db.users.update({name : "Tom"}, {$inc: {age:2}})
// Если обновляемого поля в документе нет, то оно добавляется
```

Для удаления отдельного ключа используется оператор $unset:

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

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

```
> db.users.updateOne({name : "Tom", age: 29}, {$set: {salary : 360}})
```

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

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

ОБОЛОЧКА MONGO

Оператор $push позволяет добавить еще одно значение к уже существующему массиву:

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

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

Оператор $position задает позицию в массиве для вставки элементов, а оператор $slice указывает, сколько элементов оставить в массиве после вставки.

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

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"}})
```

ОБОЛОЧКА MONGO

Оператор $pop позволяет удалять элемент из массива:

```
> db.users.update({name : "Tom"}, {$pop: {languages: 1}})
//значение 1, мы удаляем первый элемент с конца; -1 – мы удалим первый элемент
```

Оператор $pull. Он удаляет каждое вхождение элемента в массив

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

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

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

ОБОЛОЧКА MONGO

Для удаления документов в MongoDB предусмотрен метод remove:

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

Как и в случае с find, мы можем задавать условия выборки для удаления различными способами:

```
> db.users.remove({name : /T\w+/i})
> db.users.remove({age: {$lt : 30}})
> db.users.remove({name : "Tom"}, true) //если указан второй аргумент в true будет удален только один документ
```

Удаление коллекции и базы данных:

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

ПОЛЕЗНАЯ ССЫЛКА

https://metanit.com/nosql/mongodb/

- Руководство по MongoDB