
mongoDB

- **Язык:** C++
- Master/slave репликация (auto-failover with replica sets)
- Sharding built-in
- Javascript-like язык запросов
- Запуск кастомных javascript-функций на стороне сервера
- Ипользует mmap
- Performance over features
- Журналирование (--journal)
- На 32bit системах ограничение ~2.5Gb
- Пустая база ~192Mb
- GridFS для хранения больших файлов и метаданных
- Геопространственные индексы

Кастомный – произвольный, созданный под конкретного прользователя
Mmap – отображение файла в оперативную память
Performance over features – производительность важнее функций
Журналирование – форма автоматической записи в хронологическом порядке


craigslist


ebay


foursquare


SAP


CERN

4.1 Настройка MongoDB

4.1.1 Скачивание

Для загрузки перейдите по адресу `https://mongodb.com` и нажмите на зелёную кнопку «Download» в правом верхнем углу. Прокрутите вниз появившуюся страницу и нажмите кнопку «Download» в середине под надписью «Installation Package» (снимок 4.1).

4.1.2 Установка

Запустите скачанный файл, примите лицензионное соглашение, затем нажмите кнопки «Complete» и «Install».

4.1.3 Конфигурирование

Добавьте папку `C:▸Program Files▸MongoDB▸Server▸3.2▸bin`[1] к переменной Path среды. Для этого в основном меню операционной системы наберите «Панель управления», откройте её и выберите раздел «Система». В списке слева нажмите на пункт «Дополнительные параметры системы», а в появившемся окне «Свойства системы» — кнопку «Переменные среды...». В группе «Системные переменные» окна «Переменные среды» выделите переменную Path и нажмите кнопку «Изменить...». Дальнейшие действия в новом окне зависят от версии операционной системы:

- в Windows 7 в поле «Значение переменной» допишите точку с запятой в конец строки, затем вставьте путь до папки;

- в Windows 10 нажмите кнопку «Создать» и в появившееся поле вставьте путь до папки.

Закройте все открытые окна нажатием кнопок «OK».

Из терминала PowerShell при помощи команды `mkdir` создайте директорию для баз данных.

```
mkdir C:/Data/DB
```

Для удаления файлов и папок используют команду `rm`.

Чтобы запустить сервер MongoDB, наберите в терминале `mongod`. Обратите внимание, что закрытие терминала повлечёт за собой остановку сервера.

4.2 Клиент MongoDB

Запустите второй терминал и выполните `mongo`, чтобы подключиться к MongoDB. По умолчанию используется база данных с именем test. Введите **db**, чтобы проверить имя текущей базы.

```
db
<: test
```

Для вывода списка всех непустых баз данных наберите **show databases**.

```
show databases
<: local  0.000GB
```

Выберем базу данных с именем sandbox в качестве текущей. Для этого воспользуйтесь командой **use**. Поскольку база с указанным именем не существует, она будет автоматически создана.

```
use sandbox
<: switched to db sandbox
```

4.3 Функциональность

4.3.1 Структура

Каждая база данных MongoDB состоит из *коллекций*. Коллекция — это набор *документов* (словарей) преимущественно одинакового устройства, записываемых при помощи JSON.

```
[
    {
        name: "John",
        age: 30
    },
    {
        name: "Mary",
        age: 23
    }
]
```

4.3.2 Запросы

Добавим по одному документы в коллекцию humans текущей базы.

```
db.humans.insert({ name: "John", age: 30 })
<: WriteResult({ "nInserted" : 1 })
db.humans.insert({ name: "Mary", age: 23 })
<: WriteResult({ "nInserted" : 1 })
```

Теперь совершим при помощи функции **find** запрос, выбирающий все документы из коллекции, а затем распечатаем отформатированный результат функцией **pretty**.

```
var documents = db.humans.find();
documents.pretty()
<: {
<:         "_id" : ObjectId("..."),
<:         "name" : "John",
<:         "age" : 30
<: }
<: {
<:         "_id" : ObjectId("..."),
<:         "name" : "Mary",
<:         "age" : 23
<: }
```

Обратите внимание, что в клиенте MongoDB используется обычный JavaScript.

Для вывода списка всех непустых коллекций используют команду **show collections**.

```
show collections;
<: humans
```

4.3.2.1 Одноуровневый поиск

Добавим ещё несколько документов в нашу коллекцию.

```
db.humans.insertMany([
...     {
...         name: "John",
...         age: 25
...     },
...     {
...         name: "Анна",
...         age: 27
...     }
... ])
<: {
<:     "acknowledged" : true,
<:     "insertedIds" : [
<:             ObjectId("..."),
<:             ObjectId("...")
<:     ]
<: }
```

Подумайте, какой тип имеет аргумент функции **insertMany**.

Теперь выберем те документы, у которых поле **name** имеет значение **"John"**.

```
db.humans.find({ name: "John" })
<: { "_id" : ObjectId("..."), "name" : "John", "age" : 30 }
<: { "_id" : ObjectId("..."), "name" : "John", "age" : 25 }
```

Условия на значения полей можно задавать при помощи словарей, ключами которых являются специальные названия операторов сравнения. Приведём некоторые из них:

- **$lt** — «**l**ess **t**han» — меньше;
- **$gt** — «**g**reater **t**han» — больше;
- **$lte** — «**l**ess **t**han or **e**qual» — меньше или равно;
- **$gte** — «**g**reater **t**han or **e**qual» — больше или равно;
- **$eq** — «**eq**ual» — равно;
- **$ne** — «**n**ot **e**qual» — не равно.

```
db.humans.find({ age: { $lt: 26 } })
<: { "_id" : ObjectId("..."), "name" : "Mary", "age" : 23 }
<: { "_id" : ObjectId("..."), "name" : "John", "age" : 25 }
```

Несколько условий можно комбинировать внутри одного словаря, имитирующего работу логического «И».

```
db.humans.find({ age: { $gte: 25, $lt: 28 } })
<: { "_id" : ObjectId("..."), "name" : "John", "age" : 25 }
<: { "_id" : ObjectId("..."), "name" : "Анна", "age" : 27 }
```

Подумайте, как можно перевести следующий запрос на естественный язык.

```
db.humans.find({ name: "John", age: { $gt: 28 } })
<: { "_id" : ObjectId("..."), "name" : "John", "age" : 30 }
```

Чтобы объединить несколько условий посредством логического «ИЛИ», их записывают в массив, являющийся значением поля **$or**.

```
db.humans.find({ $or: [{ name: "John" }, { name: "Mary" }] })
<: { "_id" : ObjectId("..."), "name" : "John", "age" : 30 }
<: { "_id" : ObjectId("..."), "name" : "Mary", "age" : 23 }
<: { "_id" : ObjectId("..."), "name" : "John", "age" : 25 }
```

Однако в нашем случае запрос можно переписать проще при помощи **$in**.

```
db.humans.find({ name: { $in: ["John", "Mary"] } })
<: { "_id" : ObjectId("..."), "name" : "John", "age" : 30 }
<: { "_id" : ObjectId("..."), "name" : "Mary", "age" : 23 }
<: { "_id" : ObjectId("..."), "name" : "John", "age" : 25 }
```

Противоположностью **$in** является **$nin**.

```
db.humans.find({ name: { $nin: ["John", "Mary"] } })
<: { "_id" : ObjectId("..."), "name" : "Анна", "age" : 27 }
```

4.3.2.2 Многоуровневый поиск

Пусть в коллекции sportsmen хранятся данные о выступлениях спортсменов за различные клубы.

```
{
    name: "Dominik Hašek",
    clubs: [
        {
            clubName: "HC Pardubice",
            firstYear: 1980,
            lastYear: 1989
        },
        {
            clubName: "HC Spartak Moscow",
            firstYear: 2010,
            lastYear: 2011
        }
    ]
}
```

Известно, что данные в массиве упорядочены по времени. Требуется найти тех спортсменов, для которых первым профессиональным клубом был HC Pardubice, то есть задать соответствующее условие на поле **clubName** элемента с индексом 0 в массиве **clubs**.

В JavaScript для обращения к полям вложенных структур используются квадратные скобки.

```
sportsman["clubs"][0]["clubName"]
```

А в запросах MongoDB в качестве разделителя применяется точка.

```
db.sportsmen.find({ "clubs.0.clubName": "HC Pardubice" })
```

Подумайте, зачем ключ взят в кавычки.

4.3.3 Удаление

При помощи функции **deleteMany** можно удалять документы из коллекции.

```
db.humans.deleteMany({ name: "Анна" })
<: { "acknowledged" : true, "deletedCount" : 1 }
```

Чтобы удалять коллекции, используют функцию **drop**.

```
db.humans.drop()
<: true
```

Для удаления баз данных существует функция **dropDatabase**.

```
db.dropDatabase()
<: { "dropped" : "sandbox", "ok" : 1 }
```

Команды

1. install: **sudo apt-get install mongodb**;
2. start : **mongo**;
3. Показать БД: **show dbs**;
4. Выбрать БД: **use dbName**;
5. Показать все коллекции: **show collections**; OR **show tables**;
6. insert: **db.collectionName.insert(document)**;
7. select limit 20: **db.collectionName.find()**;
8. select limit 1: **db.collectionName.findOne()**;
9. drop collection: **db.collectionName.drop()**;
10. drop database: **db.dropDatabase()**;
11. truncate collection: **db.collectionName.remove()**;
12. update: **db.collectionName.update(document1, document2);**

Поиск

1. **Like Query:**

   db.Tag.find({name:/^term/});

2. **Sort Query:**

   1 for ascending sort

   -1 for descending sort

   db.Tag.find().sort({userName:-1, age:1});

3. **Limit Query:**

   db.Tag.find().limit(10);

Поиск

4. **Count Query:**

   db.Tag.find().count();

5. **Skip Query:**

   db.Tag.find().skip(50);

6. **Keys**:

   db.Tag.find({}, { _id:0, firstProperty:1})

NOTE: чтобы просмотреть JS код функции – вызовете ее без скобок.

Модификация

1. **Not Equal Modifier($ne):**

   db.Tag.find({firstProperty : {$ne : 3}});

2. **Greater/Less than Modifier($gt, $lt, $gte, $lte):**

   db.Tag.find({firstProperty : {$gt : 2}});

   db.Tag.find({firstProperty : {$lt : 2}});

   db.Tag.find({firstProperty : {$gte : 2}});

   db.Tag.find({firstProperty : {$lte : 2}});

Модификация

3. **Increment Modifier($inc):**

   db.Tag.update({thirdProperty:"Hello19"},
   {"$inc": {firstProperty:2}})

4. **Set Modifier($set):**

   db.Tag.update({thirdProperty:"Hello19"},
   {"$set": {fourthProperty: "newValue"} })

NOTE: Key-value пара создается даже, если ключ не существовал до этого.

Модификация

5. **Unset Modifier($unset):**

   db.Tag.update({thirdProperty:"Hello19"},
   {"$unset": {fourthProperty: "anything"} })

6. **Push Modifire($push):**

   db.Blog.update({title:"1st Blog"}, { $push:
   {comment:"1st Comment"} });

NOTE: "$push" создаст массив, если его не было до этого.

Модификация

7. **AddToSet Modifier($addToSet):**

   db.Blog.update({title:"1st Blog"}, { $addToSet:
   {comment:"2nd Comment"} });

8. **Pop Modifier($pop):**

   comment = -1 remove an element from start.

   comment = 1 remove an element from end

   db.Blog.update({title:"1st Blog"}, {$pop:{comment:-1} });

Модификация

9. Pull Modifier($pull):

db.Blog.update({title:"1st Blog"}, {$pull:{comment:"2st Comment"}})

10. Position Operator($): The positional operator updates only the first match.

db.embededDocument.update({"comments.author" : "John"},
{"$inc" : {"comments.0.votes" : 1}})
db.embededDocument.update({"comments.votes" : 3},
{"$set" : {"comments.$.author" : "Amit Kumar"}})

Модификация

11. "$in"

```
db.Blog.find( {comment: {$in:["5th Comment", "1st Comment"]} } )
```

12. "$or"

```
db.Tag.find( {$or: [ {firstProperty:0}, {secondProperty:/J/} ] } )
```

NOTE: есть еще "$nin".

13. "$all"

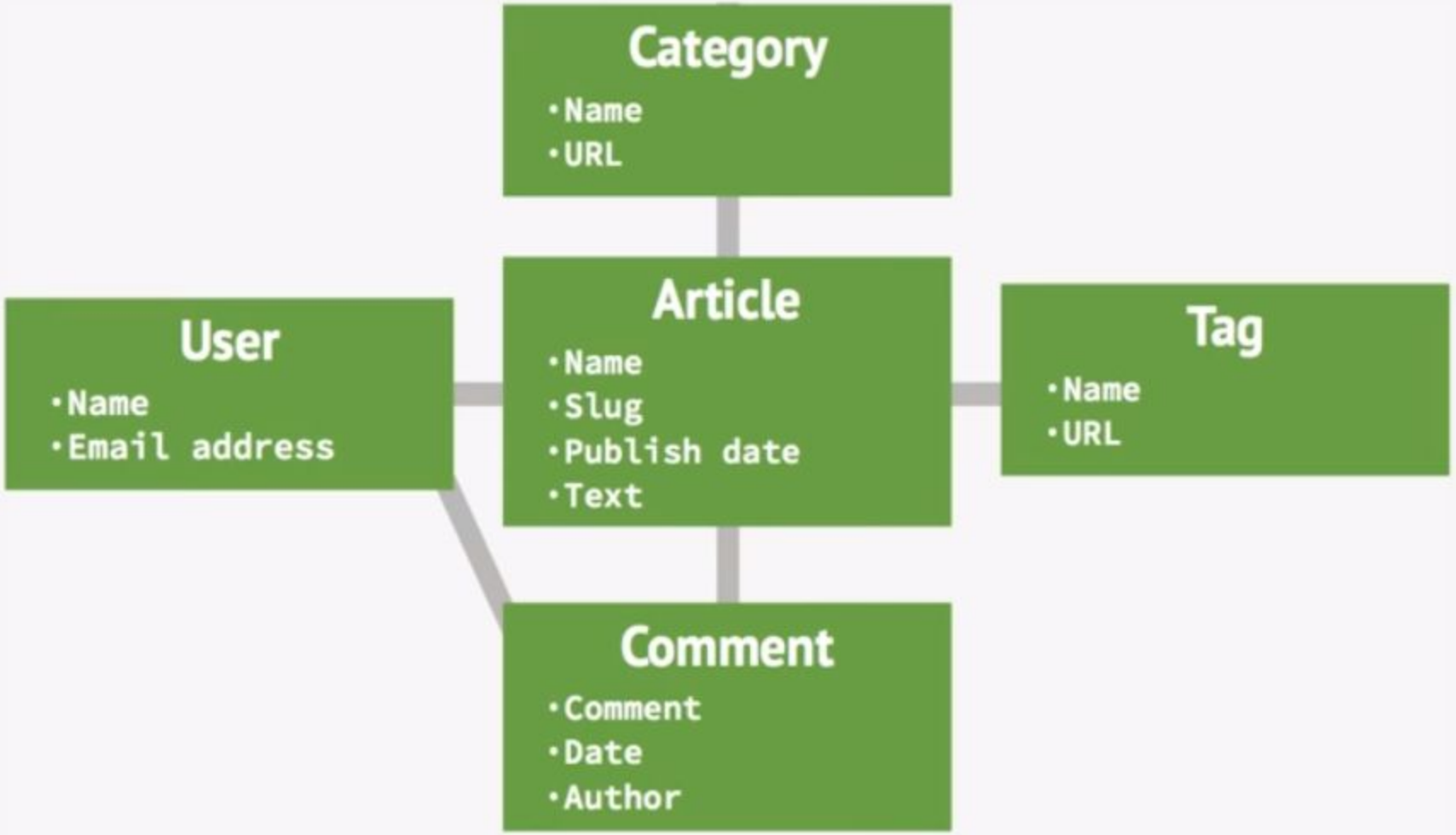
```
db.Blog.find( {comment:{$all:["5th Comment", "1st Comment"]} } )
```

Upsert

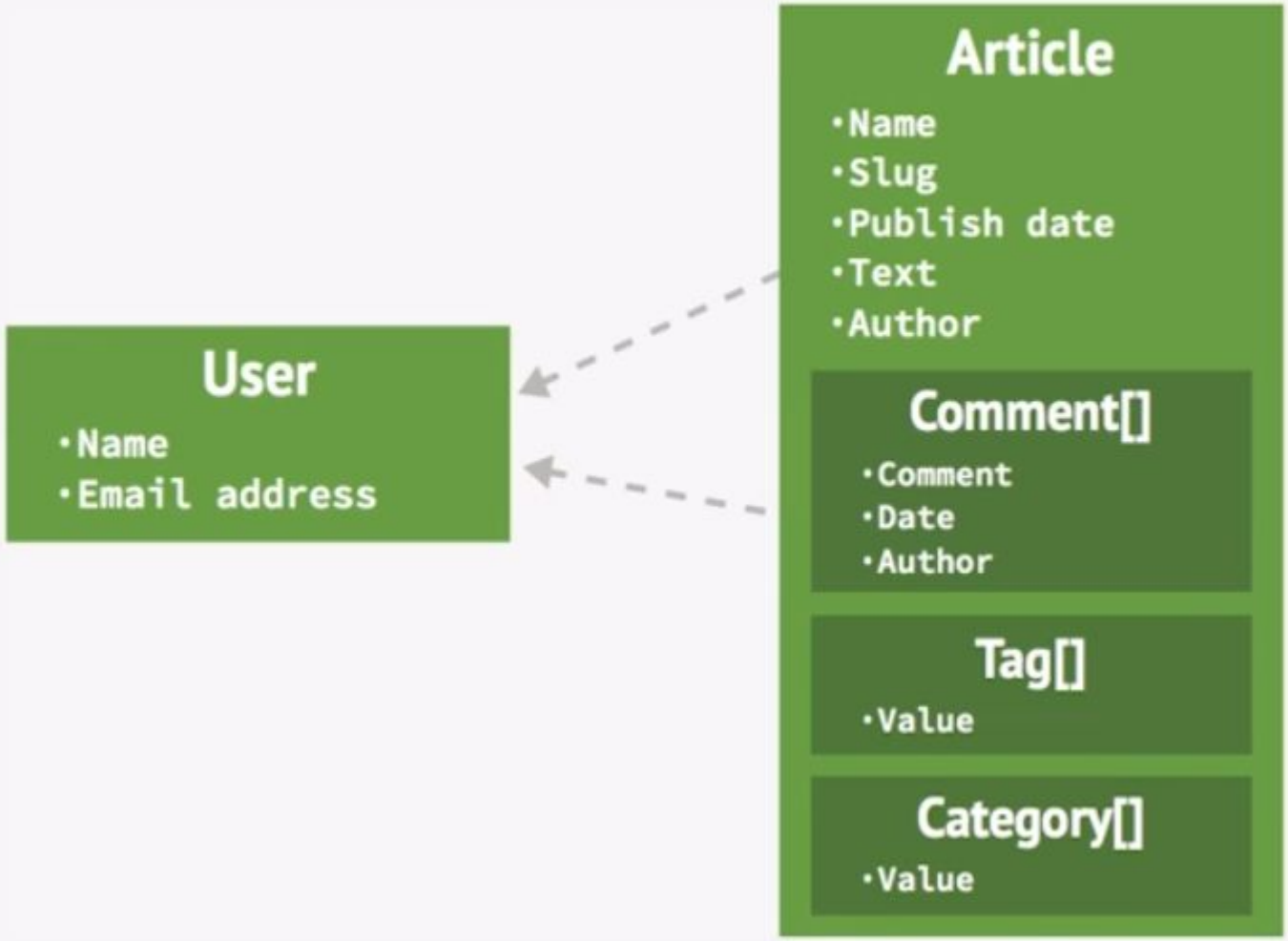
```
db.Tag.update(document1, document2, {upsert: true});
db.Tag.update({secondProperty:”Jav”}, {"$inc":{"firstProperty": 100}}, {upsert: true});
db.Tag.save(document)

db.Tag.update({secondProperty:/J/}, {"$inc":{"firstProperty":100}}, {multi: true})
```

Normalized Data



De-Normalized (embedded) Data



Library Management Application

Patrons

Books

Authors

Publishers

```
patron = {
    _id: "joe",
    name: "Joe Bookreader”
}

address = {
    patron_id = "joe",
    street: "123 Fake St. ",
    city: "Faketon",
    state: "MA",
    zip: 12345
}
```

```
patron = {
    _id: "joe",
    name: "Joe Bookreader",
    address: {
        street: "123 Fake St. ",
        city: "Faketon",
        state: "MA",
        zip: 12345
    }
}
```

```
patron = {
  _id: "joe",
  name: "Joe Bookreader",
  join_date: ISODate("2011-10-15"),
  addresses: [
    {street: "1 Vernon St.", city: "Newton", state: "MA", ...},
    {street: "52 Main St.", city: "Boston", state: "MA", ...}
  ]
}
```

```
book = {
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    publisher: {
        name: "O'Reilly Media",
        founded: "1980",
        location: "CA"
    }
}
```

```
publisher = {
    name: "O'Reilly Media",
    founded: "1980",
    location: "CA"
}

book = {
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}
```

```
publisher = {
    _id: "oreilly",
    name: "O'Reilly Media",
    founded: "1980",
    location: "CA"
}

book = {
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    publisher_id: "oreilly"
}
```

```
publisher = {
    name: "O'Reilly Media",
    founded: "1980",
    location: "CA"
    books: [ "123456789", ... ]
}

book = {
    _id: "123456789",
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}
```

```
patron = {
    _id: "joe",
    name: "Joe Bookreader",
    join_date: ISODate("2011-10-15"),
    address: { ... },
    checked_out: [
        { _id: "123456789", checked_out: "2012-10-15" },
        { _id: "987654321", checked_out: "2012-09-12" },
        ...
    ]
}
```

```
patron = {
    _id: "joe",
    name: "Joe Bookreader",
    join_date: ISODate("2011-10-15"),
    address: { ... },
    checked_out: [
        { _id: "123456789",
          title: "MongoDB: The Definitive Guide",
          authors: [ "Kristina Chodorow", "Mike Dirolf" ],
          checked_out: ISODate("2012-10-15")
        },
        { _id: "987654321"
          title: "MongoDB: The Scaling Adventure",
                                  ...
        }, ...
    ]
}
```

```
book = {
    title: "MongoDB: The Definitive Guide",
    authors = [
        { _id: "kchodorow", name: "K-Awesome" },
        { _id: "mdirolf", name: "Batman Mike" },
    ]
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}

author = {
    _id: "kchodorow",
    name: "Kristina Chodorow",
    hometown: "New York"
}
```

```
book = {
    _id: 123456789,
    title: "MongoDB: The Definitive Guide",
    authors = [ "kchodorow", "mdirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}

author = {
    _id: "kchodorow",
    name: "Kristina Chodorow",
    hometown: "Cincinnati",
    books: [ 123456789, ... ]
}
```

```
book = {
    _id: 123456789,
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}

category = { _id: MongoDB, children: [ 123456789, ... ] }
category = { _id: Databases, children: ["MongoDB", "Postgres"}
category = { _id: Programming, children: ["DB", "Languages"] }
```

How do I create indexes?

```
// The client remembers the index and raises no errors
db.recipes.ensureIndex({ main_ingredient: 1 })
```

```
// Multiple fields (compound indexes)
db.recipes.ensureIndex({
    main_ingredient: 1,
    calories: -1
})

// Arrays of values (multikey indexes)
{
    name: 'Curry Wurst mit Pommes',
    ingredients : ['pork', 'curry']
}

db.recipes.ensureIndex({ ingredients: 1 })
```

- https://www.youtube.com/watch?v=_I9vLOAsFew