### Лабораторная работа № 9

### Введение в MongoDB (NoSQL - Not only SQL)

В соответствии с рейтингами последних лет к лидирующим моделям баз данных относятся реляционные СУБД (relational DBMS): Oracle Database, MySQL, Microsoft SQL Server, PostgreSQL и документоориентированная СУБД (document-oriented database) MongoDB. Следует отметить, что DBMS MongoDB относится к нереляционным СУБД NoSQL с новыми методами хранения данных и средствами доступа к ним.

В зависимости от модели базы данных к основным СУБД NoSQL относятся:

* БД на основе пар «ключ - значение» (DynamoDB, Redis, Aerospike и др.)
* Колоночно-ориентированные СУБД или column-oriented DBMS (Cassandra, Hypertable, ConcourseDB и др)
* Документоориентированные СУБД (MongoDB, Azure DocumentDB, CouchDB и др.)
* Графовые БД (Neo4j, InfoGrid, Bigdata и др.)
* Базы данных XML (EMC Documentum xDB, eXist, Sedna и др.)

Если реляционную модель баз данных традиционно используют для оперативной обработки транзакций или в системах поддержки принятия решений, то модель NoSQL применяют для оперативной обработки данных. Технология NoSQL обеспечивает эффективное хранение и обработку огромных объемов неструктурированных данных, которые требуют высоких скоростей для выполнения операций чтения и записи.

Следует отметить, что базы данных NoSQL являются одной из технологий обработки Big data. Особенностью технологий NoSQL является неограниченное горизонтальное масштабирование и высокая производительность. Необходимо также отметить, что в некоторых реляционных СУБД, например, таких как PostgreSQL реализованы средства обработки неструктурированных данных в формате JSON.

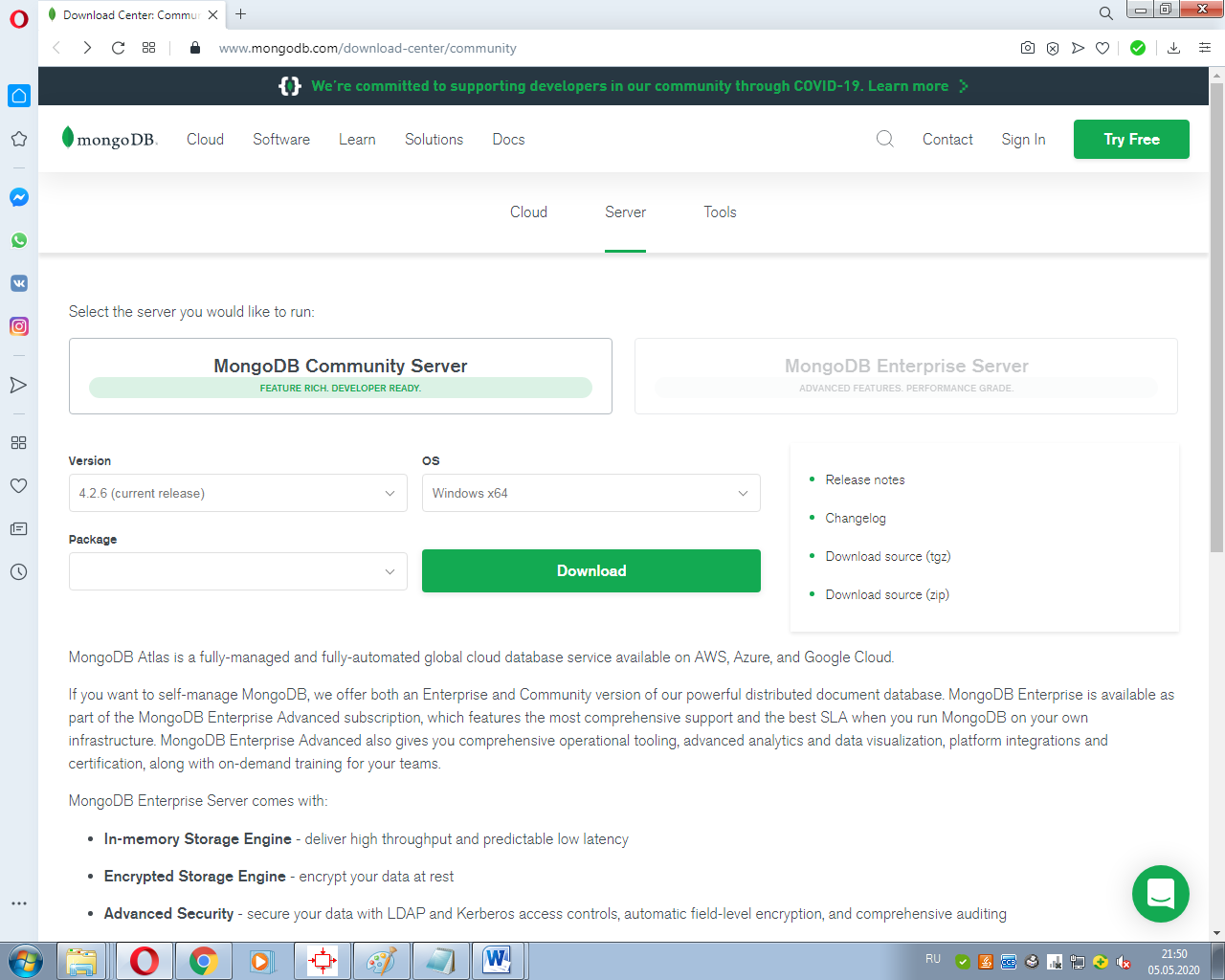
БД MongoDB востребованна при разработке Web-приложений на основе Node.js. (традиционный стек технологий для построения веб-приложений – LAMP: Linux, Apache, MySQL, PHP; стек технологий с использованием БД MongoDB - MEAN (Mongo, Express, Angular, Node).

**ЗАДАНИЕ №1. Ознакомиться с видео** [**https://www.youtube.com/watch?v=RBCVGRkDMQM**](https://www.youtube.com/watch?v=RBCVGRkDMQM)

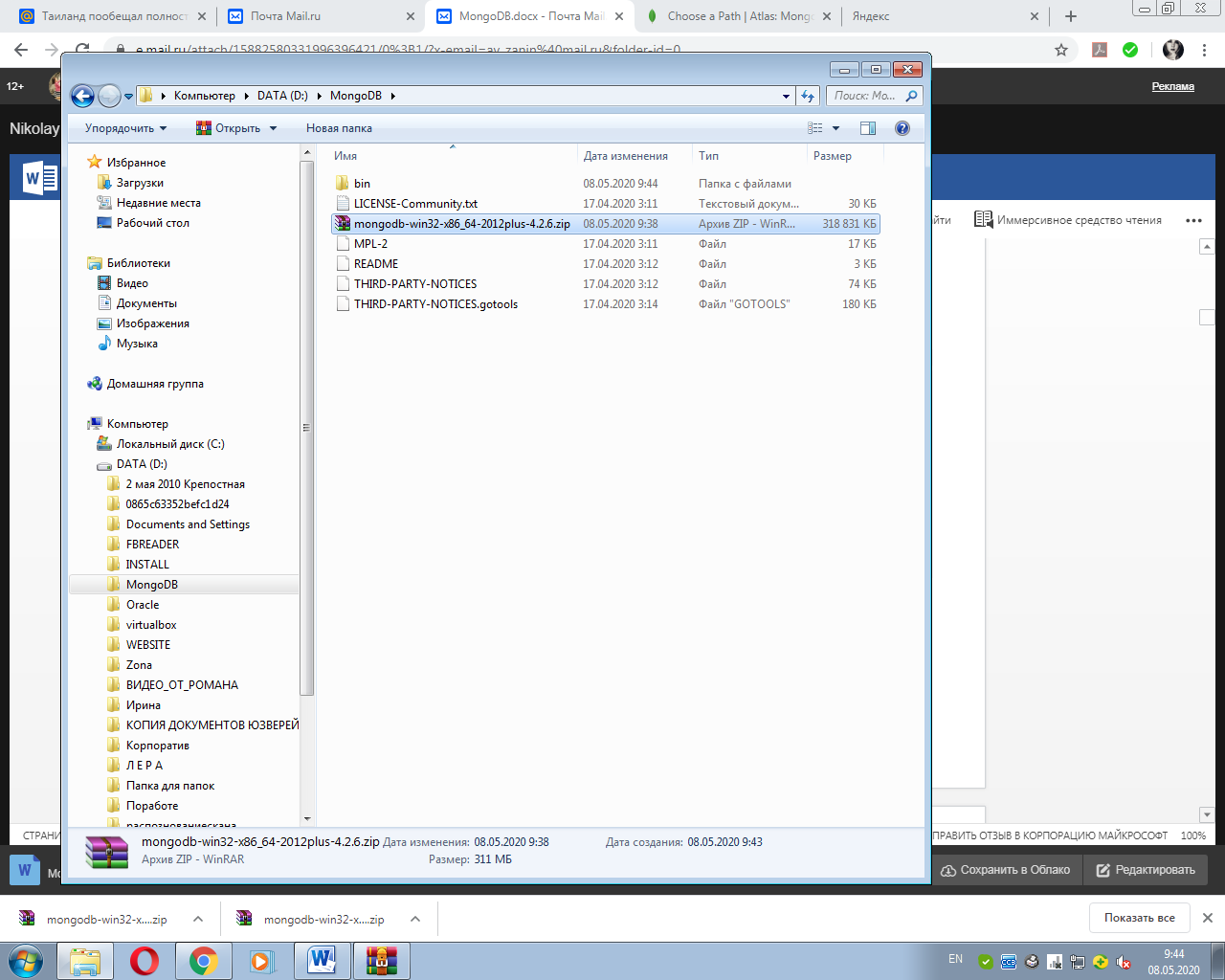
MongoDB — документоориентированная СУБД с открытым исходным кодом, не требующая описания схемы таблиц. Она классифицируется как NoSQL и использует BSON (бинарный JSON\*), т.е. каждая запись это документ, без жестко заданной схемы, который может содержать вложенные документы. Хорошо масштабируется, написана на языке C++ и поддерживает синтаксис JavaScript. Поддержка SQL отсутствует. У MongoDB есть драйверы для многих популярных языков программирования (Си, C++, C#, Go, Java, JavaScript, Perl, PHP, Python, Ruby и др.). Также есть неофициальные и поддерживаемые сообществом драйверы для прочих языков программирования.

**ЗАДАНИЕ №2 Установка и начало работы с MongoDB**

Для установки MongoDB загрузим один распространяемых пакетов с официального сайта <https://www.mongodb.com/download-center/community>



Сформируйте архив ZIP, версии 4.2.6 (пошаговую установку см. <https://metanit.com/nosql/mongodb/1.1.php>)



Создайте директорию c:\MongoDB. Распакуйте архив в эту директорию.

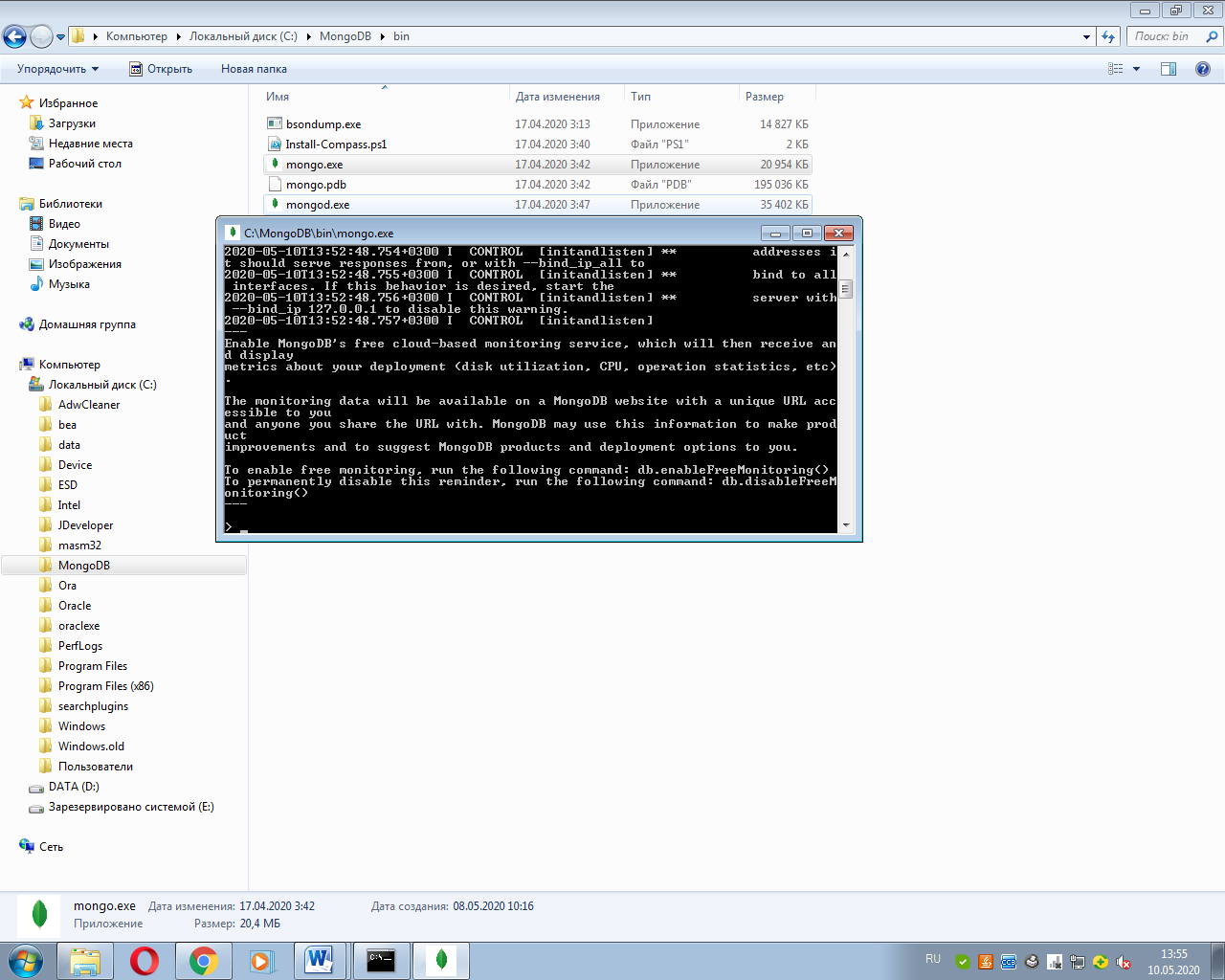
Создайте директорию для хранения базы с:\data\db

Запустите командную строку (в Windows) или консоль в Linux.

Для работы с БД установите нужный каталог cd c:\MongoDB\bin

Запустите БД c:\MongoDB\bin>mongod

Из каталога c:\MongoDB\bin (в Windows) запустите mongo.exe



Дальнейшая работа будет в интерактивном режиме из строки

>

( если возникли проблемы с установкой см. видео <https://www.youtube.com/watch?v=TIH3dcZjrzk>)

**ЗАДАНИЕ №3 Осуществить следующие манипуляции в командной строке (БД)**

- подсказка

>help

- вывести названия всех имеющихся бд

>show dbs

-создадим новую базу данных по имени users

> use users

>show dbs

-создадим еще одну новую базу данных по имени users1

> use users1

- узнать, какая бд используется в текущей момент

>db

- перейти в базу данных users

> use users

- удалить базу данных можно командой (важный момент - нужно находиться в той базе данных, которую необходимо удалить) :

>db.dropDatabase()

>show dbs

- перейти в базу данных users1 и удалить ее

> use users1

>db.dropDatabase()

>show dbs

- получить статистику о сервере MongoDB

>db.stats ()

**КОЛЛЕКЦИИ**

|  |  |
| --- | --- |
|  |  |

**\*JSON** (JavaScript Object Notation) - простой формат обмена данными, удобный для чтения и написания, как человеком, так и компьютером. Он основан на подмножестве [языка программирования JavaScript](http://javascript.crockford.com/). JSON - текстовый формат, полностью независимый от языка реализации, но он использует соглашения, знакомые программистам C-подобных языков, таких как C, C++, C#, Java, JavaScript, Perl, Python и многих других. Эти свойства делают JSON идеальным языком обмена данными. JSON основан на двух структурах данных: Коллекция пар ключ/значение. В разных языках, эта концепция реализована как *объект*, запись, структура, словарь, хэш, именованный список или ассоциативный массив. Упорядоченный список значений. В большинстве языков это реализовано как *массив*, вектор, список или последовательность. Это универсальные структуры данных. Почти все современные языки программирования поддерживают их в какой-либо форме.

Метод createCollection () db.createCollection (имя, параметры) используется для создания коллекции.

Синтаксис команды createCollection () следующий:

db.createCollection(name, options)

name это имя коллекции, которая будет создана. Параметры – это документ, который используется для указания конфигурации коллекции.

|  |  |  |
| --- | --- | --- |
| параметр | Тип | Описание |
| название | строка | Название коллекции, которая будет создана |
| Опции | Документ | (Необязательно) Укажите параметры, касающиеся объема памяти и индексации |

Параметр options является необязательным, поэтому необходимо указывать только название коллекции. Ниже приведен список опций, которые вы можете использовать –

|  |  |  |
| --- | --- | --- |
| поле | Тип | Описание |
| ограничен | логический | (Необязательно) Если установлено значение true, включает ограниченную коллекцию. Ограниченная коллекция – это коллекция фиксированного размера, которая автоматически перезаписывает свои самые старые записи, когда достигает максимального размера. Если вы указываете true, вам также нужно указать параметр размера. |
| autoIndexId | логический | (Необязательно) Если установлено значение true, автоматически создавать индекс для поля \_id.s Значение по умолчанию – false. |
| размер | число | (Необязательно) Указывает максимальный размер в байтах для ограниченной коллекции. Если значение capped равно true, то вам также необходимо указать это поле. |
| Максимум | число | (Необязательно) Указывает максимально допустимое количество документов в ограниченном собрании. |

При вставке документа MongoDB сначала проверяет поле размера коллекции с ограничениями, а затем проверяет поле max.

**ЗАДАНИЕ №4 Осуществить следующие манипуляции в командной строке (простые команды)**

> db.mycoll.help()

> db.mycoll. createCollection.help()

- Создать коллекцию c именем "mycollection" в БД test (коллекция может создаваться автоматически при добавлении документов, но можно создать явно)

>use test

>db.createCollection("mycollection")

- Создать коллекцию

>db.createCollection("mycol", { capped : true, autoIndexId : true, size :

6142800, max : 10000 } )

- Создать коллекцию:

db.createCollection("ps")

- Вывести названия всех имеющихся коллекций

>show collections

 - Удаление коллекции:

>db.ps.drop()

- Переименовать коллекцию (см. коллекции не было, создание – автоматически) :

>db.ps.renameCollection("peoples")

 >show collections

- Добавить один документ в коллекцию:

> db.insertOne.help()

>db.peoples.insertOne({"name": "Mike", "lastname": "Kozlov"})

-Добавить несколько документов в коллекцию:

>db.peoples.insertMany([{"name": "Mike", "lastname": "Kozlov"},{"name": "Ivan", "lastname": "Kruglov"}])

-Посчитать количество документов в коллекции и при выборке:

>db.peoples.count()

>db.peoples.find({name: "Mike"}).count()

>db.peoples.find({name: "Ivan"}).count()

-Добавить любое количество документов в коллекцию:

>db.peoples.insert([{"name": "Mike", "lastname": "Kozlov"}])

-Пример добавления с вложенными данными:

>db.peoples.insert({"name": "Mike", "lastname": "Kozlov", address: {"country": "Russia", "city": "Moscow", "index": 111511}})

-Выборка из коллекции:

>db.peoples.find()

-Выборка из коллекции уникальных значений поля "name":

>db.peoples.distinct("name")

-Выборка по одному и нескольким условиям:

>db.peoples.find({name: "Mike"})

>db.peoples.find({lastname: "Kozlov"})

-Выборка документов без какого-то свойства(полю присваиваем значение 0 либо true\false):

>db.peoples.find({name: "Mike"}, {lastname: 0})

>db.peoples.find({name: "Mike"}, {lastname: false})

-Выборка всех документов без какого-то свойства:

>db.peoples.find({}, {lastname: 0})

-Выборка по вложенному элементу:

>db.peoples.find({"address.country": "Russia"})

-Выборка с ограничением по количеству выводимых документов:

>db.peoples.find().limit(2)

-Выборка с сортировкой по полю(по возрастанию - 1, по убыванию - -1):

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

-Выборка с несколькими функциями:

>db.peoples.find().sort({name: 1}).limit(2)

-Выборка с помощью JavaScript:

>func = function() { return this.name=="Mike"; }

>db.peoples.find(func)

-Выборка с условными операторами:

$eq - =

$gt - >

$lt - <

$gte - >=

$lte - <=

$ne - <>

-Пример с двумя условиями больше и меньше указанных значений:

>db.peoples.find({"address.index": {$gt : 111500, $lt : 111700}})

 Изменение данных

Изменение документа:

Функция update принимает три параметра для обновления документа:

1 документ, который будет изменен

2 документ, на который будет изменен

3 Параметры обновления документов (upsert и multi)

3.1.1 upsert имеет значение true, то будет обновлен документ, если он найден, и создан новый, если документа нет

3.1.2 upsert имеет значение false, то не будет создан новый документ, если найденного документа нет

3.2.1 multi определяет, должен ли обновиться первый элемент в выборке (используется по умолчанию, если не указан)

3.2.2 multi определяет, должны обновляться все документы выборки

-Пример (найденный документ - {name : "Mike"}, будет заменен документом {name: "Mike", lastname: "Degtyarev"}):

>db.peoples.update( { name : "Mike" }, { name: "Mike", lastname: "Degtyarev" }, { upsert: true } )

-Изменение одного поля первого в выборке документа, укажем оператор $set, если документт не имеет изменяемого поля, создастся новое поле в документете:

>db.peoples.update({name : "Mike", lastname: "Degtyarev"}, {$set: {lastname: "Kozlov"}})

-В данном случае изменится первый в выборке документ, указав оператор multi:true, изменятся все документы из выборки:

>db.peoples.update({name : "Mike", lastname: "Degtyarev"}, {$set: {lastname: "Kozlov"}}, {multi:true})

Удаление данных

-Удаление поля, укажем оператор $unset:

>db.peoples.update({name : "Mike"}, {$unset: {lastname: "Kozlov"}})

Просмотрим документы

>db.peoples.find()

>db.peoples.update({name : "Mike"}, {$unset: {lastname: "Kozlov", name: "Mike"}})

 Просмотрим документы

>db.peoples.find()

-Удаление документа:

Посчитаем количество

>db.peoples.find({name: "Mike"}).count()

>db.peoples.deleteOne({name : "Mike"})

 Посчитаем количество

>db.peoples.find({name: "Mike"}).count()

-Удаление множества документов:

>db.peoples.remove({name : "Mike"})

-Удаление всех элементов коллекции:

db.peoples.remove({})

 Посчитаем количество

>db.peoples.find({name: "Mike"}).count()

Репликация и шардинг являются важнейшими технологиями СУБД MongoDB, поскольку позволяют организовать распределенные вычисления.

Репликация

Под репликацией понимается размещение и обслуживание серверов базы данных на нескольких физических узлах. Цель репликации — обеспечить избыточность и обработку отказа (например, для восстановления после сбоя аппаратного обеспечения) и увеличить доступность и балансировку данных (например, для распределенных приложений).

В MongoDB имеется два типа репликации: первый тип — «главный—подчиненный» (Master Slave Replication), второй тип — «наборы реплик» (Replica Sets). Последний является развитием механизма репликации Master Slave Replication и сохраняет основные ее принципы работы. И в том, и в другом случае имеется первичный узел (Primary), который выполняет все операции записи, и вторичные узлы (Secondary), которые считывают описания операций, выполненных на первичном узле, и применяют их. Однако использование наборов реплик предпочтительнее ввиду наличия более широких возможностей, например автоматического восстановления (в частности, в случае отказа первичного узла имеется механизм автоматического назначения первичным узлом одного из вторичных).

Шардинг (сегментирование)

Шардинг является ключевой концепцией MongoDB. До сих пор каждый запущенный процесс mongod оперировал с полной копией базы данных, пусть даже эти копии были размещены на различных узлах (в случае репликации). Принципиальным отличием шардинга является сегментирование базы данных и размещение сегментов на различных серверах. Это особенно актуально в тех случаях, когда объем базы данных не позволяет, например, организовать хранение данных ввиду ограничений возможностей адресации или снижает эффективность работы пользователей из-за того, что сервер не справляется с нагрузкой. Важным является то, что интерфейс MongoDB позволяет приложению работать с сегментированной базой данных абсолютно точно так же, как и с базой данных, расположенной на одном сервере. В состав сегментированного кластера входят собственно сегменты (shards), причем каждый из сегментов для повышения надежности состоит из набора реплик, маршрутизаторов mongos (процесс, перенаправляющий операции с данными к требуемому сегменту) и конфигурационных серверов, которые отвечают за конфигурацию кластера, т. е. хранят данные о расположении баз данных, коллекций, диапазонах хранящихся данных, журналы изменений.

**!!!!!! ОЗНАКОМИТЬСЯ !!!!!!!!**

https://proselyte.net/tutorials/mongodb/

Введение

MongoDB состоит из БД, которые состоят из **коллекций**. Коллекции, в свою очередь, состоят из **документов**. Каждый документ состоит из **полей**.  
[www.mongodb.com](https://www.mongodb.com/)

MongoDB клиенты

* <https://robomongo.org/>
* <https://studio3t.com/> (+)

Конфиг mongodb для windows

В папке bin создайте файл и назовите его mongodb.config.

Указываем в этом файле путь, где будем хранить БД, например, для windows:

dbpath=c:\mongodb\data

Запуск

* mongod - это **сервер** баз данных MongoDB. Он обрабатывает запросы, управляет форматом данных и выполняет различные операции в фоновом режиме по управлению БД. Командная строка отобразит нам ряд служебной информации, например, что сервер запускается на localhost на порту 27017.
* mongo - клиентская **консоль** для взаимодействия с базами данных.
* mongodump - утилита создания бэкапа БД.
* mongorestore - позволяет записывать данные из дампа, созданного mongodump, в новую или существующую БД.

Стартуем mongod (сервер):

mongod --config c:\mongodb\bin\mongodb.config

*// не забудьте заранее создать папку data*

Подключаемся к запущенному серверу

Команда mongo позволяет подключиться к запущенному серверу (стартуем mongo оболочку/shell).

mongo

Работаем с БД

Выводим все БД в mongo:

show dbs

Переходим (и одновременно создаем) к нужной БД

use name\_bd

Команды db возвращает имя БД, внутри которой мы сейчас находимся:

db

//test

Коллекции

Показать все коллекции в БД

show collections

Метод find()

Показать весь контент нужной коллекции:

db.collection\_name.find()

Пример:

db.band.find()

db.band.find().pretty()

Метод pretty выводит результат в удобном для чтения виде.

Метод count()

Метод count выводит количество документов в коллекции:

db.band.count()

Метод remove()

Метод remove используется, чтобы удалить документ из коллекции (или всю коллекцию).

db.unicorns.remove({name: "Leto"})

Метод insert()

Заносим данные в коллекцию band (создаем тем самым коллекцию band, если ее нет):

db.band.insert({name: 'Queen', bid: '3'})

Добавим составы груп в коллекцию band:

db.band.update({bid: '1'}, {$set: {members: [

{name: "Jimmy Page", id: "1"},

{name: "robert Plant", id: "2"},

{name: "John Bonham", id: "3"},

{name: "John Paul Jones", id: "4"}]}})

db.band.update({bid: '2'}, {$set: {members: [

{name: "Syd Barret", id: "5"},

{name: "Roger Waters", id: "6"},

{name: "Nick Mason", id: "7"},

{name: "Richard Wright", id: "8"},

{name: "David Gilmour", id: "9"}]}})

Мы можем добавлять данные, не декларируя их предварительно: свойство members. Отсутствует схема: легко добавили массив объектов.

Мы не обязаны создавать коллекции явно. Мы просто можем **вставить документ в новую коллекцию**. Чтобы это сделать, используйте команду insert, передав ей вставляемый документ:

db.unicorns.insert({name: 'Aurora', gender: 'f', weight: 450})

Модификация данных

Оператор $set

Оператор $set заставляет команду update модифицировать лишь те ключи, которые ему переданы (см. пример выше).

Оператор $unset

Оператор $unset удаляет указанный ключ

db.collection.update({id: 2}, {$unset: {myKey: 1}});

db.example.update({}, {$unset: {words:1}}, false, true);

Оператор $inc

Оператор $inc увеличивает значение поля на указанную величину

db.collection.update({id: 2}, {$inc: {myCounter: 111}});

db.collection.update( {"players.playerName":"Joe"}, { $inc : { "players.$.playerScore" : 1 } }

Оператор $rename

Оператор $rename позволяет переименовать поля

db.collection.update({id: 2}, {$rename: {"old\_name": "new\_name"}});

db.band.update({bid: "1"},{$rename:{"members":"members\_new"}});

Индексы

Индексация поддерживает эффективное выполнение запросов. Без индексов MongoDB необходимо сканировать каждый документ коллекции для выбора тех документов, которые соответствуют запросу. Данный процесс крайне неэффективен и требует обработки большого количества данных.

**Индексы MongoDB** – это специальные структуры данных, которые хранят небольшие части данных в форме, которая легко распознаётся. Они хранят значение определённого поля или набора полей, упорядоченных по значению поля, указанному в индексе.

Построим индекс по ключу bid:

db.band.ensureIndex({bid: 1}) *// deprecated*

db.users.createIndex({"name" : 1}) *// actual*

ensureIndex устарел, начиная с версии 3.0, в данный момент является псевдонимом для db.collection.createIndex().

Полезные материалы: на [metanit.com/nosql/mongodb](http://metanit.com/nosql/mongodb/2.12.php) - Работа с индексами,  
[proselyte.net/tutorials/mongodb/indexing](https://proselyte.net/tutorials/mongodb/indexing/) - Индексация в MongoDB

Схемы и модели

Схемы определяют структуру документов внутри коллекции, а модели используются для создания копий данных, хранящихся в документах.

Основы MongoDB

* **1**    Внутри MongoDB может быть ноль или более баз данных.
* **2**    База данных может иметь ноль или более «**коллекций**» (коллекция практически тоже что и таблица).
* **3**    Коллекции состоят из нуля или более «**документов**». Опять же, документ можно рассматривать как «**строку**».
* **4**    Документ состоит из одного или более «**полей**», которые — как можно догадаться — подобны «**колонкам**».
* **5**    «**Индексы**» в MongoDB почти идентичны таковым в реляционных базах данных.
* **6**    Важно понимать, что когда мы запрашиваем у MongoDB какие-либо данные, то она возвращает **курсор**, с которыми мы можем делать все что угодно.

Отличия MongoDB от реляционных БД

Основное различие в том, что реляционные базы данных определяют «колонки» на уровне «таблицы», в то время как документ-ориентированные базы данных определяют «поля» (в реляционных "колонки") на уровне «документа» (в релационных "запись").

В конечном счёте дело в том, что коллекция не содержит информации о структуре содержащихся в ней данных. Информацию о полях содержит каждый отдельный документ.

Селекторы запросов

**Селектор запросов** MongoDB (это JSON-объект) аналогичен предложению where SQL-запроса. Как таковой он используется для поиска, подсчёта, обновления и удаления документов из коллекций.

**Селектор** — это JSON-объект, в простейшем случае это может быть даже {}, что означает выборку всех документов (аналогичным образом работает null). Если нам нужно выбрать всех единорогов (англ. «unicorns») женского рода, можно воспользоваться селектором {gender:'f'}.

{поле: значение} используется для поиска всех документов, у которых есть 'поле' и у него есть 'значение'.

{поле1: значение1, поле2: значение2} работает как логическое И.

Оператор $lt, $lte, $gt, $gte, $ne

Специальные операторы $lt, $lte, $gt, $gte и $ne используются для выражения операций «меньше», «меньше или равно», «больше», «больше или равно», и «не равно».

Пример использовани селекторов с командой find (но также селекторы могут быть использованы с remove, count, update):

Например, чтобы получить всех самцов единорога, весящих более 700 фунтов, мы можем написать:

db.unicorns.find({gender: 'm', weight: {$gt: 700}})

Оператор $exists

Оператор $exists используется для проверки наличия или отсутствия поля, например:

db.unicorns.find({vampires: {$exists: false}})

Оператор $or

Оператор $or используется как ИЛИ

db.unicorns.find({gender: 'f', $or: [{loves: 'apple'}, {loves: 'orange'}, {weight: {$lt: 500}}]})

//... или любят яблоки, или любят апельсины, или весят менее 500 фунтов

Отметьте

db.unicorns.insert({name: 'Leia', dob: new Date(2001, 9, 8, 14, 53), loves: ['apple', 'watermelon'],

weight: 601, gender: 'f', vampires: 33});

Поле loves это массив. MongoDB поддерживает массивы как объекты первого класса. Самое интересное это та простота, с которой делается выборка по значению массива: {loves: 'watermelon'} вернёт нам все документы, у которых watermelon является одним из значений поля loves.

Оператор $where

Оператор $where (в след. разделах)

Самый гибкий оператор — $where, позволяющий нам передавать JavaScript для его выполнения на сервере.

Оператор ObjectId

ObjectId, сгенерированный MongoDB для поля \_id, подставляется в селектор следующим образом:

db.unicorns.find({\_id: ObjectId("TheObjectId")})

update

В простейшей форме, update принимает 2 аргумента: селектор для выборки и то, чем обновить соответствующее поле. **Второй параметр используется для полной замены оригинала**:

db.unicorns.update({name: 'Roooooodles'}, {weight: 590})

По умолчанию, update обновляет лишь первый найденный документ

Модификатор $set

Модификатор $set обновляет конкретные поля, а не весь документ.

Если вам нужно всего лишь изменить пару полей, лучше всего использовать модификатор $set:

db.unicorns.update({weight: 590},

{$set: {name: 'Roooooodles', dob: new Date(1979, 7, 18, 18, 44), loves: ['apple'], gender: 'm', vampires: 99}})

db.unicorns.update({name: 'Roooooodles'}, {$set: {weight: 590}})

Не забывайте использовать модификатор $set, если вам нужно обновить лишь некоторые поля.

модификатор $inc

Модификатор $inc - увеличить или уменьшить значение поля. Модификатор воздействуют непосредственно на поля, а не на весь документ.

db.unicorns.update({name: 'Pilot'}, {$inc: {vampires: -2}})

Модификатор $push

Модификатор $push - позволяет добавить данные в массив. Модификатор воздействуют непосредственно на поля, а не на весь документ.

db.unicorns.update({name: 'Aurora'}, {$push: {loves: 'sugar'}})

Разрешаем вставку при обновлении (3-й параметр)

Обновление/вставка: обновляет документ, если он найден, или создаёт новый — если не найден. Чтобы разрешить вставку при обновлении (**если элемент не будет найден**), установите **третий параметр в** true.

db.hits.update({page: 'unicorns'}, {$inc: {hits: 1}});

db.hits.find();

Вставки и обновления не будет, так как 3-й параметр опущен, а документа с {page: 'unicorns'} отсутствует в коллекции.

db.hits.update({page: 'unicorns'}, {$inc: {hits: 1}}, true);

db.hits.find();

Поскольку документы с полем page, равным unicorns, не существуют, то будет создан новый документ. Если выполнить это вторично, существующий документ будет обновлён, и поле hits увеличится до 2.

Одновременно создастся коллекция hits, если она отсутствует.

Множественные обновления (4-й параметр)

Чтобы обновить множество документов нужно установить **четвертый параметр** в true:

db.unicorns.update({}, {$set: {vaccinated: true }}, false, true);

Этим мы обновили все поля добавив везде поле vaccinated со значением true

find (курсор)

**Курсор базы данных** - это объект БД, который позволяет приложениям работать с записями "по-одной", а не с множеством сразу. То есть курсор (как мы помним это объект), который позволяет передвигаться по выборке (назад на одно, вперед на одну, в конец/начало) при помощи своих методов.

Как уже упоминалось, результатом find является курсор. **Второй необязательный параметр** у find это список полей, которые мы хотим получить.

db.unicorns.find(null, {name: 1});

cursor = db.unicorns.find(null, {name: 1});

\_id по умолчанию возвращается всегда. Но мы можем исключить \_id следующим образом: {name:1, \_id: 0}.

Получаем все поля, кроме поля name:

db.unicorns.find({}, {name: 0})

Как уже упоминалось, результатом find является **курсор**. Поэтому мы **можем присоединить к нему ряд методов**:

Сортировка (метод sort)

Синтаксис метода sort: мы указываем поля, по которым надо сортировать, используя 1 для сортировки по возрастанию и -1 для сортировки по убыванию. Например:

db.unicorns.find().sort({weight: -1})

*// по убыванию 999,998,997 ...*

db.unicorns.find({}, {name: true}).sort({name: -1})

Но для сортировки большого объема данных в MongoDB необходимо использовать **индексы**.

Метод limit()

db.unicorns.find({}, {name: true}).sort({name: -1}).limit(3)

Метод skip()

Метод skip() позволяет пропустить определенное количество записей.

db.unicorns.find({}, {name: true}).sort({name: -1}).limit(3).skip(1)

Обратите внимание как мы соединяем методы в **цепочки**.

Моделирование данных

MongoDB не поддерживает JOIN. По существу мы должны делать второй запрос, чтобы найти связанные данные.

p.s: Для создания нового ObjectID используется следующий код: NewObjectId = ObjectId()

Моделируем 'один-ко-многим' или 'многие-ко-многим'

Когда требуется смоделировать отношения «**один-ко-многим**» или «**многие-ко-многим**» можно использовать массивы ( в MongoDB массивы это объекты первого класса).

db.employees.insert({\_id: ObjectId("4d85c7039ab0fd70a117d733"), name: 'Siona',

manager: [ObjectId("4d85c7039ab0fd70a117d730"), ObjectId("4d85c7039ab0fd70a117d732")] })

При этом следующий find сработает:

db.employees.find({manager: ObjectId("4d85c7039ab0fd70a117d730")})

Массивы значений намного удобнее в использовании, нежели таблицы связи «многие-ко-многим»

Вложенные документы

MongoDB поддерживает вложенные документы:

db.employees.insert({\_id: ObjectId("4d85c7039ab0fd70a117d734"),

name: 'Ghanima', family: {mother: 'Chani', father: 'Paul', brother: ObjectId("4d85c7039ab0fd70a117d730")}})

Вложенные документы можно запрашивать с помощью точечной нотации:

db.employees.find({'family.mother': 'Chani'})

Денормализация

Традиционный путь ассоциировать пользователя с его постом — это колонка userid в таблице posts. С такой моделью нельзя отобразить список постов без дополнительного извлечения данных (JOIN) из таблицы пользователей. Возможное решение — хранить имя пользователя (name) вместе с userid для каждого поста.

Команды (выжимка)

db.version()

*//показывает номер версии сервера*

db.getCollectionNames()

*//получить список коллекций внутри нашей БД*

db.unicorns.find()

*//вернет список документов (записей)*

db.unicorns.remove()

*//поскольку мы не передали селектора, произойдёт удаление всех документов*

//Например, чтобы получить всех самцов единорога, весящих более 700 фунтов, мы можем написать:

db.unicorns.find({gender: 'm', weight: {$gt: 700}})

//Оператор $exists используется для проверки наличия или отсутствия поля, например:

db.unicorns.find({vampires: {$exists: false}})

//Оператор $or используется как ИЛИ

db.unicorns.find({gender: 'f', $or: [{loves: 'apple'}, {loves: 'orange'}, {weight: {$lt: 500}}]})

//update принимает 2 аргумента: селектор (where) для выборки и то, чем обновить соответствующее поле.

//Второй параметр используется для полной замены оригинала

db.unicorns.update({name: 'Roooooodles'}, {weight: 590})

//Модификатор $set обновляет конкретные поля, а не весь документ

db.unicorns.update({name: 'Roooooodles'}, {$set: {weight: 590}})

//модификатор $inc - увеличить или уменьшить значение поля

db.unicorns.update({name: 'Pilot'}, {$inc: {vampires: -2}})

//модификатор $push - позволяет добавить данные в массив

db.unicorns.update({name: 'Aurora'}, {$push: {loves: 'sugar'}})

*//Обновление/вставка обновляет документ, если он найден, или создаёт новый — если не найден.*

*//Чтобы разрешить вставку при обновлении, установите третий параметр в true (ниже мы создаем коллекцию hits, если ее нет)*

db.hits.update({page: 'unicorns'}, {$inc: {hits: 1}}, true);

//если установить 4-й параметр в true, то обновятся все документы

db.unicorns.update({}, {$set: {vaccinated: true }}, false, true);

*//Второй необязательный параметр у find указывает на список полей, которые мы хотим получить.*

db.unicorns.find(null, { name:true })

//получаем все поля, кроме поля name:

db.unicorns.find({}, {name: 0})

*//сортировка по убыванию*

db.unicorns.find().sort({weight: -1})

*// сортируем по весу, но получаем 2 и 3 по весу единорога, пропуская 1-го*

db.unicorns.find().sort({weight: -1}).limit(2).skip(1)

*//подсчитать кол-во единорогов на счету которых более 60 вампиров*

db.unicorns.count({vampires: {$gt: 50}})

//Когда требуется смоделировать отношения «один-ко-многим» или «многие-ко-многим» можно использовать массивы

db.employees.insert({\_id: ObjectId("4d85c7039ab0fd70a117d733"), name: 'Siona',

manager: [ObjectId("4d85c7039ab0fd70a117d730"), ObjectId("4d85c7039ab0fd70a117d732")] })

// Ищем значение в массиве manager

db.employees.find({manager: ObjectId("4d85c7039ab0fd70a117d730")})

//MongoDB поддерживает вложенные документы:

db.employees.insert({\_id: ObjectId("4d85c7039ab0fd70a117d734"), name: 'Ghanima',

family: {mother: 'Chani', father: 'Paul', brother: ObjectId("4d85c7039ab0fd70a117d730")}})

//Вложенные документы можно запрашивать с помощью точечной нотации:

db.employees.find({'family.mother': 'Chani'})

*// версия > 3.2*

db.employees.updateOne(…);

db.employees.updateMany (…);

[operator update](https://docs.mongodb.com/manual/reference/operator/update/) (документация)

*// другие команды для работы с коллекциями:*

insertOne

insertMany

*// получаем кол-во документов в коллекции*

db.cats.count()

*// модификаторы:*

*// длина массива равен 3*

{$size: 3}

*// выборка по типу*

{$type: number}

Подключение MongoDB в Node.js

*// Retrieve*

var MongoClient = require('mongodb').MongoClient;

*// Connect to the db*

MongoClient.connect("mongodb://localhost:27017/exampleDb", function(err, db) {

if(!err) {

console.log("We are connected");

}

});

[соединение с БД mongo DB](https://mongodb.github.io/node-mongodb-native/api-articles/nodekoarticle1.html#getting-that-connection-to-the-database)

Mongoose

ODM – Object-Document Mapper (объектно-документное отображение). У MongoDB нет жесткой структуры, а вот **Mongoose** позволяет нам ввести понятие схемы.

Установка и подключение (работаем через mongoose)

*//install*

$ npm install mongoose

*//use*

var mongoose = require('mongoose');

mongoose.connect('mongodb://localhost/test');

var db = mongoose.connection;

db.on('error', console.error.bind(console, 'connection error:'));

db.once('open', function() {

*// connect*

});

Установка и подключение (работаем через нативный клиент - MongoClient)

var MongoClient = require("mongodb").MongoClient;

mongoClient.connect("mongodb://localhost:27017/test", function(err, db){

if(err){

return console.log(err);

}

*// работаем с БД*

db.close();

});

Схема

**Схема** в Mongoose определяет метаданные модели - ее свойства, типы данных и ряд другой информации. [mongoosejs.com/docs/guide.html](http://mongoosejs.com/docs/guide.html)

var mongoose = require('mongoose');

var Schema = mongoose.Schema;

*// внутри перечисляем наши поля*

var blogSchema = new Schema({

title: String,

author: String,

body: String,

comments: [{ body: String, date: Date }],

date: {

type: Date,

default: Date.now,

required: [true, 'Укажите дату']

},

hidden: Boolean,

meta: {

votes: Number,

favs: Number

}

});

var Blog = mongoose.model('Blog', blogSchema);

*// ready to go!*

Типы схем

[mongoosejs.com/docs/schematypes.html](http://mongoosejs.com/docs/schematypes.html)

* [String](http://mongoosejs.com/docs/api.html#schema-string-js)
* [Number](http://mongoosejs.com/docs/api.html#schema-number-js)
* [Date](http://mongoosejs.com/docs/api.html#schema-date-js)
* [Buffer](http://mongoosejs.com/docs/api.html#schema-buffer-js) (например, для изображений)
* Boolean
* Mixed (любой тип данных)
* [Objectid](http://mongoosejs.com/docs/api.html#schema-objectid-js)
* Array

Модель

**Модели** (<http://mongoosejs.com/docs/models.html>) - это конструкторы, составленные из определения нашей схемы. Экземпляры модели представляют собой документы, которые могут быть сохранены и извлечены из нашей БД.

var schema = new mongoose.Schema({ name: 'string', size: 'string' });

var Tank = mongoose.model('Tank', schema);

Первый параметр в методе mongoose.model указывает на название модели, а второй параметр - схема.

Сохраняем объект в БД

Кроме метода save() также можно использовать метод Person.create() (см. код ниже). Первый параметр метода - сохраняемый объект.

var Person = mongoose.model('Person', yourSchema);

var subject = new Person({name: 'John'});

subject.save(function(err) {

if (err) return handkeError(err);

})

*// или*

Person.create({name: 'John'}, function(err, subject) {

if (err) return handkeError(err);

})

Поиск (find, findById, findOne)

<http://mongoosejs.com/docs/queries.html>

Методы Для получения данных:

find - возвращает все объекты, которые соответствуют условию фильтрации. find() в качестве первого параметра принимает условие фильтрации; второй параметр метода find() - функция обратного вызова, в которую передаются полученные из БД документы. Если в качестве условия фильтрации передаются пустые фигурные скобки ({}), то возвращаются все объекты.

findById - возвращает один объект по значению поля \_id. Метод возвращает документ с определенным идентификатором.

findOne - возвращает один объект, который соответствует критерию фильтрации. В отличие от метода find, метод findOne() возвращает один объект.

var Person = mongoose.model('Person', yourSchema);

*// { 'name.last': 'Ghost' } - условие*

*// 'name occupation' - выбираем нужные поля*

*// function (err, person) { ... - обрабатываем данные в callback'е*

*// find each person with a last name matching 'Ghost', selecting the `name` and `occupation` fields*

Person.findOne({ 'name.last': 'Ghost' }, 'name occupation', function (err, person) {

if (err) return handleError(err);

console.log('%s %s is a %s.', person.name.first, person.name.last, person.occupation)

*// Space Ghost is a talk show host.*

})

*// находим все*

Person.find({ 'name.last': 'Ghost' }, 'name occupation', function (err, docs) {})

*// альтернатива callback*

Person.find({ 'name.last': 'Ghost' }, 'name occupation').exec(function (err, docs) {})

Редактирование (update, findByIdAndUpdate)

Каждая модель имеет метод update(), который позволяет обновить документы в БД. **Первый параметр** метода — условие фильтрации. **Второй параметр** описывает, что и как надо изменить. В функцию обратного вызова передается результат операции.

Нередко для обновления используется фильтрация по \_id. И на этот случай мы можем использовать метод findByIdAndUpdate().

Первый параметр метода findByIdAndUpdate() - значения для поля \_id у обновляемого документа, а второй — набор новых значений для полей объекта. В функцию обратного вызова передается обновленный документ.

*// меняем Karl на Johny*

var query = {name: 'Karl'};

Model.update(query, {name: 'Johny'}, options, callback); *// все заменит на {name: 'Johny'}*

Model.update(query, {$set: {name: 'Johny'}}, options, callback); *// меняем только поле name*

Person.findOne({name: 'Johny'}, function (err, person) {

if (err) return handleError(err);

person.name = 'Johny';

person.save();

})

Удаление (remove, findOneAndRemove)

Для удаления применяется метод remove(). В метод remove() передается критерий фильтрации документов на удаление. Объект, который передается в функцию обратного вызова, содержит информацию об операции удаления.

Метод findOneAndRemove() позволяет удалить один документ. В функцию обратного вызова метод findOneAndRemove() передается удаленный документ.

И частная разновидность этого метода - удаление по полю \_id в виде метода findByIdAndRemove().

*// удаление:*

Model.remove({name: 'Johny'}, function (err, person) {

if (err) return handleError(err);

})

Валидация в Mongoose

Mongoose имеет ряд встроенных правил валидации, которые следует указывать в схеме:

* required - обязательно наличие значения для свойства.
* min и max - задают минимальное и максимальное значения для числовых данных.
* minlength и maxlength - задают минимальную и максимальную длину для строк.
* enum - строка должна представлять одно из значений в указанном массиве строк.
* match - строка должна соответствовать регулярному выражению.

Если мы попытаемся добавить некорректные данные в БД, то запрос на добавление вернет ошибку.

Пример:

const personScheme = new Schema({

name: {

type: String,

required: true,

minlength:3,

maxlength:15

},

age: {

type: Number,

required: true,

min: 1,

max:125

}

});

Promise и Mongoose

С MongoDB можно использовать Promise.

С помощью метода then мы можем получить данные, которые возвратил нам сервер и выполнить обработку результата.

user.save()

.then(function(data){

console.log("Сохранен объект", data);

mongoose.disconnect(); *// отключение от базы данных*

})

.catch(function (err){

console.log(err);

mongoose.disconnect();

});

mlab.com

[mlab.com](https://mlab.com/) - это облачный сервис по предоставлению БД mongoDB.

* Создаем БД
* Добавляем пользователя
* И подключаемся в своем приложении:

mongoose

.connect(`mongodb://user\_name:pass@ds147072.mlab.com:47072/name\_bd`);

CRUD

**CRUD** — (create, read, update, delete — «создание, чтение,обновление, удаление») - 4 основные функции, используемые при работе базами данных.

Послесловие

Node.js, а значит и Mongo, применяется для высоконагруженных проектов, там, где необходимо передавать много информации, чаты, игры.