**Лабораторная работа №5**

**Документ-ориентированная СУБД MongoDB**

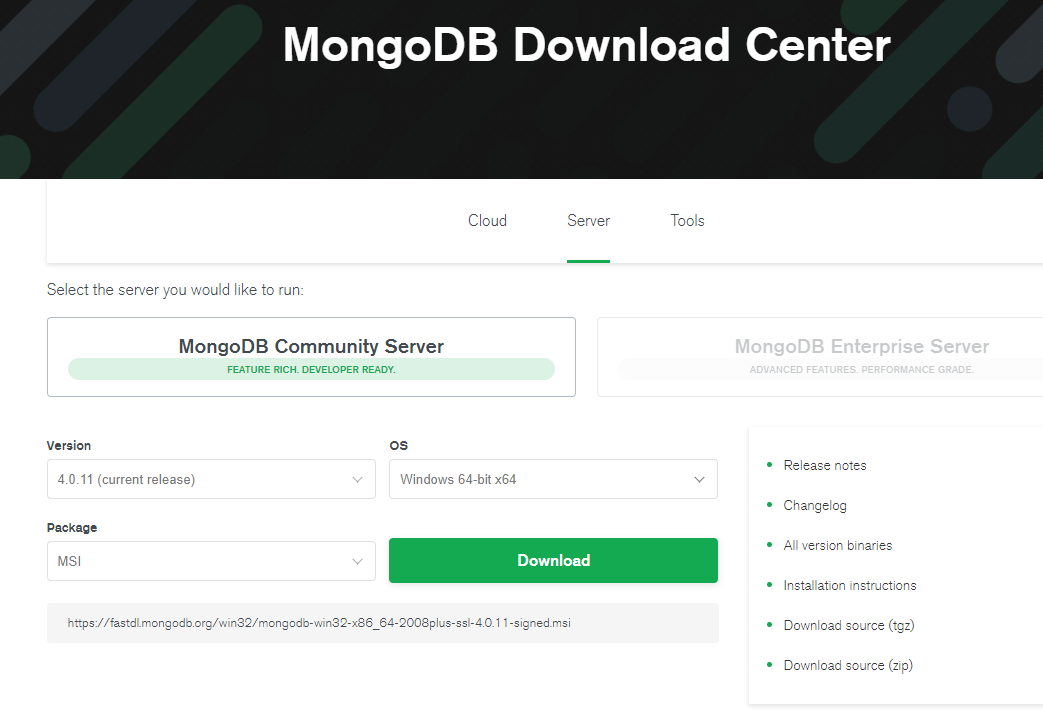
**Теоретическая часть**

**Установка MongoDB**

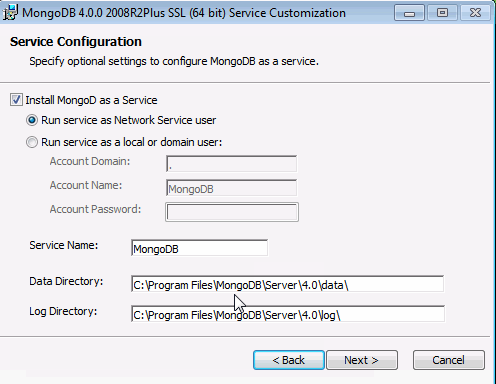
Для установки MongoDB загрузим один распространяемых пакетов с официального сайта <https://www.mongodb.com/download-center/community>.

Официальный сайт предоставляет пакеты дистрибутивов для различных платформ: Windows, Linux, MacOS, Solaris. И каждой платформы доступно несколько дистрибутивов. Причем есть два вида серверов - Community и Enterprise. В данном случае надо установить версию **Community.** Хотя Enterprise-версия обладает несколько большими возможностями, но она доступна только в триальном режиме или по подписке.

Для загрузки необходимого функционала выберем нужную операционную систему и подходящий тип пакета (рекомендуется msi).



Выполним установку сервера как службы Windows.



Если после установки мы откроем папку *C:\Program Files\MongoDB\Server\4.0\bin\,* то сможем найти содержимое сервера.

* bsondump: считывает содержимое BSON-файлов и преобразует их в читабельный формат, например, в JSON;
* mongo: представляет консольный интерфейс для взаимодействия с базами данных, своего рода консольный клиент;
* mongod: сервер баз данных MongoDB. Он обрабатывает запросы, управляет форматом данных и выполняет различные операции в фоновом режиме по управлению базами данных;
* mongodump: утилита создания бэкапа баз данных;
* mongoexport: утилита для экспорта данных в форматы JSON, TSV или CSV;
* mongofiles: утилита, позволяющая управлять файлами в системе GridFS;
* mongoimport: утилита, импорирующая данных в форматах JSON, TSV или CSV в базу данных MongoDB;
* mongorestore: позволяет записывать данные из дампа, созданного mongodump, в новую или существующую базу данных;
* mongos: служба маршрутизации MongoDB, которая помогает обрабатывать запросы и определять местоположение данных в кластере MongoDB;
* mongorestat: представляет счетчики операций с бд;
* mongotop: предоставляет способ подсчета времени, затраченного на операции чтения-записи в бд.

**Основные типы данных**

* ***Integer***  
  Используется для хранения целочисленных значений. В зависимости от сервера может быть как 32-битным, так и 64-битным.
* ***Double***  
  Используется для хранения значений с плавающей точкой
* ***Boolean***  
  Используется для хранения логических (true / false) значений.
* ***String***  
  Используется для хранения символьных строк. В MongoDB используется кодировка UTF-8.
* ***Arrays***  
  Данный тип данных используется для хранения массивов значений по одному ключу.
* ***Object***  
  Используется для встроенных документов.
* ***Symbol***  
  Используется также, как и String, но, обычно, резервируется для языков, которые используют специальные символы.
* ***Null***  
  Используется для хранения значения Null.
* ***Timestamp***  
  Используется для хранения даты и времени
* ***Min / Max***  
  Используется для сравнения значений с наибольшим и наименьшим BSON (Binary JSON) элементом.
* ***Object ID***  
  Используется для хранения ID документа.
* ***Regular Expression***  
  Используется для хранения  регулярных выражений.
* ***Code***  
  Используется для хранения JavaScript кода в документе.
* ***Binary data***  
  Данный тип данных позволяет хранить бинарные данные.
* ***Date***  
  Используется для хранения текущей даты или времени в UNIX формате.

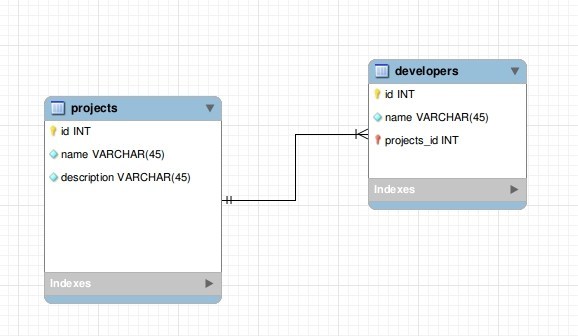
### Моделирование данных

В MongoDB данные имеют гибкую схему хранения документов в одной коллекции. Это означает, что документы не должны иметь одинаковый набор полей или структуру. Общие поля в документах коллекции могут хранить различные типы данных.

Предположим, что нам необходимо создать базу данных для хранения проектов, разработчиков, которые работают над этими проектами. Есть следующие требования:

* Каждый проект имеет уникальное имя и описание
* Каждый проект имеет много разработчиков.

Если бы мы использовали стандартную реляционную БД, то наша структура таблиц была бы, примерно, следующей:

[](https://i0.wp.com/proselyte.net/wp-content/uploads/2016/10/sqlSchemaProjects.jpg)

В MongoDB у нас будет одна коллекция проектов со следующей структурой:

*{*

*\_id: PROJECT\_ID*

*name: NAME\_OF\_PROJECT,*

*developers: [*

*{*

*name:'DEVELOPER\_NAME',*

*},*

*{*

*name: 'DEVELOPER\_NAME'*

*}*

*]*

*}*

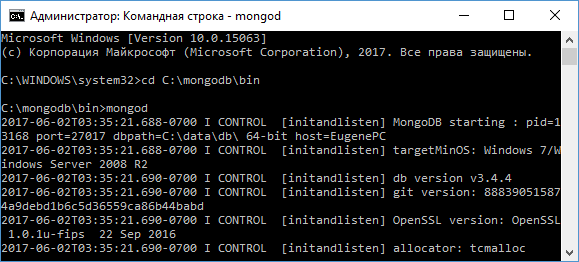
### Создание каталога для БД и запуск MongoDB

После установки надо создать на жестком диске каталог, в котором будут находиться базы данных MongoDB.

В ОС Windows по умолчанию MongoDB хранит базы данных по пути  *\data\db*, поэтому, если вы используете Windows, вам надо создать соответствующий каталог (если он не был создан автоматически). В ОС Linux и MacOS каталогом по умолчанию будет *\data\db*.

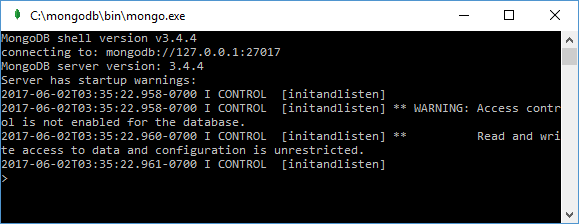
Если же возникла необходимость использовать какой-то другой путь к файлам, то его можно передать при запуске MongoDB во флаге --dbpath.

Итак, после создания каталога для хранения БД можно запустить сервер MongoDB. Сервер представляет приложение mongod, которое находится в папке bin. Для этого запустим командную строку (в Windows) или консоль в Linux и там введем соответствующие команды. Для ОС Windows это будет выглядеть так:



Командная строка отобразит нам ряд служебной информации, например, что сервер запускается на localhost на порту 27017.

И после удачного запуска сервера мы сможем производить операции с бд через оболочку mongo. Эта оболочка представляет файл mongo.exe, который располагается в выше рассмотренной папке установки. Запустим этот файл:



Это консольная оболочка для взаимодействия с сервером, через которую можно управлять данными. Второй строкой эта оболочка говорит о подключении к серверу mongod.

На сервере MongoDB может быть размещено несколько баз данных.

Чтобы просмотреть существующие базы, можно выполнить команду ***show dbs/show databases****.*

***use mydatabase*** - выбор текущей базы. ***mydatabase*** - имя вашей базы. Если базы не существует, она будет создана. После выполнения ***use*** создается специальный объект с именем ***db,*** через который осуществляется доступ к текущей базе.

Список всх команд, которые можно выполнить с базой, можно получить, используя команду ***db.help().***

Каждая база состоит из коллекций документов.

**Коллекция** - это аналог таблицы в реляционной СУБД. Главным отличием является отсутствие фиксированного списка колонок. Команда ***show collections*** позволяет получить список коллекций базы данных.

В базе данных создать collection можно командой: ***db.createCollection('name\_collection').***

Коллекции состоят из документов.

**Документ** - это аналог записи в реляционной СУБД. Документ содержит произвольный набор полей, в том числе некоторые из них могут быть объектами и массивами. Каждый документ обязательно содержит поле **\_id,** значение которого должно быть уникальным в пределах коллекции. После создания записи изменить значение этого поля нельзя. Вы можете сами создавать это поле. Если при создании записи вы не указали значение поля ***\_id***, то оно будет сгенерировано автоматически.

**CRUD-операции**

**Вставка**

***db.collectionname.insert(doc)*** - добавляет один или несколько документов в коллекцию. Документ должен быть представлен в формате JSON.

Например:

*db.unicorns.insert({*

*name: 'Aurora',*

*gender: 'f',*

*weight: 450})*

Метод ***save()*** работает таким же образом, как и метод ***insert(),*** в случае, если мы не указываем ***\_id***. Если же мы указываем данное поле, тогда документ перезапишет все данные документа с таким ***\_id.***

Для загрузки датасета используется команда ***load()***, позволяющая загрузить JS файл.

*load(“books.js")*

Также можно выполнить импорт **JSON-файла** c использованием утилиты **mongoimport.**

Пример импорта файла contacts.json в коллекцию contacts БД users.

*mongoimport --db users --collection contacts --file contacts.json*

**Выборка**

***db.collectionname.find(query, projection)*** - упрощенный аналог SQL-оператора select. Параметр ***query*** задает условие выборки; ***projection*** - список полей в выборке.

Пример: найти всех пользователей, старше 27 лет.

*db.users.find( { age: 27 } );*

Пример: найти все документы, в массиве фруктов которых, есть и яблоки, и бананы.

*db.food.find( { fruits: { $all : [ "apple", "banana" ] } } );*

Пример задания проекции – ограничение полей выборки (только специальности, идентификатор выводится по умолчанию).

*db.developers.find( {}, {"specialty": 1})*

*{ "\_id" : ObjectId("5815f4e735ee883c37ac8b4d"), "specialty" : "Java" }*

*{ "\_id" : ObjectId("5815f4ec35ee883c37ac8b4e"), "specialty" : "Java" }*

*{ "\_id" : ObjectId("5815f4f535ee883c37ac8b4f"), "specialty" : "C++" }*

Для осуществления красивого вывода, можно дополнительно использовать метод ***pretty().***

**Изменение**

***db.collection.update(query, update, options)***- изменяет документы, удовлетворяющие условию ***query*** (аналогично тому, что используется в find).

Параметр ***update***может содержать:

* документ, на который будут заменены все найденные документы,
* описание изменения документа с помощью операторов ***$inc***и ***$set.***

Пример изменения записи с \_id=1:

*db.books.update(*

*{ \_id: 1 },*

*{*

*$inc: { stock: 5 },*

*$set: {*

*item: "ABC123",*

*"info.publisher": "2222",*

*tags: [ "software" ],*

*"ratings.1": { by: "xyz", rating: 3 }*

*}*

*}*

*)*

**Удаление**

***db.collectionname.remove(query)*** - удаляет документы, соответствующие условию. Параметр ***query***аналогичен соответствующему параметру в методе find().

Пример удаления всех записей в коллекции:

*db.unicorns.remove( { } )*

Пример удаления толстых единорогов (всех, у кого вес больше 300):

*db.unicorns.remove( { weight: { $gt: 300 } } )*

**Aggregation Framework в MongoDB**

Для составления сложных запросов в MongoDB имеется встроенный Aggregation Framework. Запрос выполняется методом ***db.collectionname.aggregate().*** Действия, выполняемые в запросе, описываются с помощью операторов агрегации.

Рассмотрим основные стадии:

* **$project**

Аналог списка полей в операторе SQL select. Позволяет добавить новые поля или убрать существующие.

Синтаксис: *{ $project: { <specifications> } }*

Варианты описания полей:

* + *<field>:* *<1 or true>* - указывает, что данное поле датасета необходимо включить в выборку.
  + *\_id*: *<0 or false>*- удаляет из выборки поле \_id (по умолчанию это поле включается во все выборки, удалять другие поля таким способом нельзя).
  + *<field>: <expression>* - добавление нового поля, которого нет в исходных данных или изменение значения существующего поля.
* **$match**

Аналог SQL-конструкции where. Задает условия, по которым фильтруются документы. Аналогичен условиям в методе find().

Синтаксис: *{ $match: { <query> } }*

* **$limit**

Аналог SQL-конструкции limit. Ограничивает количество выбираемых документов.

Синтаксис: *{ $limit: <positive integer> }*

* **$skip**

Аналог SQL-конструкции offset. Позволяет пропустить несколько первыйх документов.

Синтаксис: *{ $skip: <positive integer> }*

* **$unwind**

Разбивает поле, содержащее массив.

Синтаксис: *{ $unwind: <field path> }*

Пример:

Исходные данные:

*{ "\_id" : 1, "item" : "ABC1", sizes: [ "S", "M", "L"] }*

Команда:

*db.inventory.aggregate( [ { $unwind : "$sizes" } ] )*

Результат:

*{ "\_id" : 1, "item" : "ABC1", "sizes" : "S" }*

*{ "\_id" : 1, "item" : "ABC1", "sizes" : "M" }*

*{ "\_id" : 1, "item" : "ABC1", "sizes" : "L" }*

* **$group**

Группирует документы по признаку, заданному в параметре \_id и вычисляет аккумулирующие выражения (суммы, максимумы и минимумы, первое или последнее значение, собирает свойства из нескольких документов в один массив и т.д.)

Синтаксис:

*{ $group: { \_id: <expression>, <field1>: { <accumulator1> : <expression1> }, ... } }*

* **$sort**

Сортитует результаты, аналог SQL-конструкции order by.

Синтаксис:

*{ $sort: { <field1>: <sort order>, <field2>: <sort order> ... } }*

Рассмотрим работу фреймворка на примере датасета, содержащего почтовые индексы городов США. Датасет содержит записи такого вида:

*{*

*"\_id": "10280",*

*"city": "NEW YORK",*

*"state": "NY",*

*"pop": 5574,*

*"loc": [*

*-74.016323,*

*40.710537*

*]*

*}*

* \_id - индекс в виде строки.
* city - название города, у города может быть больше одного индекса.
* state - сокращенное обозначение штата.
* pop - численность населения.
* loc - географические координаты.

Города с численностью больше 10 млн.

*db.zipcodes.aggregate( [*

*{ $group: { \_id: "$state", totalPop: { $sum: "$pop" } } },*

*{ $match: { totalPop: { $gte: 10\*1000\*1000 } } }*

*] )*

Среднее по штатам:

*db.zipcodes.aggregate( [*

*{ $group: { \_id: { state: "$state", city: "$city" }, pop: { $sum: "$pop" } } },*

*{ $group: { \_id: "$\_id.state", avgCityPop: { $avg: "$pop" } } }*

*] )*

Самый большой и самый маленький город:

*db.zipcodes.aggregate( [*

*{ $group:*

*{*

*\_id: { state: "$state", city: "$city" },*

*pop: { $sum: "$pop" }*

*}*

*},*

*{ $sort: { pop: 1 } },*

*{ $group:*

*{*

*\_id : "$\_id.state",*

*biggestCity: { $last: "$\_id.city" },*

*biggestPop: { $last: "$pop" },*

*smallestCity: { $first: "$\_id.city" },*

*smallestPop: { $first: "$pop" }*

*}*

*},*

*// the following $project is optional, and*

*// modifies the output format.*

*{ $project:*

*{ \_id: 0,*

*state: "$\_id",*

*biggestCity: { name: "$biggestCity", pop: "$biggestPop" },*

*smallestCity: { name: "$smallestCity", pop: "$smallestPop" }*

*}*

*}*

*] )*

**Задания к лабораторной работе**

**Задание 1**

1. Установите MongoDB.
2. Подключитесь к серверу через mongo shell (командная строка)
3. Создайте базу данных и загрузите датасет mails.json
4. Изучите структуру датасета в графической среде Mongo Compass.
5. Выполните запросы в mongo shell:
   1. Для каждого дня недели посчитайте, сколько писем было отправлено на адрес [ebass@enron.com](mailto:ebass@enron.com)
   2. Покопаемся в переписке Shanna Husser и Eric Bass. Сколько писем каждый из них отправил другому?
   3. Сколько различных отправителей в датасете?
   4. В какой папке больше всего писем?
   5. В какой день недели отправлено максимальное количество писем?
   6. Laurie Ellis иногда посылает письма с одинаковыми темами. Для каждой темы письма, использованной в 2000 году, посчитайте, сколько раз она была использована.
   7. Сколько человек отправляют письма сами себе?

**Задание 2**

1. Разработайте базу данных, предназначенную для хранения кинотеки фильмов. Обеспечьте хранение следующих данных (не менее 10 документов:

* название фильма;
* год;
* рейтинг (PG-13, R и др.);
* жанры (может быть несколько);
* режиссер (может быть несколько);
* сценаристы (может быть несколько);
* актеры (может быть несколько);
* награды (количество побед, количество номинаций, информация о количестве номинаций и побед в текстовом виде) – может присутствовать, а может отсутствовать.

1. Выполните следующие запросы:
   1. Измените рейтинг произвольного фильма и добавьте к списку актеров Вас и Вашего лучшего друга;
   2. Напишите запрос на поиск фильма, в котором снимались Вы и Ваш друг;
   3. Найти фильмы, которые были номинированы, но не получили награды, а затем удалите их.