

Содержание

СОКРАЩЕНИЯ, ПРИНЯТЫЕ В ПОСОБИИ.....	5
ВВЕДЕНИЕ.....	6
1. ОБЗОР НЕРЕЛЯЦИОННЫХ БАЗ ДАННЫХ.....	7
1.1. Базы данных, в которых информация хранится по принципу <i>Ключ-Значение (Key-Value)</i>	7
1.2. Базы данных, являющиеся клонами СУБД BigTable	8
1.3. Документо-ориентированные базы данных.....	9
1.4. Нереляционные базы данных, основанные на графах	9
1.5. Объектно-ориентированные базы данных	9
2. СУБД MONGODB	10
2.1. Теоретическая часть СУБД MongoDB	10
2.1.1. Шесть основных концепций СУБД MongoDB	10
2.1.2. Язык запросов MongoDB	11
Вставка нового документа	11
Типы данных	12
Извлечение результатов	13
Объекты и переменные JavaScript	13
Регулярные выражения	14
Сортировка результатов.....	14
Ограничение выборки по полям документа	15
Ограничение выборки по количеству документов	16
Условная выборка.....	17
Логические операторы	17
Работа со списками.....	19

Обновление документов	20
Удаление документов	21
Команда MapReduce	22
2.2 Функции работы с MongoDB на ЯП Python	23
2.2.1. Назначение функций	23
2.2.2. Создание базы данных	23
2.2.3. Удаление базы данных	24
2.2.4. Создание коллекции и заполнение БД.....	24
3. ПРАКТИЧЕСКАЯ ЧАСТЬ СУБД MONGODB	26
3.1. Установка MongoDB.....	26
3.2. Графический интерфейс для MongoDB	29
3.3. Установка графического интерфейса	30
3.3.1. MongoDB Compass	30
3.3.2. Studio 3T	31
3.4. Примеры работы с функциями на ЯП Python	32
3.5. Пример создания базы данных “Зенковки”	34
ЛИТЕРАТУРА	37
ПРИЛОЖЕНИЕ 1	39
Установка MongoDB.....	39
Содержимое пакета MongoDB	39
Создание каталога для БД и запуск MongoDB	40
Установка драйверов MongoDB	42
ПРИЛОЖЕНИЕ 2	43
Функции работы с СУБД MongoDB на ЯП Python.....	43

Сокращения, принятые в пособии

NoSQL	Not only SQL
No RDBMS	Нереляционная база данных
БД	База данных
БЗ	База знаний
ИМ	Интерфейсный модуль
ИПС	Информационно-поисковая система
САПР ТП	система автоматизации проектирования технологических процессов
ЯП	Язык программирования.

Введение

Данное учебно-методическое пособие разработано в рамках курсов «Базы данных в ТПС» по направлению подготовки «Технология приборостроения» и «Системы баз данных» по направлению подготовки «Информатика и вычислительная техника». Материалы пособия основаны на знаниях полученных студентами при изучении дисциплин «информатика», «программирование». Пособие предназначено для подготовки лекций преподавателями и выполнения лабораторных и самостоятельных работ студентами.

Структура методического пособия представлена:

Глава 1 — проведён краткий обзор нереляционных баз данных;

Глава 2 — описана Теоретическая часть СУБД MongoDB;

Глава 3 — перечень и описание практических работ с рекомендацией по их выполнению;

В приложении 1 и 2 описаны вспомогательные материалы для работы с СУБД MONGODB.

Каждая глава пособия соответствует теме проводимого семинара в рамках отводимых на преподавание учебных часов.

Изучив данное пособие, студенты будут:

- иметь представление об основных возможностях СУБД MongoDB;
- знать основные приемы создания баз данных;
- уметь применять инструментарий СУБД MongoDB для выполнения самостоятельных работ, курсовых и дипломных проектов.

В работе описаны инструментальные программные средства и методы построения структур данных при создании САПР ТП с применением нереляционной СУБД MongoDB. Предлагается базовый пакет программ и алгоритмов, который можно развивать путем подключения новых функций. Применение инструментальных средств рассчитано на интегрированную среду разработки IDLE. Предполагается, что читатель знаком с данной средой и языками программирования Python [21,22] и JavaScript [19, 20]. Программный код, представленный в данном пособии, написан на языке программирования Python 3.4.

Данное пособие базируется на работе "Применения нереляционной СУБД MongoDB в САПР ТП" (<https://books.ifmo.ru/file/pdf/2135.pdf>).

1. Обзор нереляционных баз данных

Нереляционные базы данных не являются чем-то новым в системах обработки данных. Первые такие базы появились еще во времена первых мэйнфреймов и до сих пор используются в специализированных хранилищах документов и службах каталогов.

Термин «No RDBMS» («нереляционная база данных») не прижился, и сейчас для всех нереляционных баз данных используется сокращение NoSQL («отрицание SQL»).

Причиной появления нереляционных баз данных в первую очередь стало резкое увеличение объемов информационных хранилищ и усложнение связей между документами. Информация в базах данных перестала быть изолированной и стала менее структурированной. Каждая страница в Интернет ссылается на множество других, товары в Интернет-магазине могут иметь десятки параметров, по которым необходимо производить сложный поиск т.д. Иногда вообще невозможно жестко описать структуру хранимой информации.

Реляционные СУБД не могут эффективно использоваться в такой разнородной информационной среде. Попытки «натянуть» слабо структурированную информацию на реляционную СУБД приводят к бесконечному увеличению количества служебных таблиц, огромным неэффективным индексам и мизерной производительности. В итоге получается сложная в доработке и абсолютно немасштабируемая система.

Хотя реляционные базы данных до сих пор широко используются, как пример можно привести базу данных MySQL, но для проектов с большой нагрузкой и сложным поиском они используются все реже.

Новые NoSQL базы данных появляются постоянно, но все их можно разделить на четыре основные категории:

1.1. Базы данных, в которых информация хранится по принципу *Ключ-Значение (Key-Value)*

Самые простые по организации NoSQL хранилища. Такие базы данных представляют собой хэш-таблицы, в которых каждому значению соответствует уникальный ключ. Базы данных этого типа имеют большие ограничения по вариантам запросов, но быстро обрабатывают большие массивы данных. Самыми популярными Key-Value СУБД являются Dynomite [11], Voldemort [12], Tokyo Cabinet [13] и Redis [14].

Redis – нереляционная база данных с открытым кодом, основанная на принципе «Key-Value». В качестве модели данных используется словарь, в котором хранятся ключи и связанные с ними значения. Одним из

преимуществ Redis перед другими базами такого типа является то, что значения ключей не ограничено строками. База данных поддерживает следующие типы данных:

- Строковые списки
- Коллекции строк
- Упорядоченные множества строк
- Словари

Redis поддерживает множество языков программирования, таких как, C#, Java, JavaScript, PHP и т.д.

1.2. Базы данных, являющиеся клонами СУБД BigTable

База данных BigTable разработана компанией Google для внутреннего использования [BigTable]. Именно в ней хранятся проиндексированные Интернет-страницы, сообщения электронной почты Gmail и других сервисов Google. По принципу организации база данных представляет собой таблицу данных с тремя параметрами (измерениями): колонка, строка и временная метка. Эта архитектура позволяет базе данных работать с большой производительностью и легко масштабируется на несколько серверов.

BigTable базы не поддерживают многих функций реляционных баз данных. Например, нет запросов типа Join¹, сложные запросы типа Select и т.д. Официально база данных BigTable не распространяется, поэтому на рынке есть множество баз данных, разработанных на ее основе. Самыми известными являются Hypertable [5], Hadoop[6] и Cassandra [7].

Cassandra – нереляционная база данных ориентированная на построение масштабируемых и отказоустойчивых информационных хранилищ, в которых информация хранится в виде хеша.

Изначально база данных разрабатывалась компанией Facebook и в 2009 передана Apache Software Foundation для дальнейшего развития. Системы на базе Cassandra работают в компаниях Cisco, Twitter, IBM. Самая крупная система на базе Cassandra состоит из 400 серверов и база данных более 400Тб.

Для работы с базой данных создан собственный язык запросов CQL (Cassandra Query Language), напоминающий SQL по синтаксису, но с урезанной функциональностью. Язык позволяет выполнять простые запросы типа Select, создавать индексы с поддержкой пространств имен и семейств столбцов. Язык CQL имеет пакеты поддержки для Java, JavaScript, PHP, Ruby и Python.

¹ JOIN — оператор языка SQL, который является реализацией операции соединения реляционной алгебры. Входит в раздел FROM операторов SELECT, UPDATE или DELETE.

1.3. Документо-ориентированные базы данных

Достаточно «молодой» вид нереляционных СУБД. По организации данных такие СУБД напоминают базы типа *Key-Value*.

Базы данных специально разработаны для работы с иерархическими структурами данных (документами). Хранилище данных имеет структуру дерева – главный (корневой) узел, от которого отходит множество дополнительных узлов. Каждый узел содержит информацию о добавленном документе и формирует индексы базы данных, позволяющие быстро находить требуемый документ или его часть. В отличие от баз данных типа Key-Value более рационально использует ресурсы оборудования.

Документы могут быть дополнительно организованы в *коллекции*. Коллекции можно считать аналогом таблиц в реляционных СУБД, но в отличие от них, коллекции могут быть вложены друг в друга для более быстрой и гибкой индексации. Набор документов в коллекции может быть произвольным, но разработчики рекомендуют объединять в коллекции документы похожей структуры.

Документо-ориентированные базы в основном используются в документном поиске и издательских системах. Самыми популярными СУБД этого типа являются CouchDB [15], Berkeley DB [16], eXist [17] и MongoDB [18].

1.4. Нереляционные базы данных, основанные на графах

Этот тип баз данных основан на теории графов и отлично подходит для управления объектами, имеющими сложные взаимосвязи. Программно организация данных представляет собой узлы, связанные друг с другом ссылками. Каждый узел и ссылка могут иметь дополнительные атрибуты для ускорения поиска и индексации информации. Наиболее распространенными базами данных этого типа являются AllegroGraph [8], Sones graphDB [9] и Neo4j [10].

1.5. Объектно-ориентированные базы данных

Такие базы данных тоже можно отнести к категории нереляционных баз данных. Этот тип баз данных имеет узкую сферу применения и обычно используется только совместно с объектно-ориентированными языками программирования, такими как C#.

NoSQL имеет разные значения, и для каждого означает что-то свое. Для некоторых это система, которая участвует в хранении данных, для других NoSQL дает уверенность в том, что сам процесс сохранения данных делится

на разновидности. Большинство разработчиков программного обеспечения действуют по принципу «все в одном». А NoSQL, наоборот, для каждого процесса выделяет только один действительно нужный инструмент. NoSQL — это альтернативная и открытая технология, в состав которой входят множество дополнительных шаблонов для лучшего и более простого управления данными.

2. СУБД MongoDB¹

Как документо-ориентированная СУБД MongoDB это довольно-таки обобщённое NoSQL решение. Её также можно рассматривать как альтернативу реляционным СУБД. Подобно реляционным СУБД, она также может выигрышно дополняться более специализированными NoSQL решениями. Термины MongoDB и Mongo используются как синонимы.

2.1. Теоретическая часть СУБД MongoDB

Теоретическая часть данного пособия основывается прежде всего на книге разработчика MongoDB Karl Seguin [1]. Предполагается, что пользователь, изучающий MongoDB, знаком с основами реляционных баз данных [4]. СУБД MongoDB поддерживается языками программирования Perl, C/C++, Erlang, PHP, Java, Scala, Python, .NET, Haskell, Ruby, Javascript.

2.1.1. Шесть основных концепций СУБД MongoDB

1. MongoDB — концептуально то же самое, что обычная привычная нам база данных. Внутри MongoDB может быть ноль или более баз данных, каждая из которых является контейнером для прочих сущностей.
2. База данных может иметь ноль или более «коллекций». Коллекция настолько похожа на традиционную «таблицу», что можно смело считать их одним и тем же.
3. Коллекции состоят из нуля или более «документов». Опять же, документ можно рассматривать как «строку».
4. Документ состоит из одного или более «полей», которые подобны «колонкам».
5. «Индексы» в MongoDB почти идентичны таковым в реляционных базах данных.
6. «Курсоры» отличаются от предыдущих пяти концепций, но они очень важны (хотя порой их обходят вниманием) и заслуживают отдельного

¹ В дальнейшем по тексту MongoDB

обсуждения. Важно понимать, что, когда мы запрашиваем у MongoDB какие-либо данные, то она возвращает курсор, с которыми мы можем делать все, что угодно — подсчитывать, пропускать определённое число предшествующих записей — при этом не загружая сами данные.

Подведем итог: MongoDB состоит из «баз данных», которые состоят из «коллекций». «Коллекции» состоят из «документов». Каждый «документ» состоит из «полей». «Коллекции» могут быть проиндексированы, что улучшает производительность выборки и сортировки. И, наконец, получение данных из MongoDB сводится к получению «курсора», который отдаёт эти данные по мере надобности.

2.1.2. Язык запросов MongoDB

СУБД MongoDB относится к NoSQL базам данных, основной чертой которых является нереляционный характер и соответственно язык запросов, отличный от SQL. В MongoDB в качестве язык запросов используется **JavaScript** и JSON-структуры [3]. Выбор столь нехарактерного языка запроса объясняется тем, что эта документо-ориентированная СУБД

использует JSON-формат для представления документов и вывода результатов. Физически JSON-структуры хранятся в бинарном BSON-формате.

Вставка нового документа

Внутри текущей базы данных создаются коллекции, вставка нового документа в коллекцию при помощи метода `insert` приводит к её автоматическому созданию. В качестве аргумента, метод `insert` принимает JSON-объект, который служит телом документа. Ниже в базу данных вставляются документы с единственным ключом `title`, в качестве значения которого выступают названия различных СУБД.

```
db.mybase.insert({title: "MySQL"})
db.mybase.insert({title: "PostgreSQL"})
db.mybase.insert({title: "MongoDB"})
db.mybase.insert([{"title: "MS SQL"}, {"title: "Oracle"}])
```

Формат JSON допускает создание, как единичных объектов, так и массивов. Записи `"MySQL"`, `"PostgreSQL"` и `"MongoDB"` вставляются как отдельные вставки, в то время как `"MS SQL"` и `"Oracle"` при помощи массива, элементы которого в JSON заключаются в квадратные скобки. Для каждого документа автоматически создается идентификатор `"_id"`, который можно задавать вручную, с единственным условием, чтобы он оставался уникальным в рамках текущей базы данных. Идентификатор, который

формируется по умолчанию, формируется по следующему алгоритму: в старших четырех байтах находится время создания записи в формате UNIXSTAMP, следующие три байта — идентификатор компьютера, следующие два — идентификатор процесса, последние три — локальный счетчик процесса.

Основная особенность MongoDB заключается в том, что документ или запись может быть настолько сложной по структуре, насколько допускает формат JSON. В качестве примера, в отдельной базе данных **articles** создадим набор статей, имеющий название и набор ключевых слов.

```
db.articles.insert({title: "Обзор NoSQL решений", tags:
["NoSQL", "MongoDB", "memcached", "CouchDB", "Riak",
"HBase", "Redis"]})
db.articles.insert({title: "MongoDB", tags: ["NoSQL",
"MongoDB"]})
db.articles.insert({title: "Redis", tags: ["NoSQL",
"Redis"]})
```

Здесь помимо поля **title**, создается поле **tags**, представляющего собой массив ключевых слов. Элементы массивов и значения могут сами быть составными JSON-объектами и массивами. В результате одна запись в базе данных может представлять собой сложную структуру и значительно отличаться по своему составу от соседней (в отличие от традиционных реляционных баз данных, в которых структура записей в таблице строгая).

Типы данных

MongoDB поддерживает следующие типы данных:

- **string** — строки должны быть представлены в кодировке UTF-8;
- **числа**
 - **double** — 8-байтные числа с плавающей точкой IEEE;
 - **int** — 4-байтное целое число;
 - **long** — 8-байтное целое число;
- **datetime** — календарный тип для хранения даты и времени, представляющий собой 8-байтное целое число, хранящее количество миллисекунд, прошедших с 1 января 1970 года;

При вставке чисел через JavaScript следует иметь в виду, что JavaScript поддерживает только числовой тип **Number**, соответствующий типу **double**. Поэтому для явной вставки целых чисел типа **int** и **long** следует использовать классы **NumberInt()** и **NumberLong()**, соответственно.

```
db.numbers.save({num: NumberLong(5)});
```

Извлечение результатов

Для подсчета количества документов в текущей базе данных можно воспользоваться методом `count()`.

Пример применения:

```
db.mybase.count()
```

Результат: 5

Для того, чтобы извлечь результаты, можно воспользоваться методом `find()`

```
db.mybase.find()
{ "_id" : ObjectId("51eb8c2bb4d7d4d898b05fce"), "title" :
"MySQL" }
{ "_id" : ObjectId("51eb905ab4d7d4d898b05fcf"), "title" :
"PostgreSQL" }
{ "_id" : ObjectId("51eb9061b4d7d4d898b05fd0"), "title" :
"MongoDB" }
{ "_id" : ObjectId("51eb907db4d7d4d898b05fd1"), "title" :
"MS SQL" }
{ "_id" : ObjectId("51eb907db4d7d4d898b05fd2"), "title" :
"Oracle" }
```

Методы, допускают использование селекторов, например, для извлечения документа, соответствующего MySQL методу `find()` можно передать следующий JSON-объект:

```
db.mybase.find({"title" : "MySQL"})
{ "_id" : ObjectId ("51eb9bb1303d105141c7d74b"), "title" :
"MySQL" }
```

Аналогично можно подсчитать количество статей с названием "MySQL", передав методу `count` селектор `"title" : "MySQL"`.

Пример применения:

```
db.mybase.count({"title" : "MySQL"})
```

Результат: 1

Объекты и переменные JavaScript

Консоль `mongo` позволяет не только задействовать предопределенные объекты, но и вводить свои. Ниже вводится JavaScript-объект `where`, в котором формируется условие поиска, далее в метод `find()` может передаваться объект.

```

var where = {}
where['title'] = "MySQL"
db.mybase.find(where)
{ "_id" : ObjectId("51eba53f303d105141c7d751"), "title" :
  "MySQL" }

```

Консоль поддерживает и более сложные приемы JavaScript-программирования, например, создание собственных методов.

Регулярные выражения

В качестве селектора могут выступать не только строки, но и регулярные выражения, например, для извлечения всех записей поле **title**, которые начинаются с символа **M** можно воспользоваться регулярным выражением `/^M/` передав его методу `find()`.

```

db.mybase.find({title: /^M/});
{ "_id" : ObjectId("51f4dae823d2a4ef32d25ec4"), "title" :
  "MySQL" }
{ "_id" : ObjectId("51f4dae823d2a4ef32d25ec6"), "title" :
  "MongoDB" }
{ "_id" : ObjectId("51f4dae923d2a4ef32d25ec7"), "title" : "MS
  SQL" }

```

Сортировка результатов

Для сортировки результатов используется метод `sort()`, который принимает в качестве параметров JSON-структуру, ключом в которой выступает название сортируемого поля, а в качестве значения выступает целое число: положительное — прямая сортировка, отрицательная — обратная.

```

db.mybase.find({title: /^M/}).sort({title: 1});
{ "_id" : ObjectId("51f53e56a631742542dda011"), "title" :
  "MS SQL" }
{ "_id" : ObjectId("51f53e55a631742542dda010"), "title" :
  "MongoDB" }
{ "_id" : ObjectId("51f53e55a631742542dda00e"), "title" :
  "MySQL" }
db.mybase.find({title: /^M/}).sort({title: -1});
{ "_id" : ObjectId("51f53e55a631742542dda00e"), "title" :
  "MySQL" }
{ "_id" : ObjectId("51f53e55a631742542dda010"), "title" :
  "MongoDB" }
{ "_id" : ObjectId("51f53e56a631742542dda011"), "title" :
  "MS SQL" }

```

Сортировать можно по нескольким полям одновременно. Создадим таблицу с двумя полями **fst** и **snd**:

```

db.sortexmpl.insert({fst: 2, snd: 5})
db.sortexmpl.insert({fst: 1, snd: 12})
db.sortexmpl.insert({fst: 2, snd: 2})
db.sortexmpl.insert({fst: 1, snd: 20})
db.sortexmpl.insert({fst: 2, snd: 7})
db.sortexmpl.insert({fst: 1, snd: 6})

```

Ниже приводится пример сортировки по двум полям одновременно:

```

db.sortexmpl.find()
{ "_id" : ObjectId("51f56e07a631742542dda016"), "fst" : 2,
  "snd" : 5 }
{ "_id" : ObjectId("51f56e07a631742542dda017"), "fst" : 1,
  "snd" : 12 }
{ "_id" : ObjectId("51f56e07a631742542dda018"), "fst" : 2,
  "snd" : 2 }
{ "_id" : ObjectId("51f56e07a631742542dda019"), "fst" : 1,
  "snd" : 20 }
{ "_id" : ObjectId("51f56e07a631742542dda01a"), "fst" : 2,
  "snd" : 7 }
{ "_id" : ObjectId("51f56e07a631742542dda01b"), "fst" : 1,
  "snd" : 6 }
db.sortexmpl.find().sort({fst: 1, snd: 1})
{ "_id" : ObjectId("51f56e07a631742542dda01b"), "fst" : 1,
  "snd" : 6 }
{ "_id" : ObjectId("51f56e07a631742542dda017"), "fst" : 1,
  "snd" : 12 }
{ "_id" : ObjectId("51f56e07a631742542dda019"), "fst" : 1,
  "snd" : 20 }
{ "_id" : ObjectId("51f56e07a631742542dda018"), "fst" : 2,
  "snd" : 2 }
{ "_id" : ObjectId("51f56e07a631742542dda016"), "fst" : 2,
  "snd" : 5 }
{ "_id" : ObjectId("51f56e07a631742542dda01a"), "fst" : 2,
  "snd" : 7 }

```

Ограничение выборки по полям документа

По умолчанию выборка содержит все поля документа, однако, в том случае, если требуется выбрать лишь конкретные поля, методам `find()` и `findOne()` можно передавать второй аргумент в виде JSON-структуры, с ключами, совпадающими с названиями столбцов и значениями 1, если поле должно попадать в выборку и 0, если его необходимо исключить из выборки. В следующем запросе извлекаются только названия `title`, идентификатор `_id` исключается из выборки:

```

db.mybase.find({title: /^М/}, {title: 1, _id:
0}).sort({title: 1});
{ "title" : "MS SQL" }

```

```
{ "title" : "MongoDB" }
{ "title" : "MySQL" }
```

Более того, не обязательно указывать включаемые поля, достаточно перечислить поля, которые не должны попасть в выборку:

```
db.mybase.find({title: /^M/}, {_id: 0}).sort({title: 1});
{ "title" : "MS SQL" }
{ "title" : "MongoDB" }
{ "title" : "MySQL" }
```

Если в условии нет необходимости, то в качестве первого запроса метода `find()` передается пустой селектор:

```
db.mybase.find({}, {_id: 0}).sort({title: 1});
{ "title" : "MS SQL" }
{ "title" : "MongoDB" }
{ "title" : "MySQL" }
{ "title" : "Oracle" }
{ "title" : "PostgreSQL" }
```

Ограничение выборки по количеству документов

Для того чтобы извлечь лишь одно значение из полученной выборки, можно воспользоваться методом `findOne()`.

```
db.mybase.findOne({title: /^M/});
{ "_id" : ObjectId("51f4dae823d2a4ef32d25ec4"), "title" :
"MySQL" }
```

Однако к методу `findOne()` прибегают чаще в тех ситуациях, когда ожидается, что результирующая коллекция будет содержать лишь один документ. В тех же случаях, когда следует ограничить выборку несколькими документами, рекомендуется использовать метод `limit()`, который принимает в качестве аргумента количество извлекаемых документов. Следующий запрос вернёт только 2 статьи из коллекции `mybase`.

```
db.mybase.find({title: /^M/}).sort({title: 1}).limit(2);
{ "_id" : ObjectId("51f53e56a631742542dda011"), "title" :
"MS SQL" }
{ "_id" : ObjectId("51f53e55a631742542dda010"), "title" :
"MongoDB" }
```

Для организации постраничной навигации может потребоваться метод `skip()`, который позволяет пропустить в выборке количество аргументов, указанных в его параметре. Так, чтобы извлечь содержимое следующей «страницы»:

```
db.mybase.find({title: /^M/}).sort({title:
1}).skip(0).limit(2);
{ "_id" : ObjectId("51f53e56a631742542dda011"), "title" :
```

```

"MS SQL" }
{ "_id" : ObjectId("51f53e55a631742542dda010"), "title" :
"MongoDB" }
db.mybase.find({title: /^M/}).sort({title:
1}).skip(2).limit(2);
{ "_id" : ObjectId("51f53e55a631742542dda00e"), "title" :
"MySQL" }

```

При использовании больших значений в качестве параметра метода `skip()` следует помнить, методу приходится пропускать это значение записей, прежде чем добраться до извлекаемых. В этом случае разумно отказаться от `skip()` и использовать операторы сравнения.

Условная выборка

Создадим коллекцию `lang`, содержащую документы, состоящие из двух полей, `title` — название базы данных и `query` — используемый ею язык запросов.

```

db.lang.insert({title: "MySQL", query: "SQL"})
db.lang.insert({title: "PostgreSQL", query: "SQL"})
db.lang.insert({title: "MongoDB", query: "JavaScript"})

```

Как уже упоминалось ранее, для выбора условия необходимо передавать JSON-структуру методам `find()`, `findOne()`, `count()` и т.д. Следующий метод извлекает все базы данных, начинающиеся с символа `М`.

```

db.lang.find({title: /^M/})
{ "_id" : ObjectId("51f55285a631742542dda013"), "title" :
"MySQL", "query" : "SQL" }
{ "_id" : ObjectId("51f55286a631742542dda015"), "title" :
"MongoDB", "query" : "JavaScript" }

```

Логические операторы

JSON-структура в методе `find()`, может содержать несколько полей, при этом условия объединяются по логике **AND (И)**. В следующем примере выбираются базы данных, начинающиеся с символа `М`, и использующие JavaScript в качестве языка запросов.

```

db.lang.find({query: "JavaScript", title: /^M/})
{ "_id" : ObjectId("51f55286a631742542dda015"), "title" :
"MongoDB", "query" : "JavaScript" }

```

Для формирования логики **OR (ИЛИ)** предназначен оператор `$or`, который оперирует массивом аргументов как запрос, извлекающий

записи **или** начинающиеся с символа **М**, **или** использующие в качестве языка запроса SQL может выглядеть следующим образом:

```
db.lang.find({$or: [{query: "SQL"}, {title: /^М/}]})
{ "_id" : ObjectId("51f55285a631742542dda013"), "title" :
"MySQL", "query" : "SQL" }
{ "_id" : ObjectId("51f55285a631742542dda014"), "title" :
"PostgreSQL", "query" : "SQL" }
{ "_id" : ObjectId("51f55286a631742542dda015"), "title" :
"MongoDB", "query" : "JavaScript" }
```

В таблице 1 представлены операторы сравнения и логические операторы MongoDB.

Операторы из таблицы 1 можно использовать как по отдельности, так и в комбинации.

Таблица 1. Операторы MongoDB

Оператор SQL	MongoDB	Описание
<	\$lt	Меньше
<=	\$lte	Меньше или равно
>	\$gt	Больше
>=	\$gte	Больше или равно
<>	\$ne	Не равно
NOT	\$not	Отрицание
EXISTS	\$exists	Проверка существования поля
OR	\$or	Или
NOT OR	\$nor	Не или
RLIKE, REGEXP	\$regex	Соответствие регулярному выражению
LIKE	\$elemMatch	Соответствие всех полей вложенного документа
-	\$size	Соответствие размеру массива
-	\$type	Соответствие, если поле имеет указанный тип

```
db.lang.find({_id: {$gt:
ObjectId("51f55285a631742542dda013")}})
{ "_id" : ObjectId("51f55285a631742542dda014"), "title" :
"PostgreSQL", "query" : "SQL" }
{ "_id" : ObjectId("51f55286a631742542dda015"), "title" :
"MongoDB", "query" : "JavaScript" }
db.lang.find({_id: {$lt:
ObjectId("51f55285a631742542dda015")}})
{ "_id" : ObjectId("51f55285a631742542dda013"), "title" :
"MySQL", "query" : "SQL" }
{ "_id" : ObjectId("51f55285a631742542dda014"), "title" :
"PostgreSQL", "query" : "SQL" }
```



```

db.lang.find({_id: {$gt:
ObjectId("51f55285a631742542dda013"), $lt:
ObjectId("51f55285a631742542dda015")}})
{ "_id" : ObjectId("51f55285a631742542dda014"), "title" :
"PostgreSQL", "query" : "SQL" }

```

В оболочке mongo можно составлять более сложные программы. Так, например, двойное условие из предыдущего примера можно оформить в виде объекта, подставляемого в JSON-структуру.

```

var conditions = {}
conditions['$gt'] = ObjectId("51f55285a631742542dda013")
ObjectId("51f88c41b49ab634fd78bf93")
conditions['$lt'] = ObjectId("51f55285a631742542dda015")
ObjectId("51f88c41b49ab634fd78bf95")
db.lang.find(conditions)
db.lang.find({_id: conditions})
{ "_id" : ObjectId("51f55285a631742542dda014"), "title" :
"PostgreSQL", "query" : "SQL" }

```

Работа со списками

В таблице 2 показаны операторы, предназначенные для работы со списками.

Таблица 2. Операторы для работы со списками

Оператор SQL	MongoDB	Описание
IN	\$in	Входит в список
NOT IN	\$nin	Не входит в список
ALL	\$all	Одновременное совпадение набора элементов

Оператор \$in извлекает записи, для которых заданное значение совпадает хотя бы с одним из списка. Ниже выводится запрос, извлекающий записи, поле title которых совпадает с одним из значений в списке ['MongoDB', 'MySQL']:

```

db.lang.find({title: {$in: ['MongoDB', 'MySQL']}})
{ "_id" : ObjectId("51f55285a631742542dda013"), "title" :
"MySQL", "query" : "SQL" }
{ "_id" : ObjectId("51f55286a631742542dda015"), "title" :
"MongoDB", "query" : "JavaScript" }

```

Для извлечения всех записей, не входящих в список ['MongoDB', 'MySQL'] можно воспользоваться оператором:

```
$nindb.lang.find({title: {$nin: ['MongoDB', 'MySQL']}})
{ "_id" : ObjectId("51f88c41b49ab634fd78bf94"), "title" :
"PostgreSQL", "query" : "SQL" }
```

Обновление документов

Для обновления используется метод `update`, первый аргумент которого определяет список обновляемых документов, второй — как отобранные документы модифицируются. Следующая команда добавляет в документ с названием "MongoDB" дополнительное поле `nosql`, со значением "true":

```
db.mybase.update({"title": "MongoDB"}, {$set: {nosql:
"true"}})
db.mybase.find()
{ "_id" : ObjectId("51eb9bb1303d105141c7d74b"), "title" :
"MySQL" }
{ "_id" : ObjectId("51eb9bb1303d105141c7d74c"), "title" :
"PostgreSQL" }
{ "_id" : ObjectId("51eb9bb2303d105141c7d74e"), "title" :
"MS SQL" }
{ "_id" : ObjectId("51eb9bb2303d105141c7d74f"), "title" :
"Oracle" }
{ "_id" : ObjectId("51eb9bb1303d105141c7d74d"), "nosql" :
"true", "title" : "MongoDB" }
```

Если обновлению подвергается только одно поле, следует обязательно использовать оператор `$set`. Если его опустить, вместо обновления поля, будет обновлен весь документ:

```
db.mybase.update({"title": "MongoDB"}, {nosql: "true"})
db.mybase.find()
{ "_id" : ObjectId("51eba53f303d105141c7d751"), "title" :
"MySQL" }
{ "_id" : ObjectId("51eba53f303d105141c7d752"), "title" :
"PostgreSQL" }
{ "_id" : ObjectId("51eba53f303d105141c7d754"), "title" :
"MS SQL" }
{ "_id" : ObjectId("51eba53f303d105141c7d755"), "title" :
"Oracle" }
{ "_id" : ObjectId("51f9582829b045085952815f"), "nosql" :
"true" }
```

Для удаления дополнительного поля `nosql` вместо оператора `$set` следует подставить `$unset`

```
> db.mybase.update({"title": "MongoDB"}, {$unset: {nosql:
"none"}})
> db.mybase.find({"title": "MongoDB"})
{ "_id" : ObjectId("51eb9bb1303d105141c7d74d"), "title" :
"MongoDB" }
```

Для того чтобы добавить ключевое слово в массив тэгов **tags** базы данных **articles** следует воспользоваться оператором **\$addToSet**

```
db.articles.update({tags: "Redis"}, {$addToSet: {tags: "Обзор"}})
```

Если запросить состав текущей базы данных при помощи метода **find()**, то можно заметить, что обновился только один документ из двух, которые имеют ключевое слово **"Redis"**. Для того, чтобы обновить все документы, следует передать четвертому параметру метода **update()** значение **true**, что заставит MongoDB обновить все найденные документы.

```
db.articles.update({tags: "Redis"}, {$addToSet: {tags: "Обзор"}}, false, true)
```

Ниже приводится сводная таблица 3 по операторам, используемым совместно с методом **update()**.

*Таблица 3. Операторы, используемые совместно с методом **update()***

Оператор	Описание
\$set	Обновление или создание поля
\$unset	Удаление поля
\$inc	Увеличение значения поля на заданное число
\$pop	Удаляет последний (или первый) элемент массива
\$pushAll	Помещает несколько элементов в массив
\$push	Помещает новый элемент в массив
\$addToSet	Помещает новый элемент в массив (исключаются дубликаты)
\$pull	Удаляет из массива значение (при его наличии)
\$pullAll	Удаляет из массива все подходящие значения

Удаление документов

Полностью удалить документы из текущей базы данных можно при помощи метода **remove()**.

```
db.mybase.remove()
```

Точно так же, как и другие методы, в качестве параметра метод может принимать селекторы, например, удалить все записи с ключом **nosql** равным **"true"** можно при помощи следующей команды

```
db.mybase.remove({nosql: "true"})
```

Команда MapReduce

В настоящее время в соответствии с требованиями новых информационных технологий создаются и функционируют многие системы управления, связанные с необходимостью отображения информации на электронной карте:

- ✓ Геоинформационные системы (ГИС);
- ✓ Системы федерального и муниципального управления;
- ✓ Системы проектирования;
- ✓ Системы военного назначения и т.д.

Эти системы управления регулируют деятельность технических и социальных систем, функционирующих в некотором операционном пространстве (географическом, экономическом и т.п.) с явно выраженной пространственной природой.

Геоинформационные технологии предназначены для широкого внедрения в практику методов и средств работы с пространственно-временными данными, представляемыми в виде системы электронных карт, и предметно-ориентированных сред обработки разнородной информации для различных категорий пользователей.

Геоинформационная система (*Географическая информационная система, ГИС*) — система сбора, хранения, анализа и графической визуализации пространственных (географических) данных и связанной с ними информации о необходимых объектах.

Геоинформационная система может включать в свой состав пространственные базы данных (в том числе, под управлением универсальных СУБД), редакторы растровой и векторной графики, различные средства пространственного анализа данных. Для этих целей используется технология MapReduce - фреймворк для вычисления некоторых наборов распределенных задач с использованием большого количества компьютеров (называемых «нодами»), образующих **кластер**.

Работа MapReduce состоит из двух шагов: Map и Reduce, названных так по аналогии с одноименными функциями высшего порядка, map и reduce.

Пример применения

Канонический пример приложения, написанного с помощью MapReduce, — это процесс, подсчитывающий, сколько раз различные слова встречаются в наборе документов:

```
// функция, используемая рабочими нодами на Map-шаге
// для обработки пар ключ-значение из входного потока
void map(String name, String document):
// Входные данные:
```

```

//  name - название документа
//  document - содержимое документа
for each word w in document:
    EmitIntermediate(w, «1»);

// функция, используемая рабочими нодами на Reduce-шаге
// для обработки пар ключ-значение, полученных на Map-шаге
void reduce(String word, Iterator partialCounts):
// Входные данные:
//  word - слово
//  partialCounts - список группированных промежуточных
результатов. Количество записей в partialCounts и есть
//  требуемое значение
int result = 0;
for each v in partialCounts:
    result += parseInt(v);
Emit(AsString(result));

```

В этом коде на Map-шаге каждый документ разбивается на слова, и возвращаются пары, где ключом является само слово, а значением — «1». Если в документе одно и то же слово встречается несколько раз, то в результате предварительной обработки этого документа будет столько же этих пар, сколько раз встретилось это слово.

Библиотека объединяет все пары с одинаковым ключом и передает их на вход функции reduce, которой остается сложить их, чтобы получить общее количество вхождений данного слова во все документы.

2.2 Функции работы с MongoDB на ЯП Python

Для работы с БД, созданной в среде MongoDB, разработана группа функций на языке программирования Python (Приложение 2).

2.2.1. Назначение функций

- **mdbCreate** (NameDB, ...) – создание базы данных;
- **mdbDelete** (NameDB, ...) – удаление базы данных;
- **mdbRead** (Collection, ...) – чтение данных по значению поля;
- **mdbWrite** (Collection, ...) – запись данных в базу;
- **mdbUpdate** (Collection, ...) – обновление документа.

2.2.2. Создание базы данных

Название функции: **mdbCreate** (NameDB, Connection)

Параметры функции:

- NameDB – имя создаваемой БД;

• Connection – объект соединения (англ. – connection object) – результат работы функции `pymongo.Connection`;

Тип возвращаемого значения: `pymongo.database.Database`

Пример применения:

```
db = mdbCreate ("testDB", connection)
print ("База данных " + db.name + " создана")
```

2.2.3. Удаление базы данных

Название функции: `mdbDelete(NameDB, Connection)`

Параметры функции:

- NameDB – имя удаляемой БД;
- Connection – объект соединения (англ. – connection object) – результат работы функции `pymongo.Connection`

Тип возвращаемого значения: `pymongo.database.Database`

Пример применения:

```
db = mdbDelete("testDB", connection)
print ("База данных " + db.name + " удалена")
```

2.2.4. Создание коллекции и заполнение БД

Последующие функции работают не с БД в целом, а с конкретной коллекцией.

Создание коллекции

```
collection = db['university']
# Тестовые данные.
collection.save({"name": "Сергей", "surname": "Иванов",
"gender": "m", "age": 20})
collection.save({"name": "Виктор", "surname": "Андреев",
"gender": "m", "age": 19, "hobbies": ["basketball",
"programming"]})
collection.save({"name": "Николай", "surname": "Николаев",
"gender": "m", "age": 21})
```

Чтение данных по значению поля

Название функции:

`mdbRead(Collection, Key, Value)`

Параметры функции:

- Collection – коллекция, в которой производится поиск;
- Key – ключ, по которому ведется поиск;
- Value – значение ключа;

Тип возвращаемого значения: `pymongo.cursor.Cursor`

Пример применения:

```

key = input ("Введите ключ для поиска: ")
value = input ("Введите значение ключа для поиска: ")
res = mdbRead(collection, key, value)
print ("Результат поиска:")
for men in res:
    print (men["surname"] + " " + men["name"])

```

Запись данных в базу

Название функции:

```

mdbWrite(Collection, Dict)

```

Параметры функции:

- **Collection** – коллекция, в которой производится поиск;
- **Dict** – словарь с данными

Тип возвращаемого значения: –

Пример применения:

```

mdbWrite(collection, {"name": "Василий", "surname": "Тёркин",
"age": 26})
res = mdbRead(collection, "surname", "Тёркин")
for men in res:
    print (men)

```

Результат работы функций: {'surname': 'Тёркин', 'age': 26, '_id': ObjectId('552c47cfacblaa0ec8f77be2'), 'name': 'Василий'}

Обновление документа

Название функции:

```

mdbUpdate(Collection, Key, KeyValue, Field, FieldValue)

```

Параметры функции:

- **Collection** – коллекция, в которой производится поиск;
- **Key** – ключ для поиска записи;
- **KeyValue** – значение ключа;
- **Field** – записываемое поле;
- **FieldValue** – записываемое значение

Тип возвращаемого значения: –

Пример применения:

```

key = input("Введите ключ для редактирования: ")
key_value = input ("Введите значение ключа для
редактирования: ")
print ("Исходные документы:")
res = mdbRead(collection, key, key_value)
for men in res:
    print (men)
field = input ("Введите поле для редактирования: ")
field_value = input("Введите новое значение поля: ")
res = mdbUpdate(collection, key, key_value, field,

```

```
field_value)
```

Пример применения:

```
print ("Результат редактирования:")
res = mdbRead(collection, field, field_value)
for men in res:
    print (men)
```

Результат работы функций:

Введите ключ для редактирования: name

Введите значение ключа для редактирования: Анна

Исходные документы:

```
{'_id': ObjectId('552c4d0facb1aa18586f058d'), 'surname':
'Соколова', 'name': 'Анна', 'age': 21, 'gender': 'f'}
```

Введите поле для редактирования: name

Введите новое значение поля: Аня

Результат редактирования:

```
{'_id': ObjectId('552c4d0facb1aa18586f058d'), 'surname':
'Соколова', 'name': 'Аня', 'age': 21, 'gender': 'f'}
```

В качестве дальнейшего развития можно предложить расширение списка функций и усовершенствование возможностей (расширение области применения) уже написанных функций.

3. Практическая часть СУБД MongoDB

3.1. Установка MongoDB

Загрузим дистрибутив консоли MongoDB с официального сайта СУБД [2]. Следует отметить, что данный ресурс предлагает на выбор набор различных версий СУБД для всех популярных операционных систем. В качестве версии для работы, будем использовать последнюю - 3.6.4 (Рис. 1). Если нужно выбрать другую версию, нужно зайти в пункт **Release notes** и выбрать нужную.

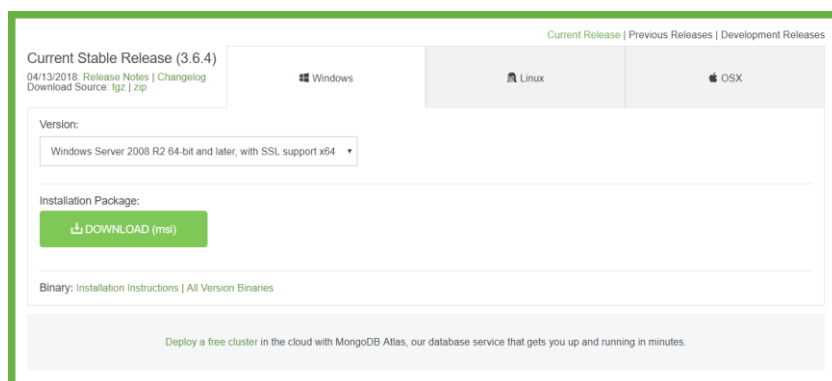


Рисунок 1. Выбор версии MongoDB на сайте

Скачаем установочный файл (кнопка **DOWNLOAD** в центре страницы). По окончании загрузки запустим установочный файл и выберем удобную директорию для установки (в нашем случае – **C:\mongodb**).

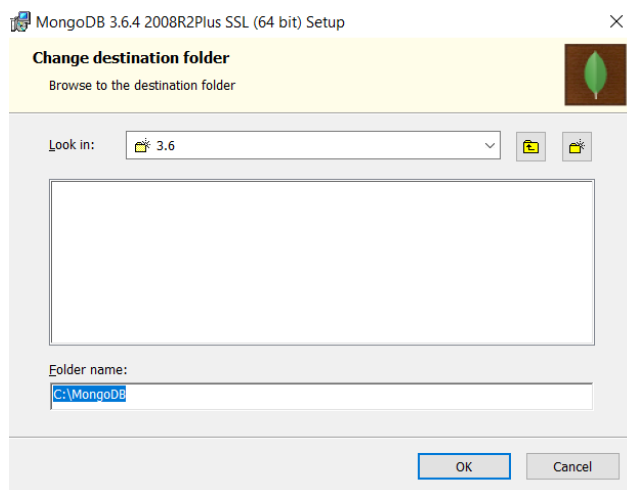


Рисунок 2. Установка MongoDB

Теперь можно осуществить запуск **сервера** и начать работу с СУБД. Оба исполняемых файла находятся в упомянутой выше директории **bin**:

- **mongod** – сервер;
- **mongo** – клиентская консоль.

Запустить сервер и клиентскую консоль можно через командную строку или просто запустить **mongod.exe/mongo.exe** из директории **bin**.

Рассмотрим первый способ (Рис. 3). По умолчанию сервер использует директорию **C:\data\db** для своей работы. Для изменения директории следует создать файл с именем **mongodb.config** и в нем указать желаемое место для работы, например: **dbpath=c:\mongodb\data**, однако такой способ работает только в запуске сервера из командной строки, который будет рассмотрен позже. В данном примере будем использовать место по умолчанию. Создаем папку **data** и в ней создаем папку **db**. Заходим в директорию, где хранится сервер и клиентская консоль. Сначала запускаем сервер (**mongod.exe**), потом клиентскую консоль (**mongo.exe**). Результат запуска на Рис. 4-5.

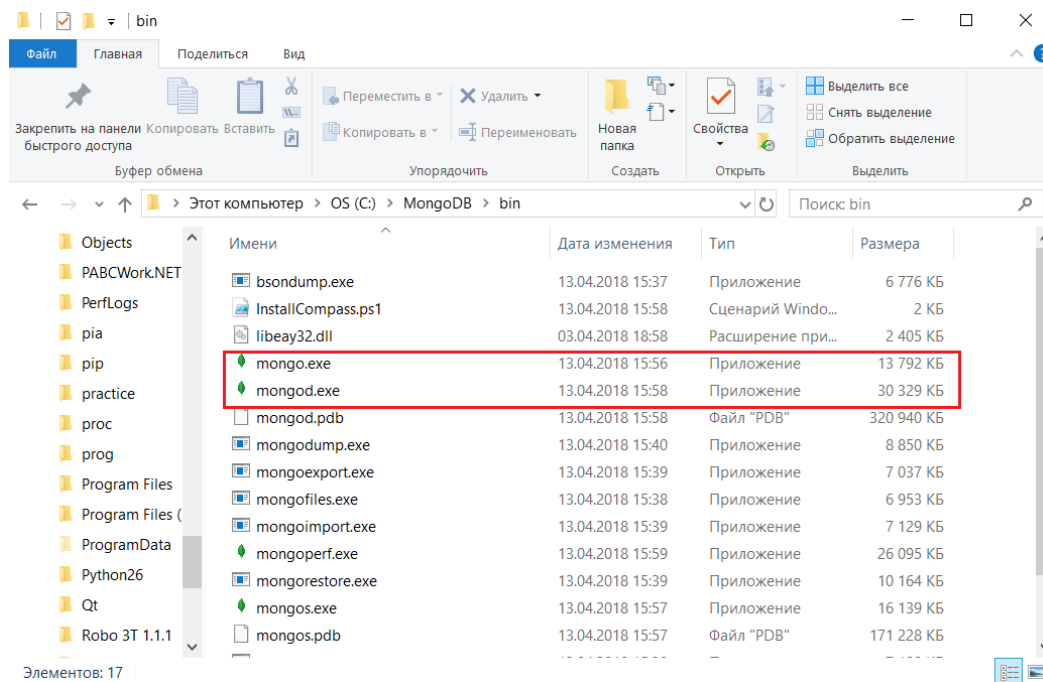


Рисунок 3. Запуск через директорию

Второй способ запуска нуждается в использовании командной строки. Для запуска сервера вызовем `mongod` с параметром:

--config/path/to/your/mongodb.config (Рис. 4). В нашем случае команда будет выглядеть следующим образом: **`c:\mongodb\bin\mongod.exe --config c:\mongodb\bin\mongodb.config`**

В случае использования места по умолчанию команда будет выглядеть следующим образом: **`c:\mongodb\bin\mongod.exe`**



Рисунок 4. Запуск через командную строку

Если все сделано правильно, сервер запустится, и командная строка будет выглядеть, как показано на рис. 5. Теперь можно подключаться к серверу. Для этого просто запустим файл **mongo.exe** из директории **bin**. При этом откроется клиентская консоль, как показано на рис. 6.

Изменить место сохранения базы данных можно не только через создание файла конфигурации. Для этого требуется использовать функцию: **--dbpath X:\data**. Как и функцию **--config**, **--dbpath** можно применить только при запуске сервера с помощью командной строки. В нашем случае команда будет выглядеть следующим образом:

`c:\mongodb\bin\mongod.exe --dbpath c:\mongodb\data`

```
C:\MongoDB\bin\mongod.exe
nrestricted.
2018-05-06T07:02:47.781-0700 I CONTROL [initandlisten]
2018-05-06T07:02:47.783-0700 I CONTROL [initandlisten] ** WARNING: This server is bound to localhost.
2018-05-06T07:02:47.784-0700 I CONTROL [initandlisten] ** Remote systems will be unable to connect to this server.
2018-05-06T07:02:47.785-0700 I CONTROL [initandlisten] ** Start the server with --bind_ip <address> to specify
2018-05-06T07:02:47.787-0700 I CONTROL [initandlisten] ** which IP
2018-05-06T07:02:47.787-0700 I CONTROL [initandlisten] ** addresses it should serve responses from, or with --
2018-05-06T07:02:47.788-0700 I CONTROL [initandlisten] ** bind_ip_all to
2018-05-06T07:02:47.788-0700 I CONTROL [initandlisten] ** bind to all interfaces. If this behavior is desired,
2018-05-06T07:02:47.789-0700 I CONTROL [initandlisten] ** start the
2018-05-06T07:02:47.789-0700 I CONTROL [initandlisten] ** server with --bind_ip 127.0.0.1 to disable this warn
2018-05-06T07:02:47.790-0700 I CONTROL [initandlisten] ing.
2018-05-06T07:02:47.790-0700 I CONTROL [initandlisten]
2018-05-06T07:02:47.799-0700 I CONTROL [initandlisten]
2018-05-06T07:02:47.802-0700 I CONTROL [initandlisten] ** WARNING: The file system cache of this machine is configured
2018-05-06T07:02:47.802-0700 I CONTROL [initandlisten] to be greater than 40% of the total memory. This can lead to increased memory pressure and poor performance.
2018-05-06T07:02:47.803-0700 I CONTROL [initandlisten] See http://dochub.mongodb.org/core/wt-windows-system-file-cache
2018-05-06T07:02:47.805-0700 I CONTROL [initandlisten]
2018-05-06T07:02:48.166+0300 W FTDC [initandlisten] Failed to initialize Performance Counters for FTDC: WindowsPdhEr
2018-05-06T07:02:48.166+0300 W FTDC [initandlisten] ror: PdhExpandCounterPathW failed with 'Указанные объекты не найдены на этом компьютере.' for counter '\Memory\Available
2018-05-06T07:02:48.166+0300 I FTDC [initandlisten] Bytes'
2018-05-06T07:02:48.166+0300 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory 'C
2018-05-06T07:02:48.172+0300 I NETWORK [initandlisten] :/data/db/diagnostic.data'
2018-05-06T07:02:48.172+0300 I NETWORK [initandlisten] waiting for connections on port 27017
2018-05-06T07:02:51.385+0300 I NETWORK [listener] connection accepted from 127.0.0.1:53914 #1 (1 connection now open)
2018-05-06T07:02:51.389+0300 I NETWORK [conn1] received client metadata from 127.0.0.1:53914 conn1: { application: { na
2018-05-06T07:02:51.389+0300 I NETWORK [conn1] me: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", version: "3.6.4" }, os: { type: "Windows", name: "Micro
2018-05-06T07:02:51.389+0300 I NETWORK [conn1] soft Windows 10", architecture: "x86_64", version: "10.0 (build 16299)" }
```

Рисунок 5. Функционирующий Сервер

```
C:\MongoDB\bin\mongo.exe
MongoDB shell version v3.6.4
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.4
Server has startup warnings:
2018-05-06T07:02:47.780-0700 I CONTROL [initandlisten]
2018-05-06T07:02:47.781-0700 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-05-06T07:02:47.781-0700 I CONTROL [initandlisten] ** Read and write access to data and configuration is u
2018-05-06T07:02:47.781-0700 I CONTROL [initandlisten] nrestricted.
2018-05-06T07:02:47.781-0700 I CONTROL [initandlisten]
2018-05-06T07:02:47.783-0700 I CONTROL [initandlisten]
2018-05-06T07:02:47.783-0700 I CONTROL [initandlisten] ** WARNING: This server is bound to localhost.
2018-05-06T07:02:47.784-0700 I CONTROL [initandlisten] ** Remote systems will be unable to connect to this server.
2018-05-06T07:02:47.785-0700 I CONTROL [initandlisten] ** Start the server with --bind_ip <address> to specify
2018-05-06T07:02:47.787-0700 I CONTROL [initandlisten] ** which IP
2018-05-06T07:02:47.787-0700 I CONTROL [initandlisten] ** addresses it should serve responses from, or with --
2018-05-06T07:02:47.788-0700 I CONTROL [initandlisten] ** bind_ip_all to
2018-05-06T07:02:47.788-0700 I CONTROL [initandlisten] ** bind to all interfaces. If this behavior is desired,
2018-05-06T07:02:47.789-0700 I CONTROL [initandlisten] ** start the
2018-05-06T07:02:47.789-0700 I CONTROL [initandlisten] ** server with --bind_ip 127.0.0.1 to disable this warn
2018-05-06T07:02:47.790-0700 I CONTROL [initandlisten] ing.
2018-05-06T07:02:47.790-0700 I CONTROL [initandlisten]
2018-05-06T07:02:47.799-0700 I CONTROL [initandlisten]
2018-05-06T07:02:47.802-0700 I CONTROL [initandlisten]
2018-05-06T07:02:47.802-0700 I CONTROL [initandlisten] ** WARNING: The file system cache of this machine is configured
2018-05-06T07:02:47.803-0700 I CONTROL [initandlisten] to be greater than 40% of the total memory. This can lead to increased memory pressure and poor performance.
2018-05-06T07:02:47.803-0700 I CONTROL [initandlisten] See http://dochub.mongodb.org/core/wt-windows-system-file-cache
2018-05-06T07:02:47.805-0700 I CONTROL [initandlisten]
>
```

Рисунок 6. Клиентская консоль после начала работы

Дальнейшая работа с MongoDB будет сводиться к написанию различных команд в клиентской консоли.

3.2. Графический интерфейс для MongoDB

Графический интерфейс пользователя (англ. Graphical User Interface, GUI) – разновидность пользовательского интерфейса, в котором элементы интерфейса, представлены пользователю на дисплее, в виде графических

изображений. В отличие от интерфейса командной строки, в GUI пользователь имеет произвольный доступ ко всем видимым экранным объектам (элементам интерфейса) и осуществляет непосредственное манипулирование ими. Использование графического интерфейса для MongoDB – самый простой способ исследовать и манипулировать данными.

3.3. Установка графического интерфейса

В качестве примера рассмотрим установку двух популярных графических интерфейсов для MongoDB:

3.3.1. MongoDB Compass

MongoDB Compass является официальным графическим интерфейсом для СУБД MongoDB. Загрузить данный GUI можно с сайта разработчика [3]. На сайте потребуется выбрать ОС и версию продукта (по умолчанию стоит последняя и стабильная версия) и пройти регистрацию. После выполнения вышеперечисленных действий начнется загрузка установочного файла. После завершения загрузки следует запустить данный файл. Процесс установки полностью автоматизирован и при его завершении запустится MongoDB Compass. После ознакомления с возможностями данного графического интерфейса можно начать работу. Внешний вид представлен на рис. 7.

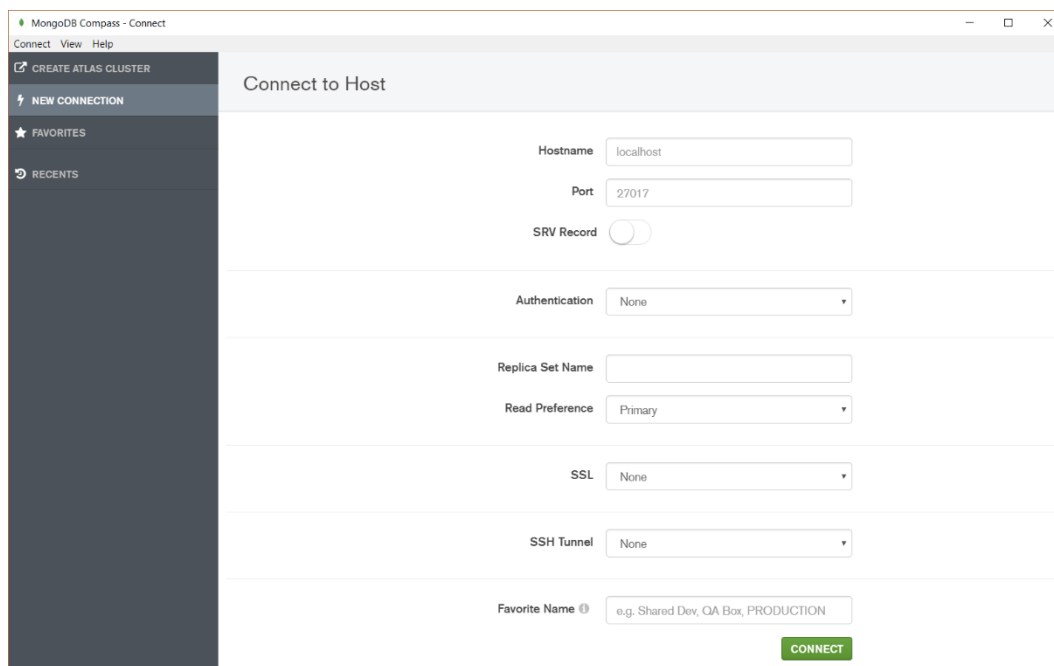


Рисунок 7. Внешний вид MongoDB Compass

Для начала работы потребуется подключиться к активному серверу либо создать локальный сервер (как было показано в пункте 3.1.). С подробным описанием функций данного графического интерфейса можно ознакомиться в официальной документации на сайте разработчика [4].

3.3.2. Studio 3T

Studio 3T - графический интерфейс для СУБД MongoDB от компании 3T Software Labs GmbH. Загрузить данный GUI можно с сайта разработчика [5]. Как и при загрузке MongoDB Compass, на сайте потребуется выбрать ОС и версию продукта, и пройти регистрацию. После выполнения вышеперечисленных действий начнется загрузка архива с установочным файлом. После завершения загрузки следует распаковать архив в любой удобной директории и запустить установочный файл. Установка состоит из нескольких шагов, в которых пользователю следует ознакомиться лицензионным соглашением и выбрать директорию для установки. Внешний вид представлен на рис. 8.

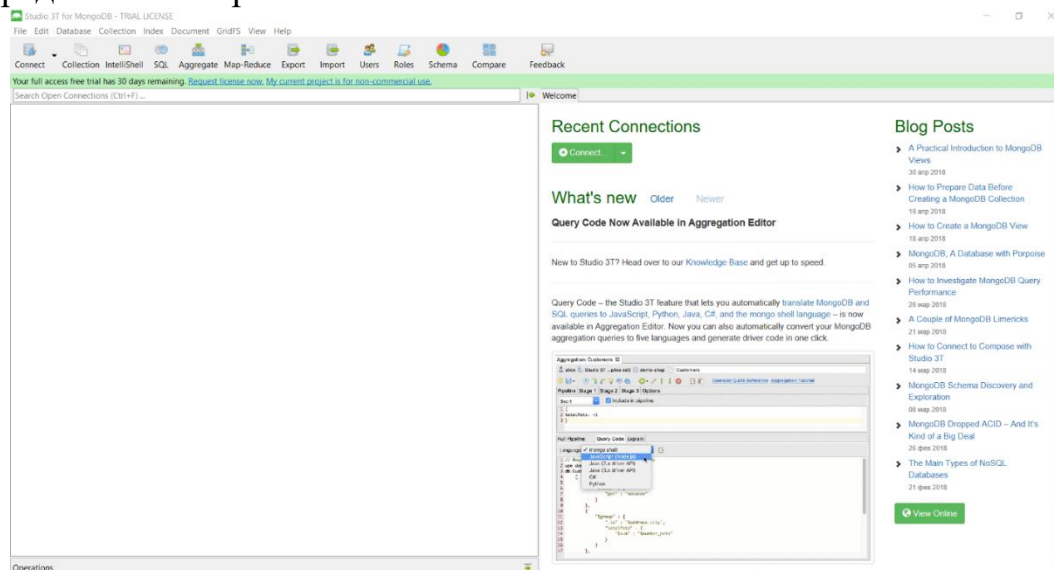


Рисунок 8. Внешний вид Studio 3T

Для начала работы потребуется подключиться к активному серверу либо создать локальный сервер (как было показано в п.3.1.). Основное отличие данного графического интерфейса от MongoDB, заключается в большем количестве функций, но доступ к ним требует покупки полной версии.

В качестве ознакомления можно 30 дневную пробную версию.

3.4. Примеры работы с функциями на ЯП Python

Имеются следующие функции:

- `mdbCreate (NameDB, ...)` – создание базы данных;
- `mdbDelete (NameDB, ...)` – удаление базы данных;
- `mdbRead (Collection, ...)` – чтение данных по значению поля;
- `mdbWrite (Collection, ...)` – запись данных в базу;
- `mdbUpdate (Collection,...)` – обновление документа.

Используем их для работы с БД.

Ход работы

1. Создадим базу данных. Для этого воспользуемся функцией `mdbCreate(NameDB, Connection)`, где:

`NameDB` – имя создаваемой БД;

`Connection` – объект соединения.

```
db = mdbCreate("PatronTable", connection)
print("База данных " + db.name + " создана")
```

2. Удалим базу данных

Используем для этого `mdbDelete(NameDB, Connection)`, где:

`NameDB` – имя удаляемой БД;

`Connection` – объект соединения.

```
db = mdbDelete ("PatronTable", connection)
print("База данных " + db.name + " удалена")
```

3. Создадим коллекцию согласно таблице 4.

Таблица 4. Патрон для нарезания резьбы плашкой

Обозначение, V_OB	Конус Морзе хвостового патрона, V_NKH	Диаметр посадочной штулки, V_DPO	Доп. вылет патрона, V_VI
6610-0021	3	38.0	199
6610-0022	4	45.0	222
6610-0023	4	55.0	237
6610-0024	4	65.0	257

Создание коллекции

```
collection = db['Patron']
```

 # Тестовые данные.

```
collection.save({"V_OB ": "6610-0021", " V_NKH ": 3, " V_DPO ":  
38.0, " V_VI ": 199})
```

```
collection.save({"V_OB ": "6610-0022", " V_NKH ": 4, " V_DPO ":  
45.0, " V_VI ": 222})
```

```
collection.save({"V_OB ": "6610-0023", " V_NKH ": 4, " V_DPO ":  
55.0, " V_VI ": 237})
```

```
collection.save({"V_OB ": "6610-00243", " V_NKH ": 4, " V_DPO ":  
65.0, " V_VI ": 257})
```

Теперь коллекция Patron создана.

Заполним коллекцию документами в соответствии с приведенной ниже таблицей до конца.

Таблица 5. Документы коллекции Patron

V_OB	V_NKH	V_DPO	V_VI
6610-0021	3	38	199
6610-0022	4	45	222
6610-0023	4	55	237
6610-0024	4	65	257
...
...
...
6610-0048	8	76	270
6610-0049	9	76	275
6610-0050	9	77	277

Осуществим чтение данных по значению поля.

Используем функцию:

`mdbRead(Collection, Key, Value)`, где:

`Collection` – коллекция, в которой производится поиск;

`Key` – ключ, по которому ведется поиск;

`Value` – значение ключа.

```
key = input("Введите ключ для поиска: ")
```

```
value = input("Введите значение ключа для поиска: ")
```

```
res = mdbRead(collection, key, value)
```

```
print("Результат поиска:")
```

```
for conus in res:
```

```
print(conus["V_NKH"] + " " + conus["V_DPO "])
```

Осуществим запись данных в базу.

Используем функцию

`mdbWrite(Collection, Dict)`, где:

`Collection` – коллекция, в которой производится поиск;

`Dict` – словарь с данными.

```
mdbWrite(collection, {"V_OB": "6610-0021", "V_NKH": 3, " V_DPO": 38.0, " V_VI ": 199})
```

```
res = mdbRead(collection, " V_OB ", "6610-0021")
```

```
for conus in res:
```

```
    print(conus)
```

Результат работы функций:

```
{"V_OB": "6610-0021", '_id': ObjectId('256c47fcacb1aa0ec8f77b'),
" V_NKH ": 3, " V_DPO ": 38.0, " V_VI ": 199}
```


Обновление документа

Используем функцию

`mdbUpdate(Collection, Key, KeyValue, Field, FieldValue)`, где:

`Collection` – коллекция, в которой производится поиск;

`Key` – ключ для поиска записи;

`KeyValue` – значение ключа;

`Field` – записываемое поле;

`FieldValue` – записываемое значение.

```
key = input("Введите ключ для редактирования: ")
```

```
key_value = input("Введите значение ключа для редактирования: ")
```

```
print("Исходные документы:")
```

```
res = mdbRead(collection, key, key_value)
```

```
for conus in res:
```

```
    print(conus)
```

```
field = input("Введите поле для редактирования: ")
```

```
field_value = input("Введите новое значение поля: ")
```

```
res = mdbUpdate(collection, key, key_value, field, field_value)
```

```
print("Результат редактирования:")
```

```
res = mdbRead(collection, field, field_value)
```

```
for conus in res:
```

```
    print(conus)
```

Результат работы функций:

Введите ключ для редактирования: V_DPO

Введите значение ключа для редактирования: 65.0

Исходные документы:

```
{'_id': ObjectId('782c4d0hacb1ca18586f058d'), ({'V_OB ': '6610-00243', ' V_NKH ': 4, ' V_DPO ': 65.0, ' V_VI ': 257}
```

Введите поле для редактирования: V_DPO

Введите новое значение поля: 100.0

Результат редактирования:

```
{'_id': ObjectId('782c4d0hacb1ca18586f058d'), ({'V_OB ': '6610-00243', ' V_NKH ': 4, ' V_DPO ': 100.0, ' V_VI ': 257}
```

3.5. Пример создания базы данных “Зенковки”

Цель работы:

Разработать БД режущего инструмента в среде разработки MongoDB.

Пользуясь средой разработки MongoDB:

1. разработать структуру базы данных “Зенковки с углом при вершине 60 центров”
2. разработать запрос на поиск
 - а) по обозначению зенковки;
 - б) по диапазону диаметра зенковки.

Выполнение работы:

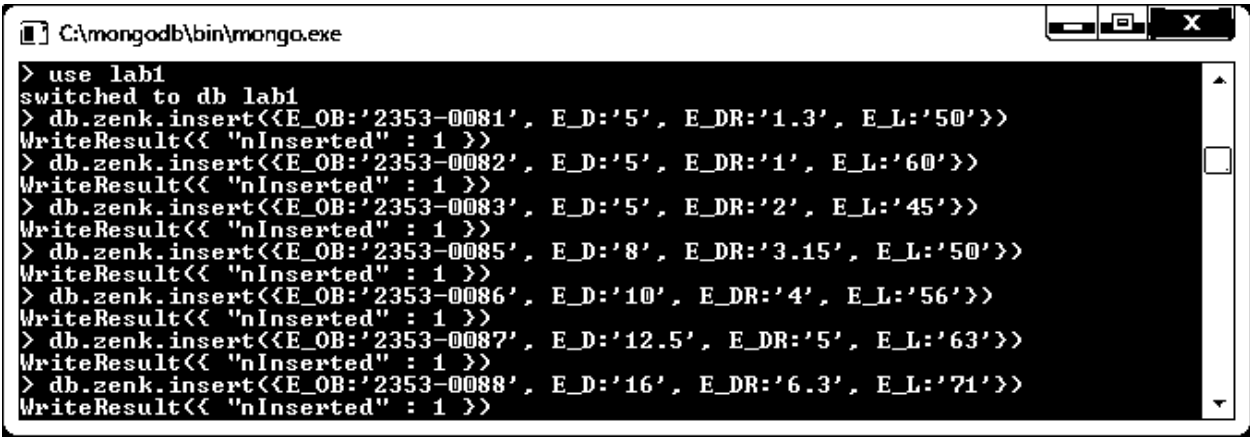
1. Выберем базу данных, с которой будет вестись работа, с помощью глобального метода `use`. При этом база данных с заданным именем будет автоматически сгенерирована в момент добавления в нее первой коллекции.

`use lab1`

2. Добавим в базу данных новую коллекцию. Так как коллекции бесструктурны, можно просто вставлять документ в новую коллекцию. Для этого используем команду `insert` («вставить»), передав ей вставляемый документ:

```
db.zenk.insert({E_OB:'2353-0081',E_D:'5', E_DR:'1.3', E_L:'50'})
```

Аналогичным образом добавим в коллекцию остальные документы. Таким образом, получим коллекцию, содержащую необходимый список документов.



```
C:\mongodb\bin\mongo.exe
> use lab1
switched to db lab1
> db.zenk.insert({E_OB:'2353-0081', E_D:'5', E_DR:'1.3', E_L:'50'})
WriteResult<< "nInserted" : 1 >>
> db.zenk.insert({E_OB:'2353-0082', E_D:'5', E_DR:'1', E_L:'60'})
WriteResult<< "nInserted" : 1 >>
> db.zenk.insert({E_OB:'2353-0083', E_D:'5', E_DR:'2', E_L:'45'})
WriteResult<< "nInserted" : 1 >>
> db.zenk.insert({E_OB:'2353-0085', E_D:'8', E_DR:'3.15', E_L:'50'})
WriteResult<< "nInserted" : 1 >>
> db.zenk.insert({E_OB:'2353-0086', E_D:'10', E_DR:'4', E_L:'56'})
WriteResult<< "nInserted" : 1 >>
> db.zenk.insert({E_OB:'2353-0087', E_D:'12.5', E_DR:'5', E_L:'63'})
WriteResult<< "nInserted" : 1 >>
> db.zenk.insert({E_OB:'2353-0088', E_D:'16', E_DR:'6.3', E_L:'71'})
WriteResult<< "nInserted" : 1 >>
```

Рисунок 9. Добавление документов в коллекцию

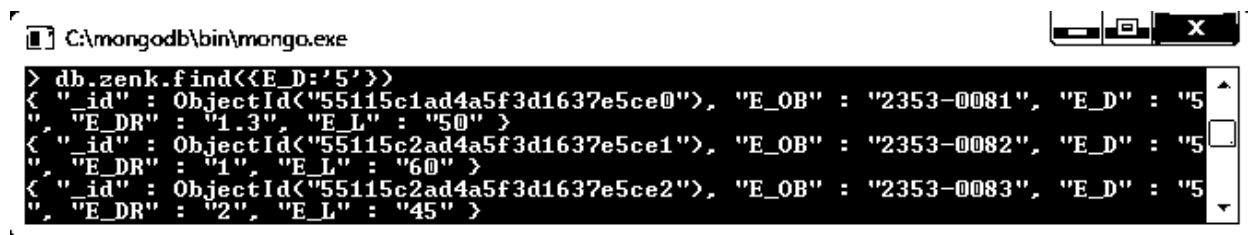
3. Воспользуемся методом `find()`, который вернет список зенковок, имеющих заданное обозначение (`E_OB`). Например, вернем зенковки, имеющие обозначение “2353-0085”. (Рис. 10).



```
C:\mongodb\bin\mongo.exe
> db.zenk.find({E_OB:'2353-0085'})
{ "_id" : ObjectId("55115c2ad4a5f3d1637e5ce3"), "E_OB" : "2353-0085", "E_D" : "8",
  "E_DR" : "3.15", "E_L" : "50" }
>
```

Рисунок 10. Поиск зенковок по обозначению

4. Для того чтобы найти зенковки по диаметру, снова используем метод `find()`, указав требуемую величину диаметра. (Рис. 11).



```
C:\mongodb\bin\mongo.exe
> db.zenk.find(<E_D:'5'>)
{ "_id" : ObjectId("55115c1ad4a5f3d1637e5ce0"), "E_OB" : "2353-0081", "E_D" : "5",
  "E_DR" : "1.3", "E_L" : "50" }
{ "_id" : ObjectId("55115c2ad4a5f3d1637e5ce1"), "E_OB" : "2353-0082", "E_D" : "5",
  "E_DR" : "1", "E_L" : "60" }
{ "_id" : ObjectId("55115c2ad4a5f3d1637e5ce2"), "E_OB" : "2353-0083", "E_D" : "5",
  "E_DR" : "2", "E_L" : "45" }
```

Рисунок 11. Поиск зенковок по диаметру

Вывод: в ходе выполнения показан пример разработки структуры базы данных “Зенковки с углом при вершине 60 центров” в среде MongoDB, а также запросы для поиска по различным параметрам.

Литература

1. Karl Seguin The Little MongoDB Book
<http://jsman.ru/mongo-book/index.html>
2. Официальный сайт СУБД MongoDB
<http://www.mongodb.org/downloads>
3. Язык запросов MongoDB
<http://softtime.info/view/>
4. Реляционные базы данных
<https://ru.wikipedia.org/wiki>
5. HYPERTABLE INC
<http://www.hypertable.com/>
6. Apache Hadoop
<http://hadoop.apache.org/>
7. Apache Cassandra
<http://cassandra.apache.org/>
8. FranzInc <http://franz.com/>
9. Sones graphDB
<http://www.sones.de/static-en/>
10. Introducing Neo4j 3.1 The Graph Foundation for the Enterpris.
<https://neo4j.com/>
11. Dynamite Nosql
<http://www.slideshare.net/adorepump/dynamite-nosql>
12. Project Voldemort
<http://www.project-voldemort.com/voldemort/>
13. Tokyo Cabinet and Kyoto Cabinet
https://en.wikipedia.org/wiki/Tokyo_Cabinet_and_Kyoto_Cabinet
14. Redis <https://redis.io/>
15. CouchDB
<http://couchdb.apache.org/>
16. Berkeley DB
<http://www.oracle.com/us/products/database/berkeley-db/index.html>
17. Exist DB
<http://www.exist-db.org/exist/apps/homepage/index.html>
18. MongoDB
<https://www.mongodb.com/>
19. Кингсли, Х.Э. JavaScript в примерах. [Электронный ресурс] / Х.Э. Кингсли, Х.К. Кингсли. — Электрон. дан. — М. : ДМК Пресс, 2009. — 272 с. — Режим доступа:
<http://e.lanbook.com/book/1271> — Загл. с экрана.
20. Зудилова, Т.В. Web-программирование JavaScript. [Электронный

- ресурс] / Т.В. Зудилова, М.Л. Буркова. — Электрон. дан. — СПб. : НИУ ИТМО, 2012. — 68 с. — Режим доступа: <http://e.lanbook.com/book/43561> — Загл. с экрана.
21. Россум, Г. Язык программирования Python / Г. Россум, Ф.Л.Дж. Дрейк, Д.С. Откидач и др.. - М.: 2001. - 107 с.
 22. Официальный сайт разработчиков языка программирования Python
<https://www.python.org/>
 23. Сайт для загрузки GUI MongoDB Compass
<https://www.mongodb.com/download-center?jmp=nav#compass>
 24. Официальная документация к GUI MongoDB Compass
<https://docs.mongodb.com/manual/>
 25. Сайт для загрузки GUI Studio3T
<https://studio3t.com/download/>

Приложение 1

Установка MongoDB

Для установки MongoDB загрузим один распространяемых пакетов с официального сайта [2].

Официальный сайт предоставляет пакеты для различных платформ: Windows, Linux, MacOS, Solaris. И различные версии: как 32-х битные, так и 64-х битные.

При использовании MongoDB следует учитывать, что MongoDB использует отображаемые в памяти файлы, поэтому на 32-х разрядных системах ее действие ограничено 2 Гб памяти. Чтобы поддерживать базы данных большего объема на 32-х разрядных системах, требуются дополнительные усилия. При развертывании на 64-х разрядных системах подобных ограничений нет.

Также обратите внимание на версию дистрибутива. На момент написания данного материала новейшим дистрибутивом был **Production Release (2.4.9)**.

В принципе как таковой установки не потребуется. После загрузки пакета MongoDB его следует распаковать. Например, на ОС Windows в корневой каталог диска C. Переименуем распакованный каталог с файлами в *mongodb*. Однако, при необходимости можно поместить пакет MongoDB в любое другое место на локальной машине. После распаковки мы можем зайти в папку пакета и далее в каталог *bin*, который, собственно, и содержит все бинарные файлы MongoDB.

Содержимое пакета MongoDB

Открыв папку *bin*, можно найти там кучу приложений, которые выполняют определенную роль. Вкратце рассмотрим их.

- **bsondump**: считывает содержимое BSON-файлов и преобразует их в читабельный формат, например, в JSON;
- **mongo**: представляет консольный интерфейс для взаимодействия с базами данных;
- **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, импортирующая данных в форматах JSON, TSV или CSV в базу данных MongoDB;
- **mongooplog**: приложение, позволяющее опрашивать удаленные серверы на наличие операций с БД и применять полученные данные к локальному серверу;
- **mongoperf**: проверяет производительность жесткого диска на предмет операций ввода-вывода;
- **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;
- **mongosniff**: позволяет проводить низкоуровневый мониторинг базам MongoDB в реальном режиме времени;
- **mongorestat**: представляет счетчики операций с БД;
- **mongotop**: предоставляет способ подсчета времени, затраченного на операции чтения-записи в БД.

Создание каталога для БД и запуск MongoDB

После установки или лучше сказать распаковки надо создать на жестком диске каталог, где будут находиться базы данных MongoDB.

В ОС Windows по умолчанию MongoDB хранит базы данных по пути `C:\data\db`, поэтому, если вы используете Windows, вам надо создать соответствующий каталог. 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.

Если у вас используется расположение баз данных, отличающееся от настроек по умолчанию, то при запуске можно задать каталог для баз данных следующим образом:

```
C:\mongodb\bin\mongod.exe --dbpath d:\test\mongodb\data.
```

В данном случае предполагается, что базы данных у нас будут находиться в каталоге d:\test\mongodb\data.

И после удачного запуска сервера мы сможем производить операции с БД через оболочку **mongo**. Запустим же mongo.exe:

Второй строкой приложение говорит о подключении к базе данных **test**.

Хотя до запуска mongo.exe такой базы данных не существовало, MongoDB автоматически создаст ее при подключении. И мы сможем увидеть в папке C:\data\db (если мы не изменили каталог для баз данных по умолчанию) ряд автоматически сгенерированных файлов.

Теперь что-нибудь добавим в БД test. Введем следующие команды:

```
db.perons.save( { id: "1", name: "Eugene" } )
db.persons.find()
```

Здесь db представляет текущую базу данных - то есть, базу данных test, далее идет persons- это коллекция, в которую затем мы добавляем новый объект. Если в SQL нам надо создавать таблицы заранее, то коллекции MongoDB создает самостоятельно при их отсутствии. Далее идет описание добавляемого объекта в формате, с которым вы возможно знакомы, если имели дело с форматом JSON.

А вторая команда выводит имеющую строку из БД на экран.

Из вывода вы можете увидеть, что к начальным значениям объекта было добавлено какое-то непонятно поле **ObjectId**. Как вы помните, MongoDB

в качестве уникальных идентификаторов документа использует поле **_id**. И в данном случае ObjectId как раз и представляет значение для идентификатора **_id**.

Установка драйверов MongoDB

Конечно, мы можем работать и через консоль `mongo`, добавляя и отображая объекты в БД. Но нам также было бы неплохо, если бы `mongoDB` взаимодействовала бы с нашими приложениями, написанными на PHP, C++, C# и других языках программирования. И для этой цели нам потребуются специальные драйверы.

На официальном сайте <http://docs.mongodb.org/ecosystem/drivers/> можно найти драйвера для таких языков программирования, как PHP, C++, C#, Java, Python, Perl, Ruby, Scala и др.

Далее уже, рассматривая взаимодействие отдельных языков программирования с MongoDB, мы подробнее рассмотрим установку и драйвера и всю необходимую конфигурацию для определенных языков программирования.

Приложение 2

Функции работы с СУБД MongoDB на ЯП Python

```
# -*- coding: cp1251 -*-
# Импорт модуля pymongo.
import pymongo
def mdbCreate(NameDB, Connection):
    '''
    Создание БД MongoDB.
    Параметры:
        NameDB - имя создаваемой БД;
        Connection - объект соединения (англ. - connection
object) - результат работы функции pymongo.Connection.
    '''
    return Connection[NameDB]
def mdbDelete(NameDB, Connection):
    '''
    Удаление БД MongoDB.
    Параметры:
        NameDB - имя удаляемой БД;
        Connection - объект соединения (англ. - connection
object) - результат работы функции pymongo.Connection.
    '''
    Connection.drop_database(NameDB)

def mdbRead(Collection, Key, Value):
    '''
    Чтение документа по ключу.
    Параметры:
        Collection - коллекция, в которой производится поиск;
        Key - ключ;
        Value - значение ключа.
    '''
    return Collection.find({Key: Value})
def mdbWrite(Collection, Dict):
    '''
    Запись документа.
    Параметры:
        Collection - коллекция, в которой производится поиск;
        Dict - словарь с данными.
    '''
    Collection.save(Dict)
def mdbUpdate(Collection, Key, KeyValue, Field, FieldValue):
    '''
    Обновление значения поля документа по ключу.
    Параметры:
```

```
Collection - коллекция, в которой производится поиск;
Key - ключ;
KeyValue - значение ключа;
Field - записываемое поле;
FieldValue - записываемое значение.
'''
    Collection.update({Key: KeyValue}, {"$set": {Field:
FieldValue}})
# Соединение с сервером базы данных.
connection = pymongo.Connection()
```