



Базы данных

какие бывают, различия, что выбрать



База данных (БД)

База данных — совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных.

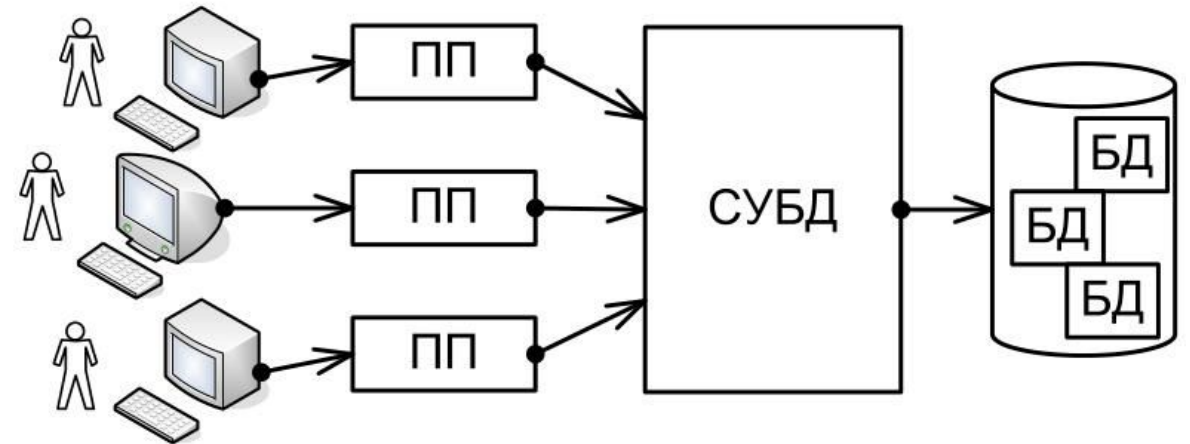
База данных — это организованная структура, предназначенная для хранения информации



Система управления базами данных (СУБД)

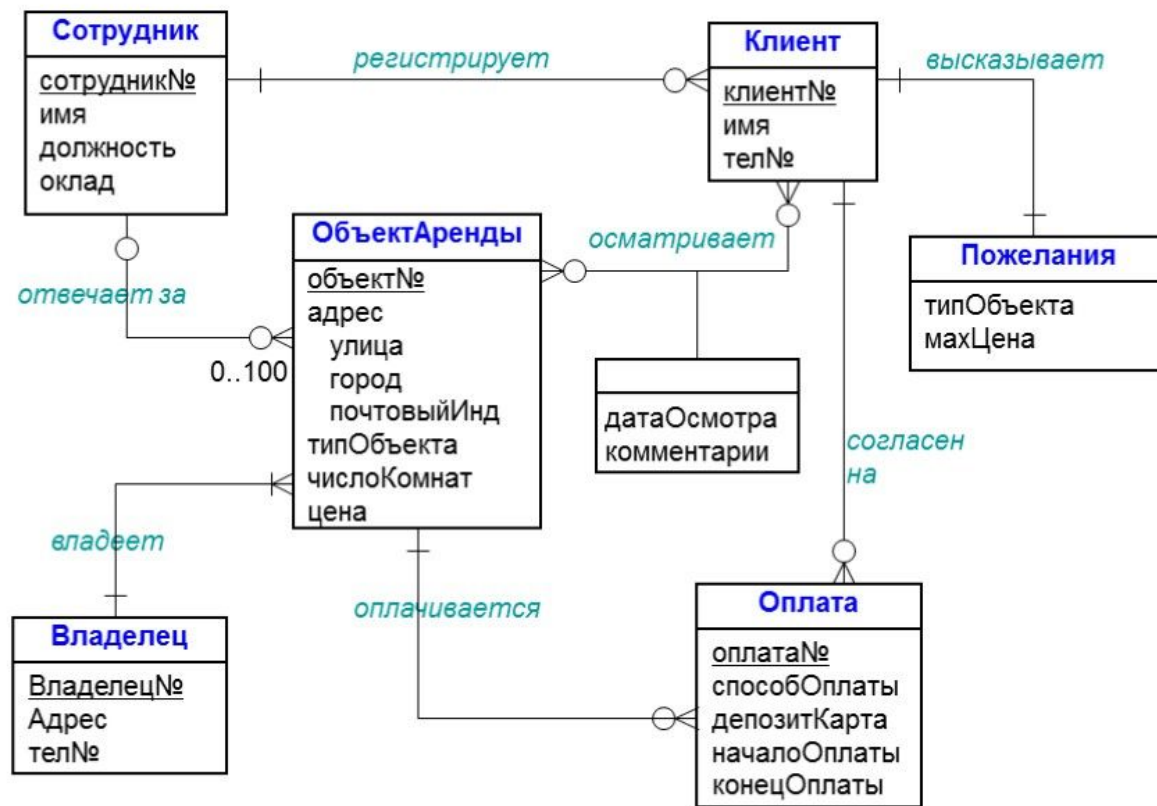
СУБД — комплекс программ, позволяющих создать базу данных и манипулировать данными (вставлять, обновлять, удалять и выбирать).

Система обеспечивает безопасность, надёжность хранения и целостность данных, а также предоставляет средства для администрирования БД.



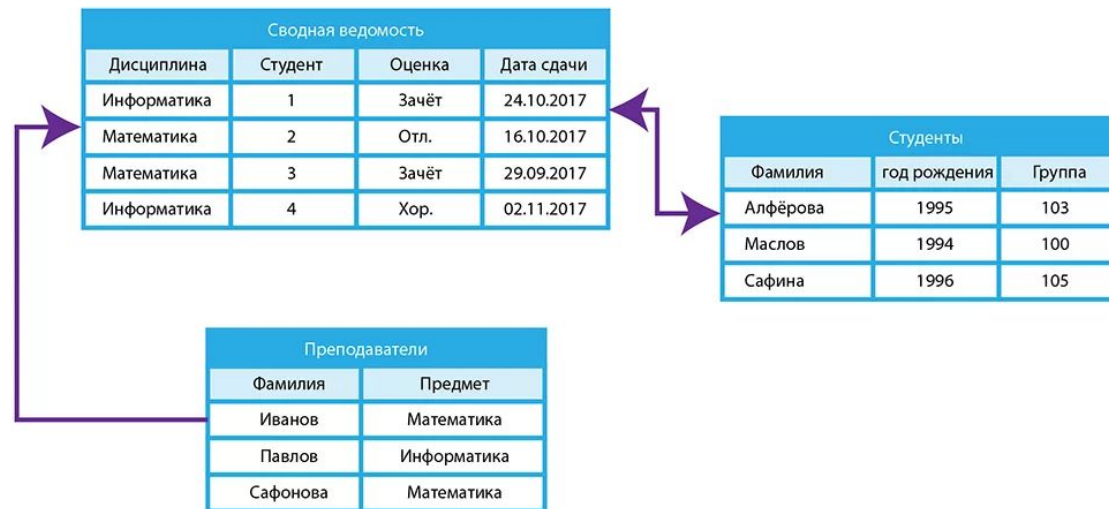
Реляционные СУБД (SQL базы)

Реляционная база данных (SQL) — база, где данные хранятся в формате таблиц, они строго структурированы и связаны друг с другом. В таблице есть строки и столбцы, каждая строка представляет отдельную запись, а столбец — поле с назначенным ей типом данных. В каждой ячейке информация записана по шаблону.



Реляционная модель данных

Представленная в 70-х, реляционная модель предлагает математический способ структуризации, хранения и использования данных. Отношения (англ. relations) дают возможность группировки данных как связанных наборов, представленных в виде таблиц, содержащих упорядоченную информацию (например, имя и адрес человека) и соотносящих значения и атрибуты (его номер паспорта).



Реляционные СУБД (SQL базы)

Основные черты:

- Жесткая структура данных в таблицах
- Нормализация данных (разделение по нескольким таблицам)
- Поддержка ACID-транзакций
 - Atomicity, или атомарность — ни одна транзакция не будет зафиксирована в системе частично
 - Consistency, или непротиворечивость — фиксируются только допустимые результаты транзакций
 - Isolation, или изолированность — на результат транзакции не влияют транзакции, проходящие параллельно ей
 - Durability, или долговечность — изменения в базе данных сохраняются несмотря на сбои или действия пользователей
- Стандартный язык SQL для выборки и манипуляции с данными
- Вертикальная масштабируемость (увеличение производительности сервера)



Самые популярные РСУБД

- **SQLite**: очень мощная встраиваемая РСУБД.
- **MySQL**: самая популярная и часто используемая РСУБД.
- **PostgreSQL**: самая продвинутая и гибкая РСУБД.
- **Oracle Database**: очень популярна в коммерческих продуктах
- **Microsoft SQL Server**: популярна в коммерческих продуктах на ОС Windows



Нереляционные базы данных (NoSQL базы)

NoSQL СУБД не используют реляционную модель структуризации данных. Существует много реализаций, решающих этот вопрос по-своему, зачастую весьма специфично. Эти решения допускают неограниченное формирование записей и хранение данных в виде ключ-значение.



Нереляционные базы данных (NoSQL базы)

Основные черты:

- Отказ от реляционной модели и языка SQL
- Schema-less данные – в NoSQL БД структура данных заранее не регламентируется
- Использование распределенной архитектуры
- Отсутствие полноценной поддержки ACID-транзакций

Преимущества:

- Большая производительность
- Хорошая масштабируемость при возрастающих нагрузках и огромных объемах данных



Популярные NoSQL СУБД

- MongoDB: документоориентированная база данных
- Redis, Memcached: база данных ключ-значение, быстрый кэш
- Neo4j, OrientDB: графовые базы данных
- Cassandra: колоночная база данных



DynamoDB



Сравнение SQL и NoSQL баз данных

SQL

- Требуют наличия однозначно определённой структуры хранения данных
- Реализуют SQL-стандарты, поэтому из них можно получать данные при помощи языка SQL
- Легкая вертикальная и горизонтальная масштабируемость
- Надежнее (ACID)
- Проще поддержка, сложнее настройка
- Необходимо поддерживать две разнородные модели: реляционную в базе и объектную в коде
- Любые изменения в схеме сущностей нужно отражать в структуре таблиц + менять SQL-запросы и проекции таблиц на объекты

NoSQL

- Не используют общий формат запроса (как SQL в реляционных базах данных). Каждое решение использует собственную систему запросов.
- Легкая вертикальная и горизонтальная масштабируемость
- Сложная поддержка, простая настройка
- Плохо подходят для операций, отличных от CRUD



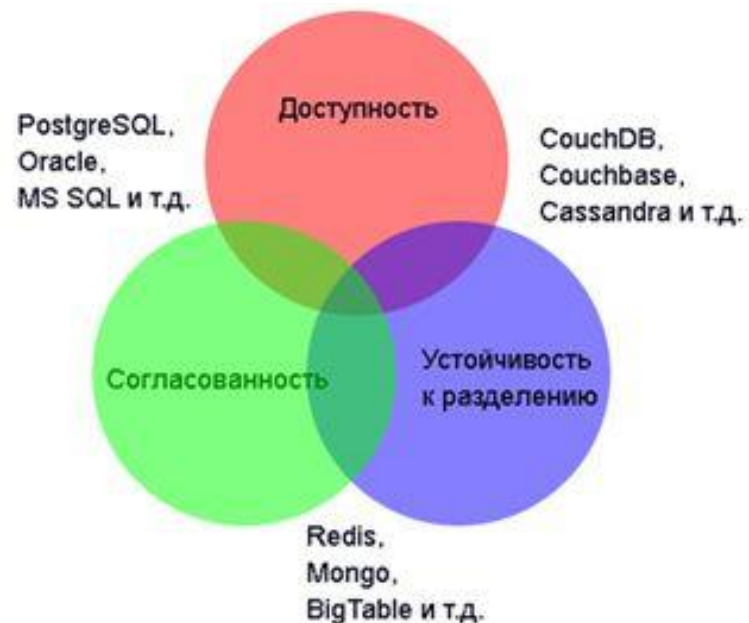
Теорема CAP

Consistency - согласованность данных

Availability - доступность

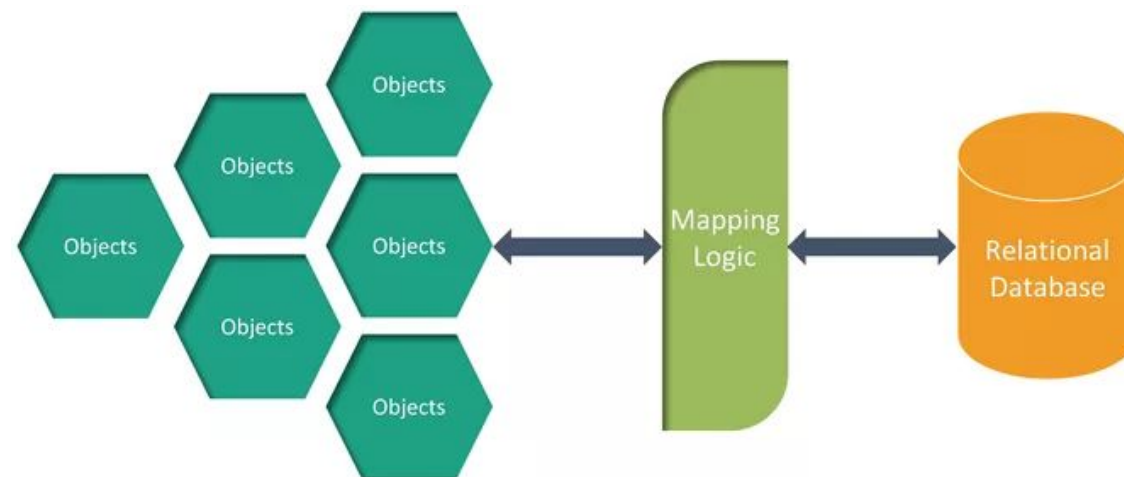
Partition tolerance - устойчивость к разделению

Долгое время консистентность была «священной коровой» для архитекторов и разработчиков, но с приходом огромных массивов информации и распределенных систем, эти требования утратили актуальность



ORM

ORM (Object-Relational Mapping) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных»



Плюсы и минусы ORM

- ✓ Для простых запросов (CRUD) ORM подходит хорошо
- ✓ Более строгая типизация результатов
- ✓ Возможность оперировать элементами языка программирования, т.е. классами, объектами, атрибутами, методами, а не элементами реляционной модели данных
- ✓ Простая навигация по связям
- ✓ Избавляет от необходимости работы с SQL и проработки значительного количества программного кода
- ✗ Дополнительная нагрузка на программиста
- ✗ Реализация ORM ведет к увеличению объема программного кода и снижению скорости работы программы
- ✗ Появление трудно поддающихся отладке ошибок в программе, если присутствуют ошибки в реализации ORM





Ваши вопросы

что необходимо прояснить в рамках
данного раздела



MongoDB

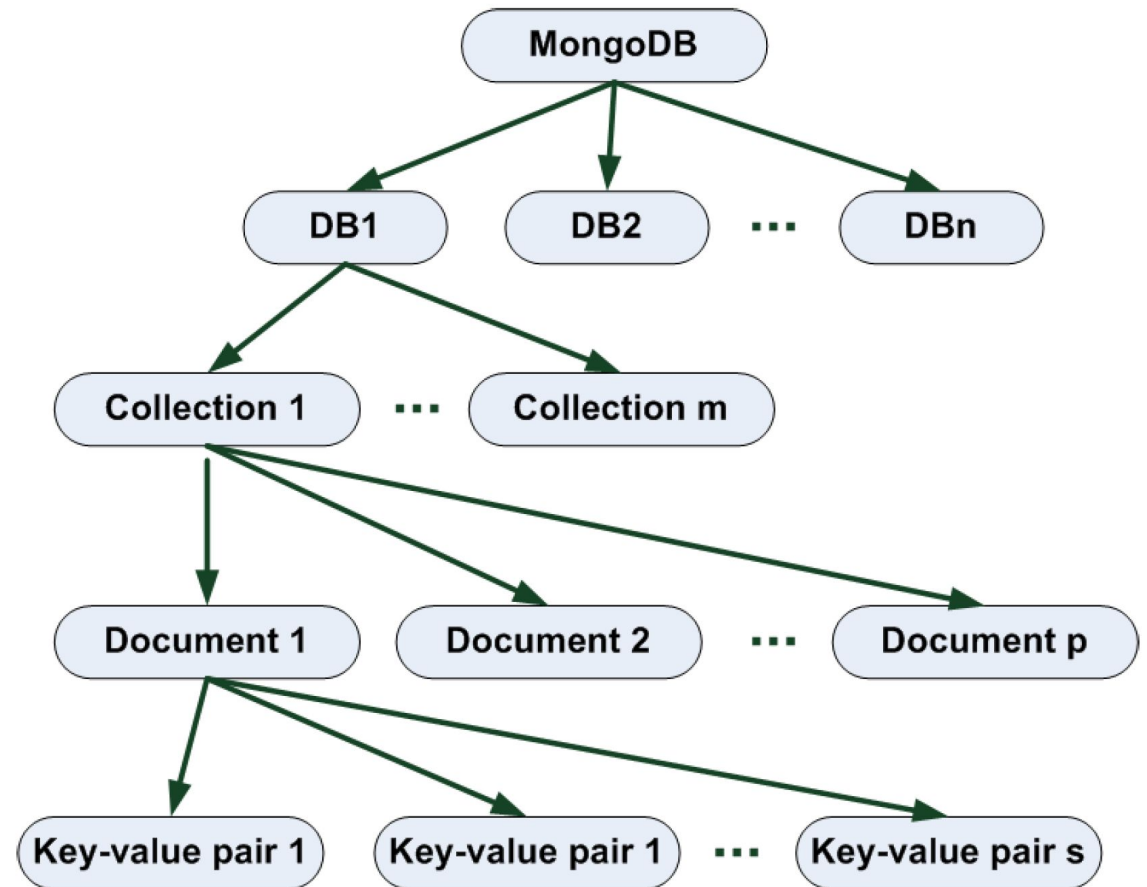
основы работы с популярной
нереляционной базой данных



MongoDB

MongoDB является документо-ориентированной системой, в которой центральным понятием является документ.

Каждая коллекция имеет свое уникальное имя - произвольный идентификатор, состоящий из не более чем 128 различных алфавитно-цифровых символов и знака подчеркивания.



Как выглядит документ в Mongo

Документы могут содержать разнородную информацию.

В одной коллекции могут быть документы с разными полями.

```
1  {  
2      "name": "Bill",  
3      "surname": "Gates",  
4      "age": "48",  
5      "company": {  
6          "name" : "microsoft",  
7          "year" : "1974",  
8          "price" : "300000"  
9      }  
10 }
```

Регистр

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

```
1  {"age" : "28"}
2  {"age" : 28}
```

Что под капотом

- Для каждого документа в MongoDB определен уникальный идентификатор, который называется «_id». При добавлении документа в коллекцию данный идентификатор создается автоматически. Однако разработчик может сам явным образом задать идентификатор, а не полагаться на автоматически генерируемые, указав соответствующий ключ и его значение в документе
- Поле «_id» должно иметь уникальное значение в рамках коллекции. И если мы попробуем добавить в коллекцию два документа с одинаковым идентификатором, то добавится один из них, а при добавлении второго получим ошибку
- Все данные хранятся в формате BSON, который близок к JSON, поэтому нам надо также вводить данные в этом формате



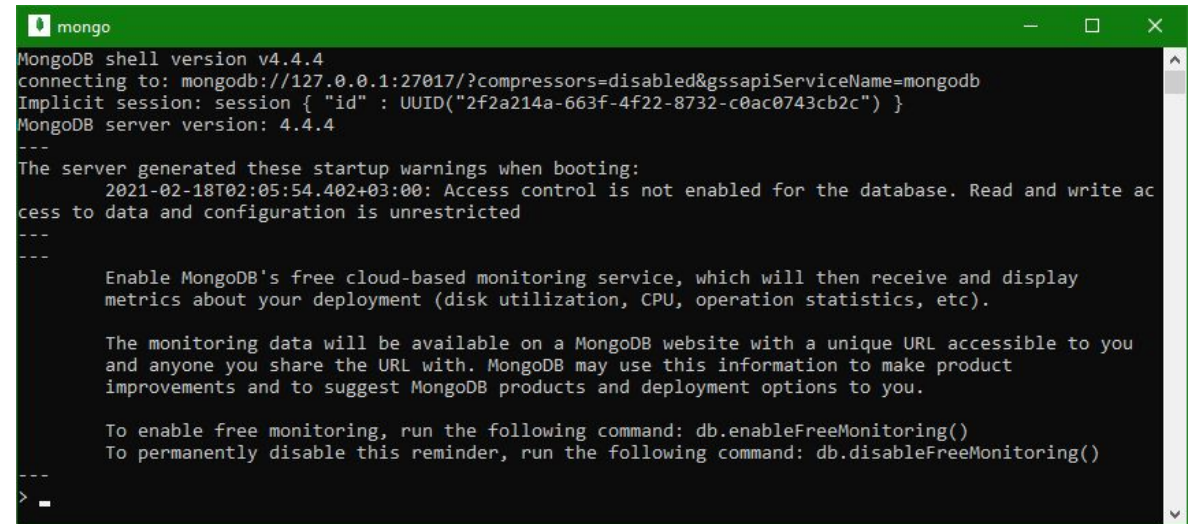
Типы данных

Тип	Название	Что хранит
String	Строка	строковый тип, используется кодировка UTF-8
Array	Массив	массив элементов
Binary data	Двоичные данные	данные в бинарном формате
Boolean	Булевый тип	логические значения TRUE или FALSE
Date	Дата	дату в формате времени Unix
Double	Вещественные числа	числа с плавающей точкой
Integer	Целые числа	целочисленные значения
JavaScript	JS код	кода javascript
Min key/Max key		используются для сравнения значений с наименьшим/наибольшим элементов BSON
Null	Пустой тип	для хранения значения Null
Object	Строковый тип	Данные некоторого объекта
ObjectId	ID документа	тип данных для хранения id документа
Regular expression	Регулярные выражения	для хранения регулярных выражений
Symbol	Тип данных, идентичный строковому	Используется преимущественно для тех языков, в которых есть специальные символы
Timestamp	Время	для хранения времени



Запускаем консоль MongoDB

Для работы с MongoDB из консоли, нужно запустить exe файл в папке, куда была установлена MongoDB



```
mongo
MongoDB shell version v4.4.4
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("2f2a214a-663f-4f22-8732-c0ac0743cb2c") }
MongoDB server version: 4.4.4
---
The server generated these startup warnings when booting:
  2021-02-18T02:05:54.402+03:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
---
---
  Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).

  The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.

  To enable free monitoring, run the following command: db.enableFreeMonitoring()
  To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> _
```

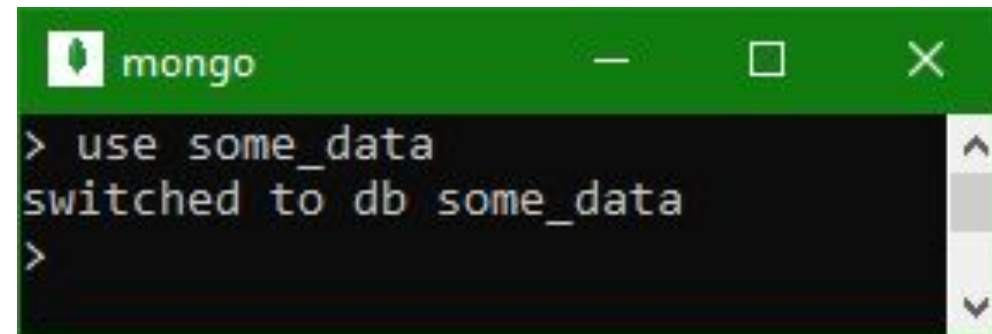


Выбираем нужную БД

Первым делом надо установить нужную нам базу данных в качестве текущей, чтобы затем ее использовать.

Для этого надо использовать команду «**use**», после которой идет название базы данных.

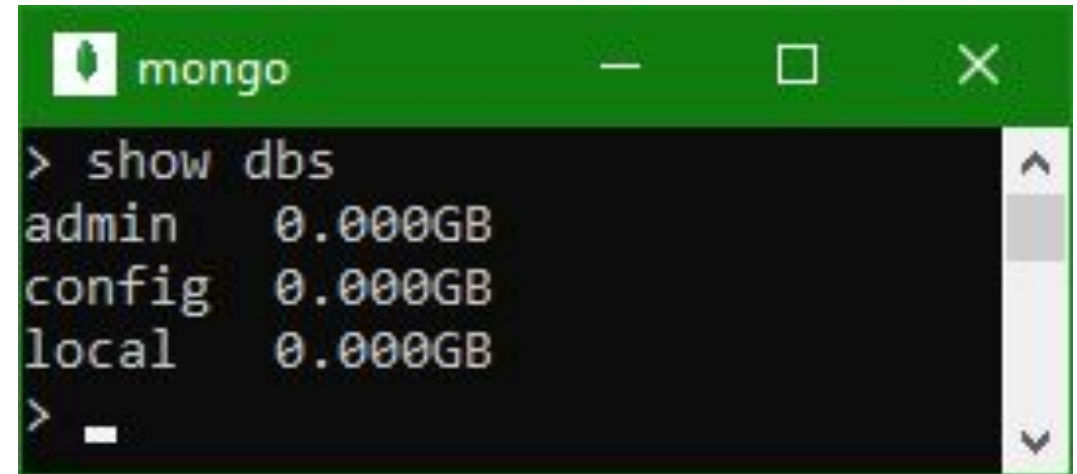
Не важно, существует ли такая БД или нет. Если ее нет, то MongoDB автоматически создаст ее при добавлении в нее данных.



```
mongo
> use some_data
switched to db some_data
>
```

Выводим список всех баз данных

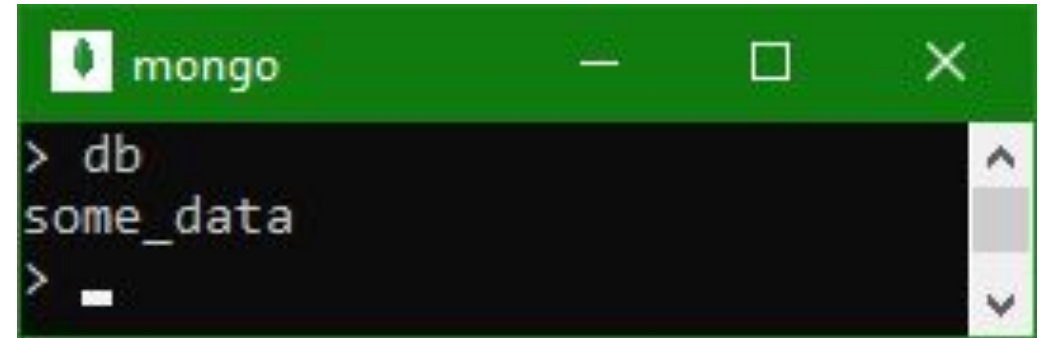
Если вы вдруг не уверены, а существует ли уже база данных с таким названием, то с помощью команды «**show dbs**» можно вывести названия всех имеющихся БД на консоль



```
mongo
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
> _
```

Узнаем текущую активную базу данных

Если мы хотим узнать, какая база данных используется в текущий момент, то мы можем воспользоваться командой «**db**»



```
> db
some_data
> _
```

Правила именований баз данных

Базы данных:

- Для базы данных можно задать любое имя, однако есть некоторые ограничения: в имени не должно быть символов /, \, ., ", *, <, >, :, |, ?, \$. Кроме того, имена баз данных ограничены 64 байтами
- Также есть зарезервированные имена, которые нельзя использовать: local, admin, config

Ключи:

- Символ «\$» не может быть первым символом в имени ключа
- Имя ключа не может содержать символ точки «.»



Статистика баз данных

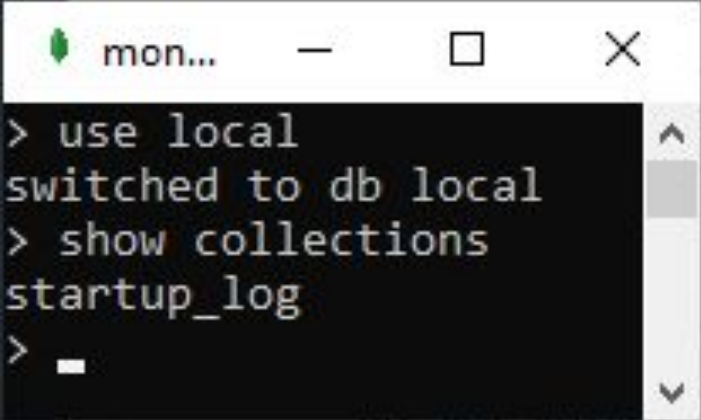
- Используя команду **db.stats()**, можно получить статистику по текущей базе данных
- Для получения информации о другой базе данных **db.имя.stats()**

```
mongo
> db.stats()
{
  "db" : "some_data",
  "collections" : 0,
  "views" : 0,
  "objects" : 0,
  "avgObjSize" : 0,
  "dataSize" : 0,
  "storageSize" : 0,
  "totalSize" : 0,
  "indexes" : 0,
  "indexSize" : 0,
  "scaleFactor" : 1,
  "fileSize" : 0,
  "fsUsedSize" : 0,
  "fsTotalSize" : 0,
  "ok" : 1
}
> db.admin.stats()
{
  "ns" : "some_data.admin",
  "size" : 0,
  "count" : 0,
  "storageSize" : 0,
  "totalSize" : 0,
  "nindexes" : 0,
  "totalIndexSize" : 0,
  "indexSizes" : {
  },
  "scaleFactor" : 1,
  "ok" : 1
}
```



Просмотр списка коллекций в БД

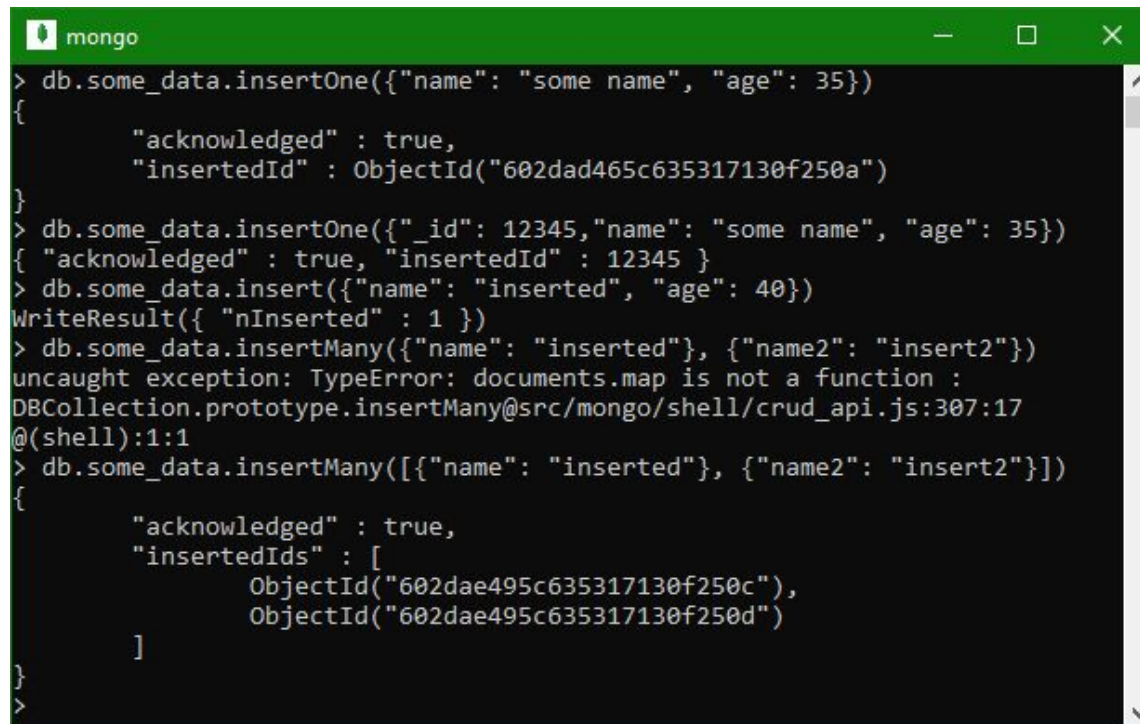
Мы можем просмотреть список всех коллекций в текущей БД с помощью команды **show collections**



```
mon...  
> use local  
switched to db local  
> show collections  
startup_log  
> _
```

Вставка документов в базу данных

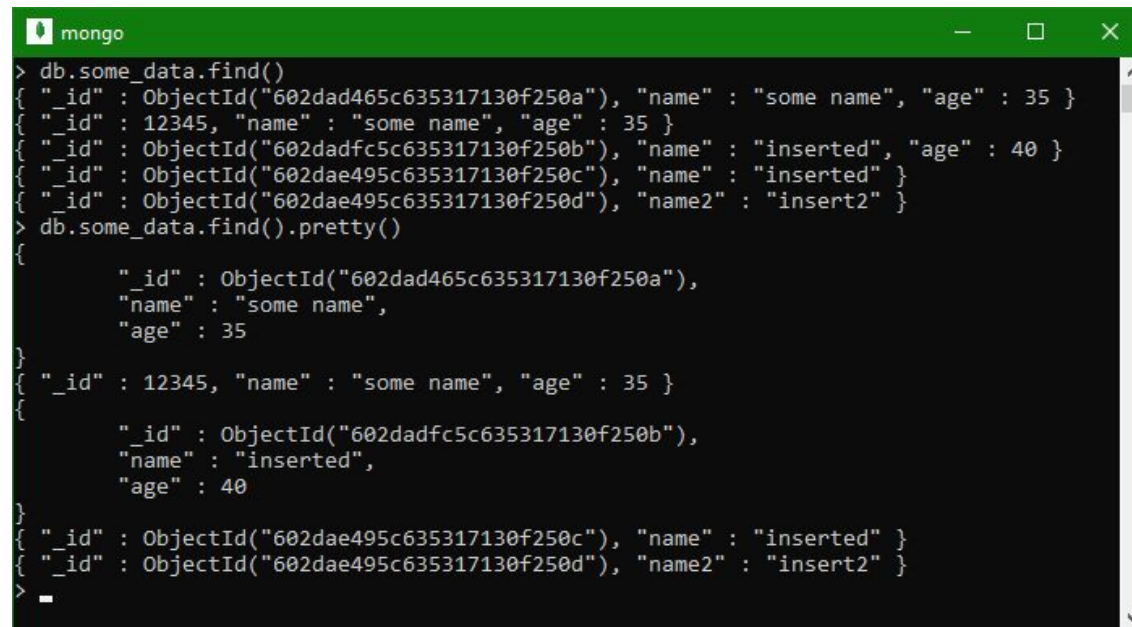
- **insertOne():** добавляет один документ
- **insertMany():** добавляет несколько документов
- **insert():** может добавлять как один, так и несколько документов



```
mongo
> db.some_data.insertOne({"name": "some name", "age": 35})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("602dad465c635317130f250a")
}
> db.some_data.insertOne({"_id": 12345, "name": "some name", "age": 35})
{ "acknowledged" : true, "insertedId" : 12345 }
> db.some_data.insert({"name": "inserted", "age": 40})
WriteResult({ "nInserted" : 1 })
> db.some_data.insertMany({"name": "inserted"}, {"name2": "insert2"})
uncaught exception: TypeError: documents.map is not a function :
DBCollection.prototype.insertMany@src/mongo/shell/crud_api.js:307:17
@(shell):1:1
> db.some_data.insertMany([{"name": "inserted"}, {"name2": "insert2"}])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("602dae495c635317130f250c"),
    ObjectId("602dae495c635317130f250d")
  ]
}
```


Вывод элементов в консоль

- Для вывода элементов на экран **db.имя.find()**
- Для красивого вывода **db.имя.find().pretty()**



```
mongo
> db.some_data.find()
{ "_id" : ObjectId("602dad465c635317130f250a"), "name" : "some name", "age" : 35 }
{ "_id" : 12345, "name" : "some name", "age" : 35 }
{ "_id" : ObjectId("602dadfc5c635317130f250b"), "name" : "inserted", "age" : 40 }
{ "_id" : ObjectId("602dae495c635317130f250c"), "name" : "inserted" }
{ "_id" : ObjectId("602dae495c635317130f250d"), "name2" : "insert2" }
> db.some_data.find().pretty()
{
  "_id" : ObjectId("602dad465c635317130f250a"),
  "name" : "some name",
  "age" : 35
}
{ "_id" : 12345, "name" : "some name", "age" : 35 }
{
  "_id" : ObjectId("602dadfc5c635317130f250b"),
  "name" : "inserted",
  "age" : 40
}
{ "_id" : ObjectId("602dae495c635317130f250c"), "name" : "inserted" }
{ "_id" : ObjectId("602dae495c635317130f250d"), "name2" : "insert2" }
```

Загрузка данных из файла

Данные для базы данных в MongoDB можно определять в обычном текстовом файле, что довольно удобно, поскольку мы можем переносить или пересылать этот файл независимо от базы данных MongoDB. Например, определим где-нибудь на жестком диске файл `users.js`

Для загрузки файла в текущую базу данных применяется функция `load()`, в которую в качестве параметра передается путь к файлу

```
1 db.users.insertMany([
2   {"name": "Alice", "age": 31, languages: ["english", "french"]},
3   {"name": "Lene", "age": 29, languages: ["english", "spanish"]},
4   {"name": "Kate", "age": 30, languages: ["german", "russian"]}
5 ])
```




```
mongo
> load("D:/users.js")
true
> db.users.find().pretty()
{
  "_id" : ObjectId("602db0585c635317130f250e"),
  "name" : "Alice",
  "age" : 31,
  "languages" : [
    "english",
    "french"
  ]
}

{
  "_id" : ObjectId("602db0585c635317130f250f"),
  "name" : "Lene",
  "age" : 29,
  "languages" : [
    "english",
    "spanish"
  ]
}

{
  "_id" : ObjectId("602db0585c635317130f2510"),
  "name" : "Kate",
  "age" : 30,
  "languages" : [
    "german",
    "russian"
  ]
}
```

Выводим документы по ключу

Для поиска документов, которые содержат определенные ключи – нужно передать ключи-значения в **db.имя.find()**

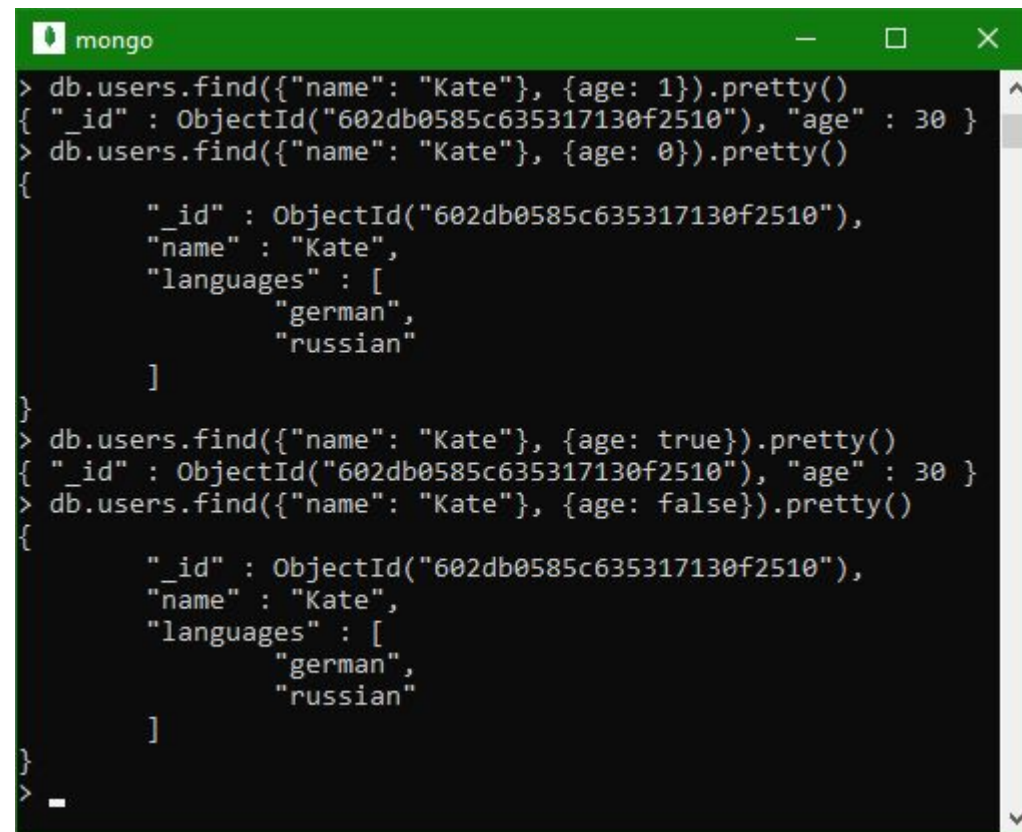


```
mongo
> db.users.find({name: "Kate"}).pretty()
{
  "_id" : ObjectId("602db0585c635317130f2510"),
  "name" : "Kate",
  "age" : 30,
  "languages" : [
    "german",
    "russian"
  ]
}
> db.users.find({name: "Kate", age: 30}).pretty()
{
  "_id" : ObjectId("602db0585c635317130f2510"),
  "name" : "Kate",
  "age" : 30,
  "languages" : [
    "german",
    "russian"
  ]
}
> db.users.find({name: "Kate", age: 32}).pretty()
{}
> db.users.find({"languages.0": "german"}).pretty()
{
  "_id" : ObjectId("602db0585c635317130f2510"),
  "name" : "Kate",
  "age" : 30,
  "languages" : [
    "german",
    "russian"
  ]
}
```



Проекция

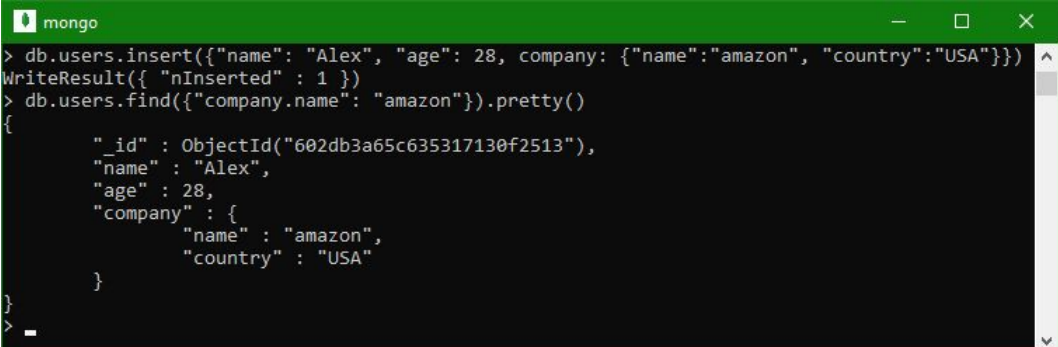
Документ может иметь множество полей, но не все эти поля нам могут быть нужны и важны при запросе. И в этом случае мы можем включить в выборку только нужные поля, используя проекцию



```
mongo
> db.users.find({"name": "Kate"}, {age: 1}).pretty()
{ "_id" : ObjectId("602db0585c635317130f2510"), "age" : 30 }
> db.users.find({"name": "Kate"}, {age: 0}).pretty()
{
  "_id" : ObjectId("602db0585c635317130f2510"),
  "name" : "Kate",
  "languages" : [
    "german",
    "russian"
  ]
}
> db.users.find({"name": "Kate"}, {age: true}).pretty()
{ "_id" : ObjectId("602db0585c635317130f2510"), "age" : 30 }
> db.users.find({"name": "Kate"}, {age: false}).pretty()
{
  "_id" : ObjectId("602db0585c635317130f2510"),
  "name" : "Kate",
  "languages" : [
    "german",
    "russian"
  ]
}
```

Запрос к вложенным объектам

Если документ содержит вложенные объекты, то к их ключам можно обратиться с помощью «.»

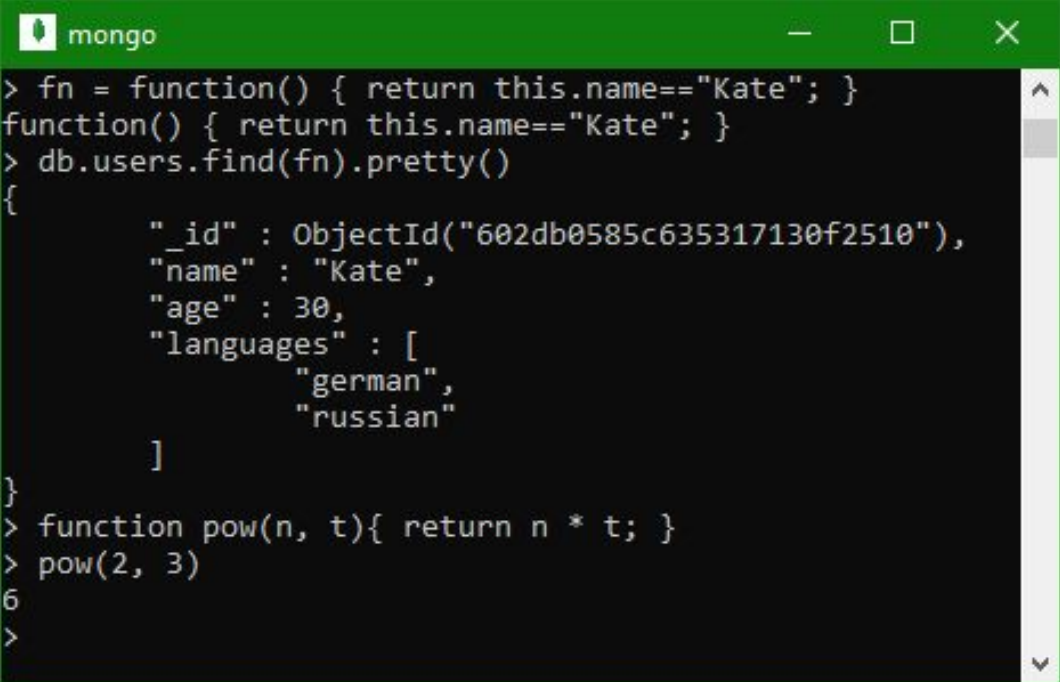


```
mongo
> db.users.insert({"name": "Alex", "age": 28, "company": {"name": "amazon", "country": "USA"}})
WriteResult({ "nInserted" : 1 })
> db.users.find({"company.name": "amazon"}).pretty()
{
  "_id" : ObjectId("602db3a65c635317130f2513"),
  "name" : "Alex",
  "age" : 28,
  "company" : {
    "name" : "amazon",
    "country" : "USA"
  }
}
```



JS в запросах

MongoDB предоставляет замечательную возможность, создавать запросы, используя язык JavaScript

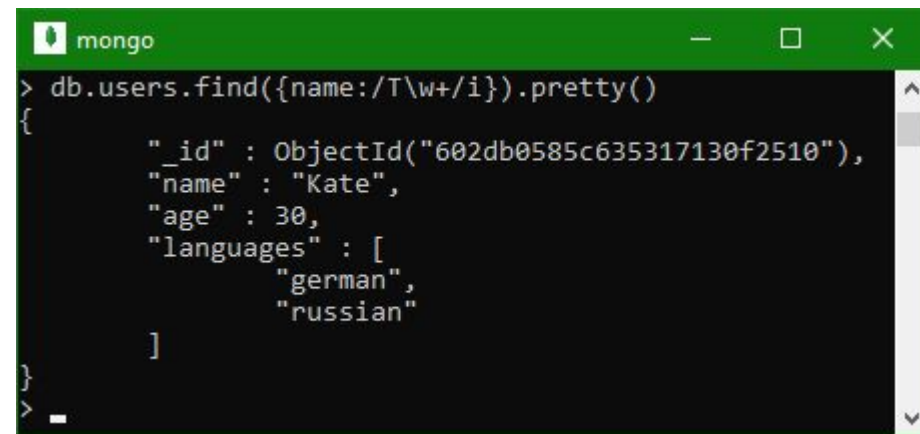


```
> fn = function() { return this.name=="Kate"; }  
function() { return this.name=="Kate"; }  
> db.users.find(fn).pretty()  
{  
  "_id" : ObjectId("602db0585c635317130f2510"),  
  "name" : "Kate",  
  "age" : 30,  
  "languages" : [  
    "german",  
    "russian"  
  ]  
}  
> function pow(n, t){ return n * t; }  
> pow(2, 3)  
6  
>
```



Использование регулярок в запросах

Еще одной замечательной возможностью при построении запросов является использование регулярных выражений.



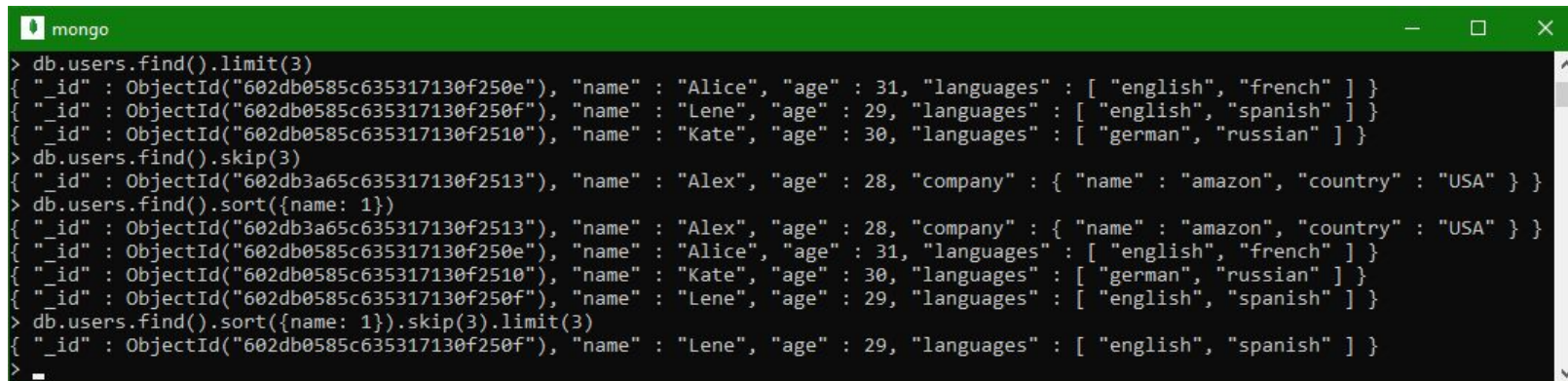
```
mongo
> db.users.find({name:/T\w+/i}).pretty()
{
  "_id" : ObjectId("602db0585c635317130f2510"),
  "name" : "Kate",
  "age" : 30,
  "languages" : [
    "german",
    "russian"
  ]
}
```



Настройка и сортировка запросов

MongoDB представляет ряд функций, которые помогают управлять выборкой из БД:

- **limit.** Задаёт максимально допустимое количество получаемых документов. Количество передается в виде числового параметра
- **skip.** Если хотим произвести выборку не сначала, а пропустив какое-то количество документов, то эта функция поможет
- **sort.** Передавая в эту функцию значения 1 или -1, мы можем указать в каком порядке сортировать: по возрастанию (1) или по убыванию (-1)

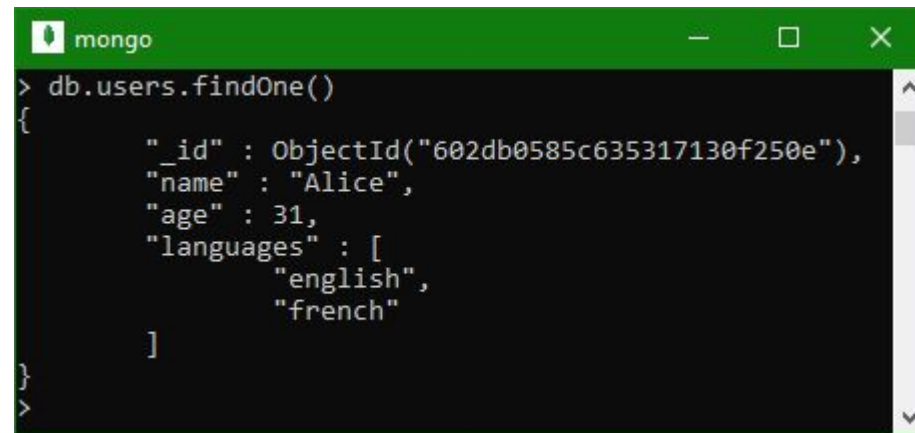


```
mongo
> db.users.find().limit(3)
{ "_id" : ObjectId("602db0585c635317130f250e"), "name" : "Alice", "age" : 31, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("602db0585c635317130f250f"), "name" : "Lene", "age" : 29, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("602db0585c635317130f2510"), "name" : "Kate", "age" : 30, "languages" : [ "german", "russian" ] }
> db.users.find().skip(3)
{ "_id" : ObjectId("602db3a65c635317130f2513"), "name" : "Alex", "age" : 28, "company" : { "name" : "amazon", "country" : "USA" } }
> db.users.find().sort({name: 1})
{ "_id" : ObjectId("602db3a65c635317130f2513"), "name" : "Alex", "age" : 28, "company" : { "name" : "amazon", "country" : "USA" } }
{ "_id" : ObjectId("602db0585c635317130f250e"), "name" : "Alice", "age" : 31, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("602db0585c635317130f2510"), "name" : "Kate", "age" : 30, "languages" : [ "german", "russian" ] }
{ "_id" : ObjectId("602db0585c635317130f250f"), "name" : "Lene", "age" : 29, "languages" : [ "english", "spanish" ] }
> db.users.find().sort({name: 1}).skip(3).limit(3)
{ "_id" : ObjectId("602db0585c635317130f250f"), "name" : "Lene", "age" : 29, "languages" : [ "english", "spanish" ] }
>
```

Поиск одиночного документа

findOne извлекает одиночный документ. Ее действие аналогично тому, как если бы мы использовали функцию `limit(1)`, которая также извлекает первый документ коллекции.

Комбинация функций `skip` и `limit` извлечет документ по нужному местоположению.



```
mongo
> db.users.findOne()
{
  "_id" : ObjectId("602db0585c635317130f250e"),
  "name" : "Alice",
  "age" : 31,
  "languages" : [
    "english",
    "french"
  ]
}
```

Сортировка по дате добавления

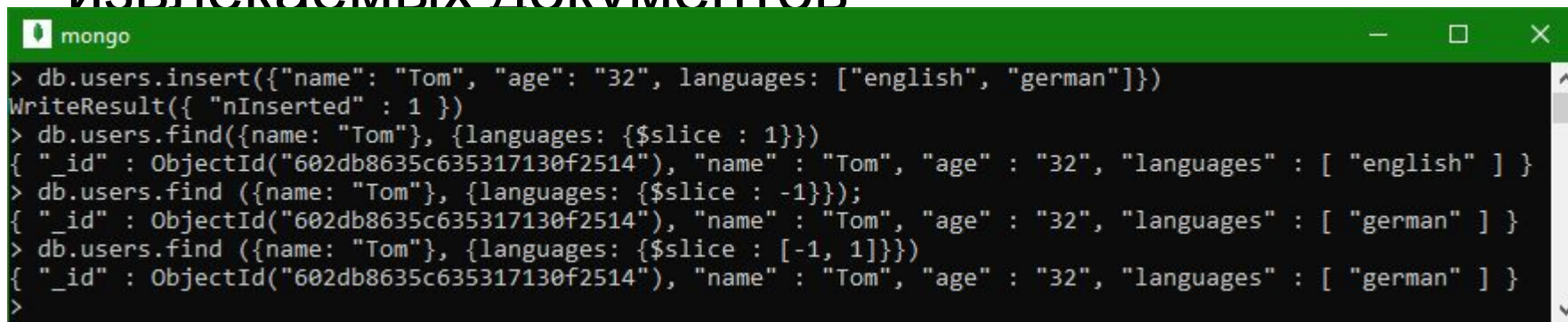
- **\$natural**. Этот параметр позволяет задать сортировку: документы передаются в том порядке, в каком они были добавлены в коллекцию, либо в обратном порядке.

```
mongo
> db.users.find().sort({ $natural: -1 }).limit(5)
{ "_id" : ObjectId("602db3a65c635317130f2513"), "name" : "Alex", "age" : 28, "company" : { "name" : "amazon", "country" : "USA" } }
{ "_id" : ObjectId("602db0585c635317130f2510"), "name" : "Kate", "age" : 30, "languages" : [ "german", "russian" ] }
{ "_id" : ObjectId("602db0585c635317130f250f"), "name" : "Lene", "age" : 29, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("602db0585c635317130f250e"), "name" : "Alice", "age" : 31, "languages" : [ "english", "french" ] }
```



Срезы в MongoDB

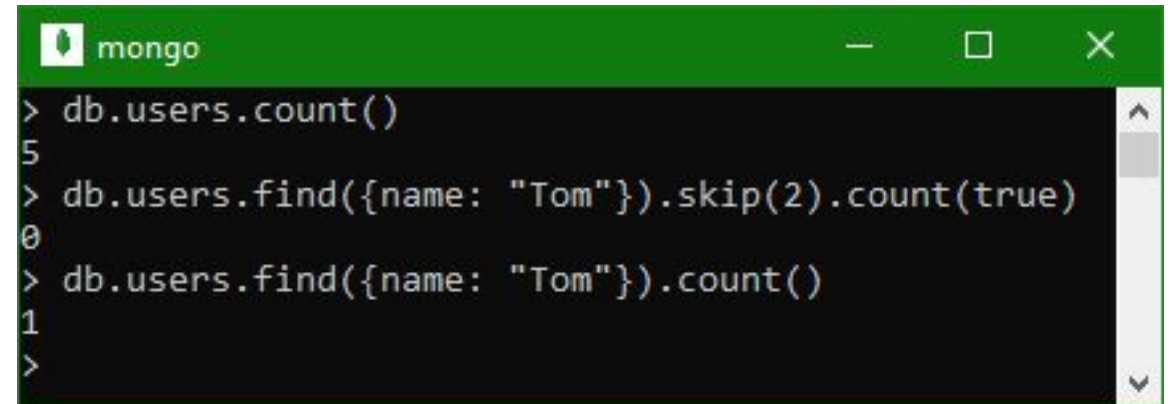
- **\$slice** является в некотором роде комбинацией функций `limit` и `skip`. Но в отличие от них `$slice` может работать с массивами. Принимает два параметра. Первый указывает на общее количество возвращаемых документов. Второй параметр необязательный, но если он используется, тогда первый параметр указывает на смещение относительно начала (как функция `skip`), а второй - на ограничение количества извлекаемых документов



```
mongo
> db.users.insert({"name": "Tom", "age": "32", "languages": ["english", "german"]})
WriteResult({ "nInserted" : 1 })
> db.users.find({name: "Tom"}, {languages: {$slice : 1}})
{ "_id" : ObjectId("602db8635c635317130f2514"), "name" : "Tom", "age" : "32", "languages" : [ "english" ] }
> db.users.find ({name: "Tom"}, {languages: {$slice : -1}});
{ "_id" : ObjectId("602db8635c635317130f2514"), "name" : "Tom", "age" : "32", "languages" : [ "german" ] }
> db.users.find ({name: "Tom"}, {languages: {$slice : [-1, 1]}})
{ "_id" : ObjectId("602db8635c635317130f2514"), "name" : "Tom", "age" : "32", "languages" : [ "german" ] }
>
```

Получаем количество элементов выборки

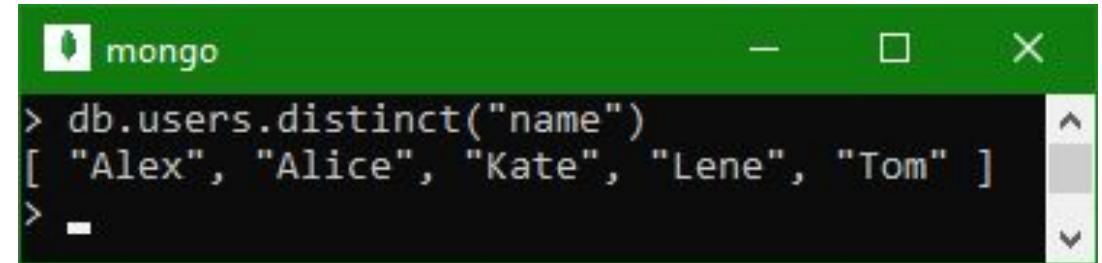
С помощью функции `count()` можно получить число элементов в коллекции. Можно комбинировать с параметрами `find`



```
mongo
> db.users.count()
5
> db.users.find({name: "Tom"}).skip(2).count(true)
0
> db.users.find({name: "Tom"}).count()
1
>
```

Получаем уникальные значения поля

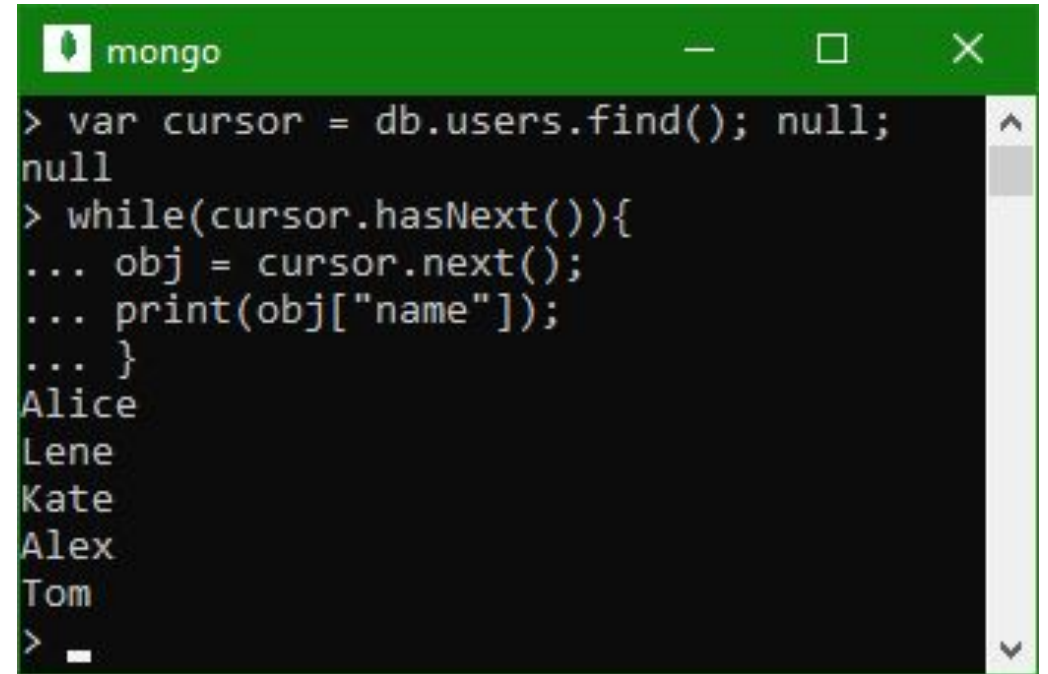
- **distinct.** В коллекции могут быть документы, которые содержат одинаковые значения для одного или нескольких полей. И нам надо найти только уникальные различающиеся значения для одного из полей документа. Для этого мы можем воспользоваться функцией `distinct`



```
mongo
> db.users.distinct("name")
[ "Alex", "Alice", "Kate", "Lene", "Tom" ]
> _
```

Курсоры

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

A screenshot of a MongoDB shell window titled 'mongo'. The window has a green title bar with standard window controls. The terminal shows a sequence of commands and their output. First, a cursor is created from the 'users' collection. Then, a while loop iterates over the cursor, printing the 'name' field of each document. The output shows the names Alice, Lene, Kate, Alex, and Tom.

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


Условные операторы

Условные операторы задают условие, которому должно соответствовать значение поля документа:

- `$eq` (равно)
- `$ne` (не равно)
- `$gt` (больше чем)
- `$lt` (меньше чем)
- `$gte` (больше или равно)
- `$lte` (меньше или равно)
- `$in` определяет массив значений, одно из которых должно иметь поле документа
- `$nin` определяет массив значений, которые не должно иметь поле документа

```
mongo
> db.users.find ({age: {$lt : 30}})
{ "_id" : ObjectId("602db0585c635317130f250f"), "name" : "Lene", "age" : 29, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("602db3a65c635317130f2513"), "name" : "Alex", "age" : 28, "company" : { "name" : "amazon", "country" : "USA" } }
> db.users.find ({age: {$gt : 30}})
{ "_id" : ObjectId("602db0585c635317130f250e"), "name" : "Alice", "age" : 31, "languages" : [ "english", "french" ] }
> db.users.find ({age: {$gt : 30, $lt: 50}})
{ "_id" : ObjectId("602db0585c635317130f250e"), "name" : "Alice", "age" : 31, "languages" : [ "english", "french" ] }
> db.users.find ({age: {$eq : 22}})
> db.users.find ({age: {$ne : 22}})
{ "_id" : ObjectId("602db0585c635317130f250e"), "name" : "Alice", "age" : 31, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("602db0585c635317130f250f"), "name" : "Lene", "age" : 29, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("602db0585c635317130f2510"), "name" : "Kate", "age" : 30, "languages" : [ "german", "russian" ] }
{ "_id" : ObjectId("602db3a65c635317130f2513"), "name" : "Alex", "age" : 28, "company" : { "name" : "amazon", "country" : "USA" } }
{ "_id" : ObjectId("602db8635c635317130f2514"), "name" : "Tom", "age" : 32, "languages" : [ "english", "german" ] }
```


Логические операторы

Логические операторы выполняются над условиями выборки:

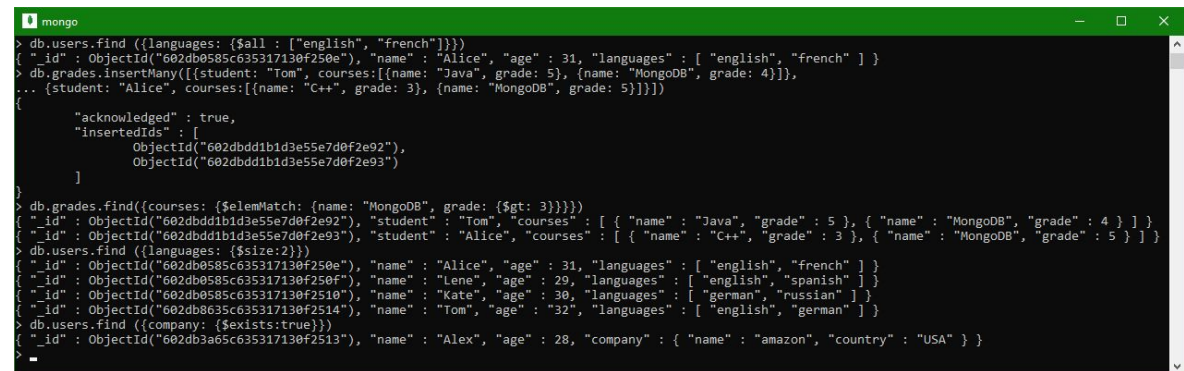
- `$or`: соединяет два условия, и документ должен соответствовать одному из этих условий
- `$and`: соединяет два условия, и документ должен соответствовать обоим условиям
- `$not`: документ должен НЕ соответствовать условию
- `$nor`: соединяет два условия, и документ должен НЕ соответствовать обоим условиям

```
mongo
> db.users.find ({$or : [{name: "Tom"}, {age: 22}]})
{ "_id" : ObjectId("602db8635c635317130f2514"), "name" : "Tom", "age" : "32", "languages" : [ "english", "german" ] }
> db.users.find ({$or : [{name: "Tom"}, {age: 22}, {languages: "german"}]})
{ "_id" : ObjectId("602db8635c635317130f2514"), "name" : "Tom", "age" : "32", "languages" : [ "english", "german" ] }
> db.users.find ({$or : [{name: "Tom"}, {age: {$gte:30}}]})
{ "_id" : ObjectId("602db0585c635317130f250e"), "name" : "Alice", "age" : 31, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("602db0585c635317130f2510"), "name" : "Kate", "age" : 30, "languages" : [ "german", "russian" ] }
{ "_id" : ObjectId("602db8635c635317130f2514"), "name" : "Tom", "age" : "32", "languages" : [ "english", "german" ] }
> db.users.find ({$and : [{name: "Tom"}, {age: 32}]})
>
```

Поиск по массивам

Ряд операторов предназначены для работы с массивами:

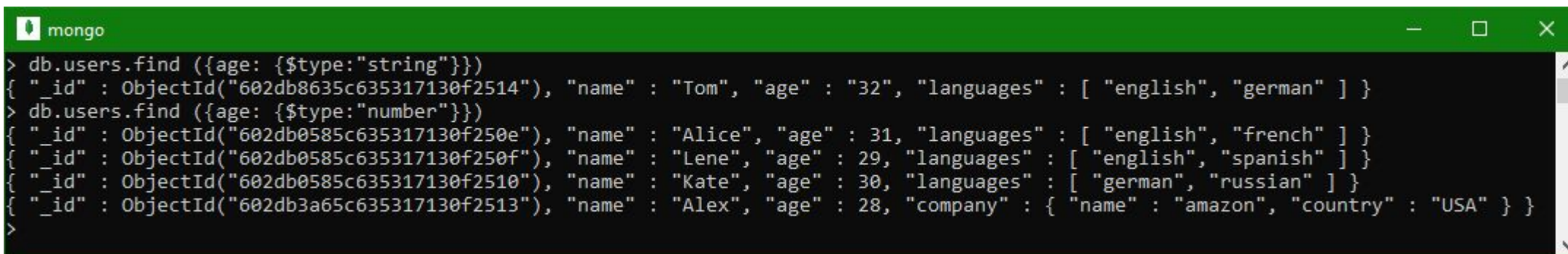
- `$all`: определяет набор значений, которые должны иметься в массиве
- `$size`: определяет количество элементов, которые должны быть в массиве
- `$elemMatch`: определяет условие, которым должны соответствовать элементы в массиве



```
mongo
> db.users.find ({languages: {$all: ["english", "french"]}})
{ "_id" : ObjectId("602db0585c635317130f250e"), "name" : "Alice", "age" : 31, "languages" : [ "english", "french" ] }
> db.grades.insertMany([{$student: "Tom", courses:[{name: "Java", grade: 5}, {name: "MongoDB", grade: 4}]},
... {student: "Alice", courses:[{name: "C++", grade: 3}, {name: "MongoDB", grade: 5}]}])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("602dbdd1b1d3e55e7d0f2e92"),
    ObjectId("602dbdd1b1d3e55e7d0f2e93")
  ]
}
> db.grades.find({courses: {$elemMatch: {name: "MongoDB", grade: {$gt: 3}}}})
{ "_id" : ObjectId("602dbdd1b1d3e55e7d0f2e92"), "student" : "Tom", "courses" : [ { "name" : "Java", "grade" : 5 }, { "name" : "MongoDB", "grade" : 4 } ] }
{ "_id" : ObjectId("602dbdd1b1d3e55e7d0f2e93"), "student" : "Alice", "courses" : [ { "name" : "C++", "grade" : 3 }, { "name" : "MongoDB", "grade" : 5 } ] }
> db.users.find ({languages: {$size:2}})
{ "_id" : ObjectId("602db0585c635317130f250e"), "name" : "Alice", "age" : 31, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("602db0585c635317130f250f"), "name" : "Lene", "age" : 29, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("602db0585c635317130f2510"), "name" : "Kate", "age" : 30, "languages" : [ "german", "russian" ] }
{ "_id" : ObjectId("602db8635c635317130f2514"), "name" : "Tom", "age" : "32", "languages" : [ "english", "german" ] }
> db.users.find ({company: {$exists:true}})
{ "_id" : ObjectId("602db3a65c635317130f2513"), "name" : "Alex", "age" : 28, "company" : { "name" : "amazon", "country" : "USA" } }
```

Оператор \$type

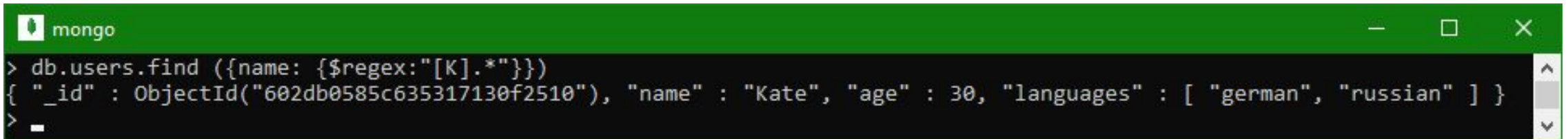
Оператор **\$type** извлекает только те документы, в которых определенный ключ имеет значение определенного типа



```
mongo
> db.users.find ({age: {$type:"string"}})
{ "_id" : ObjectId("602db8635c635317130f2514"), "name" : "Tom", "age" : "32", "languages" : [ "english", "german" ] }
> db.users.find ({age: {$type:"number"}})
{ "_id" : ObjectId("602db0585c635317130f250e"), "name" : "Alice", "age" : 31, "languages" : [ "english", "french" ] }
{ "_id" : ObjectId("602db0585c635317130f250f"), "name" : "Lene", "age" : 29, "languages" : [ "english", "spanish" ] }
{ "_id" : ObjectId("602db0585c635317130f2510"), "name" : "Kate", "age" : 30, "languages" : [ "german", "russian" ] }
{ "_id" : ObjectId("602db3a65c635317130f2513"), "name" : "Alex", "age" : 28, "company" : { "name" : "amazon", "country" : "USA" } }
```

Ищем с помощью регулярных выражений

- `$regex` задает регулярное выражение, которому должно соответствовать значение поля. Например, пусть поле `name` обязательно имеет букву "b":

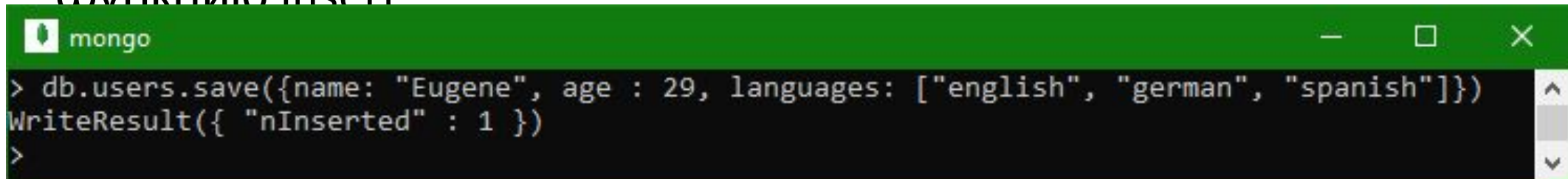


```
mongo
> db.users.find ({name: {$regex:"[K].*"}})
{ "_id" : ObjectId("602db0585c635317130f2510"), "name" : "Kate", "age" : 30, "languages" : [ "german", "russian" ] }
```



Обновление данных

- **save()**. MongoDB предоставляет возможность обновления данных. В качестве параметра этот метод принимает документ
- В этот документ в качестве поля можно передать параметр `_id`. Если метод находит документ с таким значением `_id`, то документ обновляется. Если же с подобным `_id` нет документов, то документ вставляется
- Если параметр `_id` не указан, то документ вставляется, а параметр `_id` генерируется автоматически как при обычном добавлении через функцию `insert`

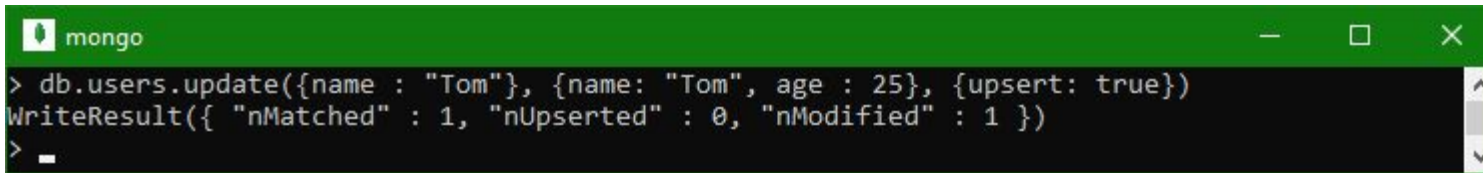


```
> db.users.save({name: "Eugene", age : 29, languages: ["english", "german", "spanish"]})
WriteResult({ "nInserted" : 1 })
>
```

Обновление данных

Более детальную настройку при обновлении предлагает функция **update**. Она принимает три параметра:

- **query**: принимает запрос на выборку документа, который надо обновить
- **objNew**: представляет документ с новой информацией, который заместит старый при обновлении
- **options**: определяет дополнительные параметры при обновлении документов. Может принимать два аргумента: `upsert` и `multi`.
 - Если `upsert` `true`, то `mongodb` будет обновлять документ, если он найден, и создавать новый, если такого документа нет. Если `false`, то не будет создавать новый документ, если запрос на выборку не найдет ни одного документа.
 - Параметр `multi` указывает, должен ли обновляться первый элемент в выборке (используется по умолчанию, если данный параметр не указан) или же должны обновляться все документы в выборке.

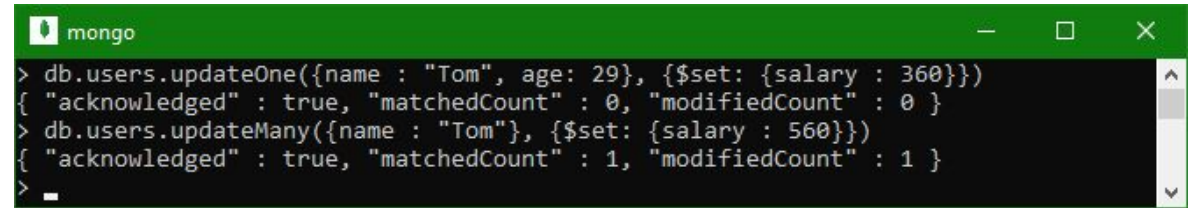


```
> db.users.update({name : "Tom"}, {name: "Tom", age : 25}, {upsert: true})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> _
```



Обновление данных

- **updateOne** похож на метод **update** за тем исключением, что он обновляет только один документ.
- Если необходимо обновить все документы, соответствующие некоторому критерию, то применяется метод **updateMany()**



```
mongo
> db.users.updateOne({name : "Tom", age: 29}, {$set: {salary : 360}})
{ "acknowledged" : true, "matchedCount" : 0, "modifiedCount" : 0 }
> db.users.updateMany({name : "Tom"}, {$set: {salary : 560}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

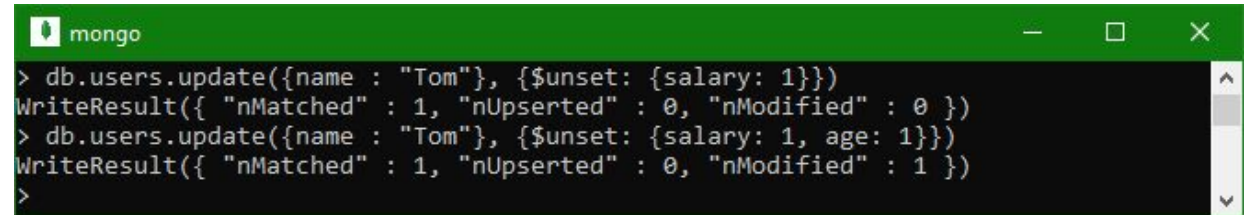

Обновление отдельного поля

- **\$set.** Используется для обновления значения одного ключей. Если документ не содержит обновляемое поле, то оно создается.
- Для простого увеличения значения числового поля на определенное количество единиц применяется оператор **\$inc**. Если документ не содержит обновляемое поле, то оно создается. Данный оператор применим только к числовым значениям.

```
mongo
> db.users.update({name : "Tom", age: 29}, {$set: {age : 30}})
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
> db.users.update({name : "Tom", age: 29}, {$set: {salary : 300}})
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
> db.users.update({name : "Tom"}, {$set: {name: "Tom", age : 25}}, {multi:true})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
> db.users.update({name : "Tom"}, {$inc: {age:2}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> -
```


Удаление поля

- **\$unset** используется для удаления отдельного ключа
- Если вдруг подобного ключа в документе не существует, то оператор не оказывает никакого влияния. Также можно удалять сразу несколько полей



```
mongo
> db.users.update({name : "Tom"}, {$unset: {salary: 1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
> db.users.update({name : "Tom"}, {$unset: {salary: 1, age: 1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

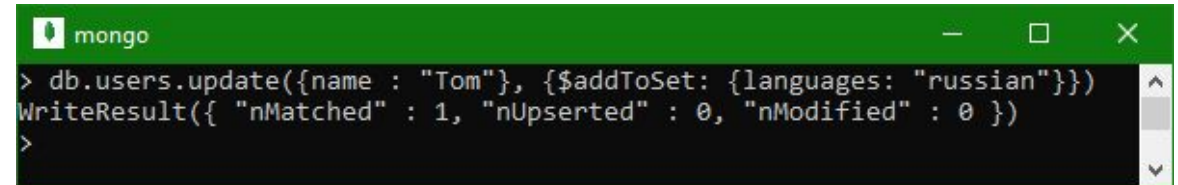
Обновление массивов

- **\$push** позволяет добавить еще одно значение к уже существующему.
 - Если ключ, для которого мы хотим добавить значение, не представляет массив, то мы получим ошибку Cannot apply \$push/\$pushAll modifier to non-array.
- **\$each** позволяет добавить сразу несколько значений
- Еще пара операторов позволяет настроить вставку. Оператор **\$position** задает позицию в массиве для вставки элементов, а оператор **\$slice** указывает, сколько элементов оставить в массиве после вставки.

```
mongo
> db.users.updateOne({name : "Tom"}, {$push: {languages: "russian"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.users.update({name : "Tom"}, {$push: {languages: {$each: ["russian", "spanish", "italian"]}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.update({name : "Tom"}, {$push: {languages: {$each: ["german", "spanish", "italian"], $position:1, $slice:5}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> _
```

Обновление массивов

- Оператор **\$addToSet** подобно оператору **\$push** добавляет объекты в массив. Отличие состоит в том, что **\$addToSet** добавляет данные, если их еще нет в массиве:



```
mongo
> db.users.update({name : "Tom"}, {$addToSet: {languages: "russian"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
>
```

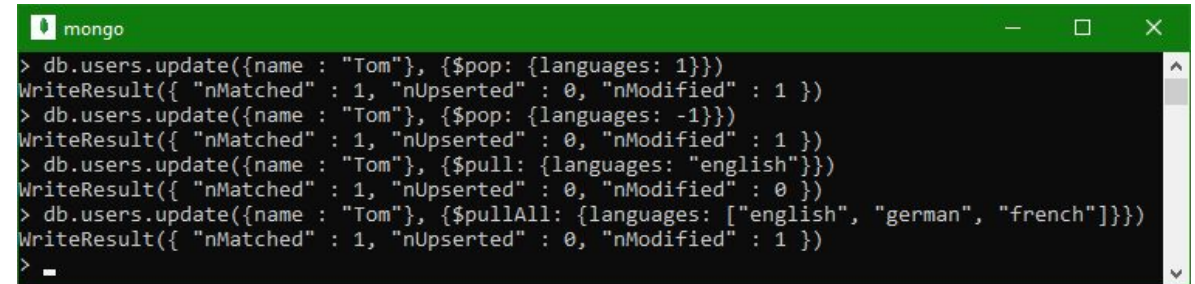
Удаление элемента из массива

Оператор **\$pop** позволяет удалять элемент из массива:

- Указывая для ключа `languages` значение `1`, мы удаляем первый элемент с конца. Чтобы удалить первый элемент сначала массива, надо передать отрицательное значение

Оператор **\$pull** удаляет каждое вхождение элемента в массив. Например, через оператор `$push` мы можем добавить одно и то же значение в массив несколько раз. И теперь с помощью `$pull` удалим его

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

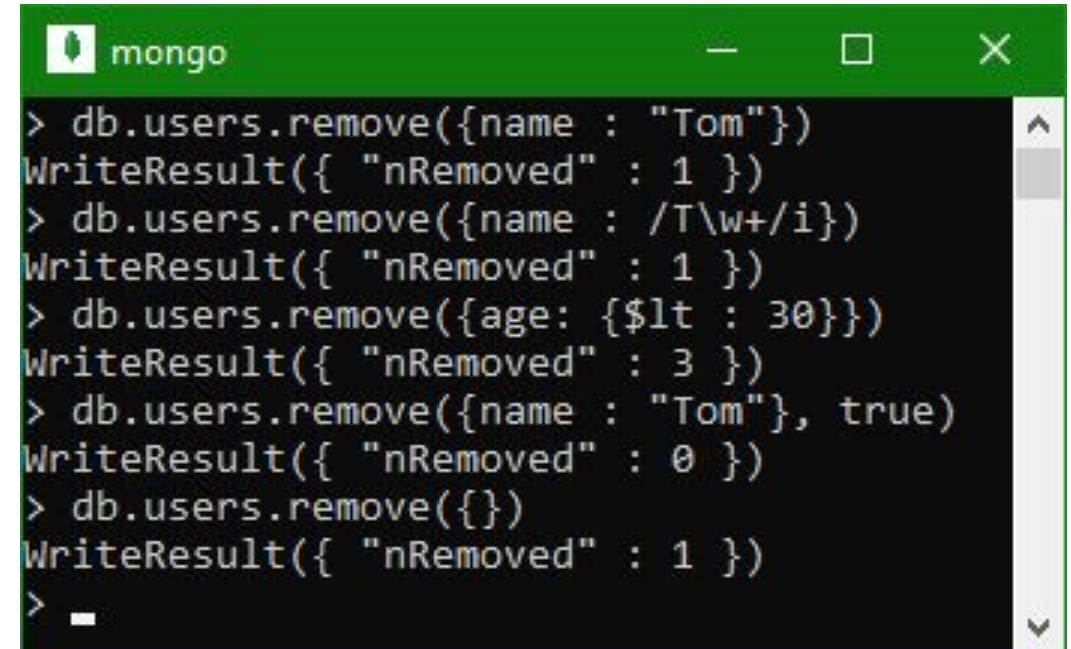


```
mongo
> db.users.update({name : "Tom"}, {$pop: {languages: 1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.update({name : "Tom"}, {$pop: {languages: -1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.update({name : "Tom"}, {$pull: {languages: "english"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
> db.users.update({name : "Tom"}, {$pullAll: {languages: ["english", "german", "french"]}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> -
```

Удаление документов

Для удаления документов в MongoDB предусмотрен метод **remove**:

- возвращает объект `WriteResult`
- В итоге все найденные документы будут удалены. Причем, как и в случае с `find`, мы можем задавать условия выборки для удаления различными способами (в виде регулярных выражений, в виде условных конструкций и т.д.):
- Может принимать второй необязательный параметр булевого типа, который указывает, надо удалять один элемент или все элементы, соответствующие условию. Если этот параметр равен `true`, то удаляется только один элемент. По умолчанию он равен `false`
- Чтобы удалить разом все документы из коллекции, надо оставить пустым параметр запроса

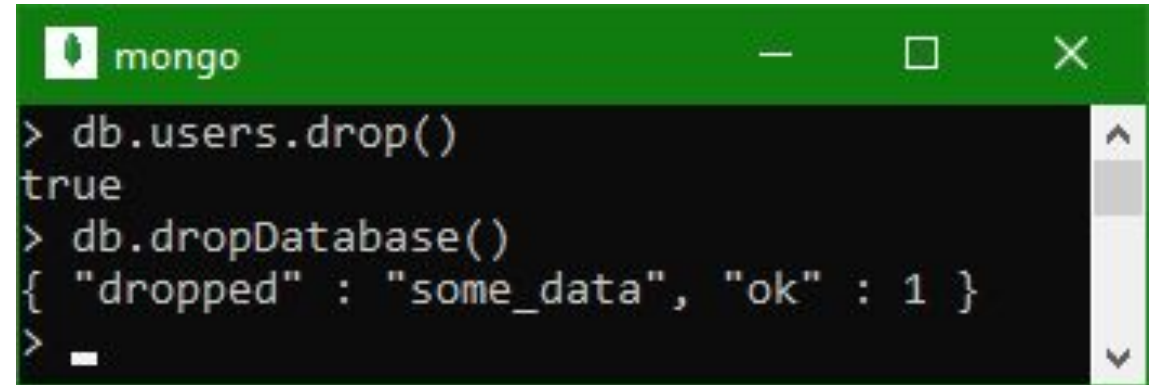


```
mongo
> db.users.remove({name : "Tom"})
WriteResult({ "nRemoved" : 1 })
> db.users.remove({name : /T\w+/i})
WriteResult({ "nRemoved" : 1 })
> db.users.remove({age: {$lt : 30}})
WriteResult({ "nRemoved" : 3 })
> db.users.remove({name : "Tom"}, true)
WriteResult({ "nRemoved" : 0 })
> db.users.remove({})
WriteResult({ "nRemoved" : 1 })
> _
```

Удаление коллекций и баз данных

Для удаления коллекций используется функция **drop**. И если удаление коллекции пройдет успешно, то консоль выведет `true`.

Чтобы удалить всю базу данных, надо воспользоваться функцией **dropDatabase()**



```
> db.users.drop()
true
> db.dropDatabase()
{ "dropped" : "some_data", "ok" : 1 }
>
```


Ручная установка ссылок

В реляционных базах данных можно устанавливать внешние ключи, когда поля из одной таблицы ссылаются на поля в другой таблице. В MongoDB также можно устанавливать ссылки.

Ручная установка ссылок сводится к присвоению значения поля `_id` одного документа полю другого документа. Допустим, у нас могут быть коллекции, представляющие компании и работников, работающих в этих компаниях. Итак, сначала добавим в коллекцию `companies` документ представляющий компанию:

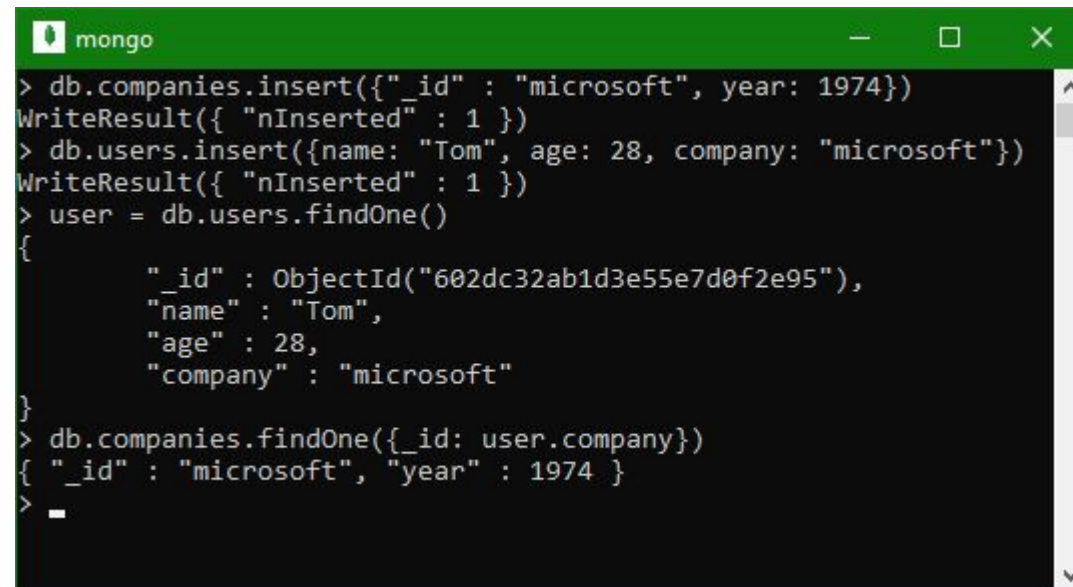
Теперь добавим в коллекцию `persons` документ, представляющий работника. В этом документе будет поле `company`, представляющее компанию, где работает работник. И очень важно, что в качестве значения для этого поля мы устанавливаем не объект `company`, а значение ключа `_id` добавленного выше документа.

Теперь при получении документа из коллекции `users`.

В данном случае имеется в виду, что выше добавленный элемент будет единственным в коллекции.

После этого консоль выводит полученный документ. И теперь найдем ссылку на его компанию в коллекции `companies`.

И если документ с таким идентификатором обнаружен, он отображается на консоли.



```
mongo
> db.companies.insert({"_id" : "microsoft", year: 1974})
WriteResult({ "nInserted" : 1 })
> db.users.insert({name: "Tom", age: 28, company: "microsoft"})
WriteResult({ "nInserted" : 1 })
> user = db.users.findOne()
{
  "_id" : ObjectId("602dc32ab1d3e55e7d0f2e95"),
  "name" : "Tom",
  "age" : 28,
  "company" : "microsoft"
}
> db.companies.findOne({_id: user.company})
{ "_id" : "microsoft", "year" : 1974 }
```



Автоматическое связывание

Используя функциональность DBRef, мы можем установить автоматическое связывание между документами. Посмотрим на примере применение данной функциональности. Вначале добавим новый документ в коллекцию companies.

Обратите внимание, что в данном случае сохранение идет с помощью метода save, не insert. Метод save при добавлении нового документа генерирует _id. И после сохранения мы можем вывести документ на консоль: > apple

Теперь создадим новый документ для коллекции person, у которого ключ company свяжем с только что добавленным документом apple.

Посмотрев на примере, теперь разберем организацию ссылок между документами. Для связывания с документом apple использовалось следующее выражение company: new DBRef('companies', apple._id)). Формальный синтаксис DBRef следующий:

- { "\$ref" : название_коллекции, "\$id": значение [, "\$db" : название_бд] }

Первый параметр \$ref указывает на коллекцию, где хранится связанный документ. Вторым параметром указывает на значение, которое и будет представлять что-то типа внешнего ключа. Третий необязательный параметр указывает на базу данных.

При тестировании в качестве запроса на выборку указывается выражение _id: steve.company.\$id. Так как person.company представляет теперь объект new DBRef('companies', apple._id)), то нам надо конкретизировать параметр steve.company.\$id

```
mongo
> apple=({"name" : "apple", "year": 1976})
{ "name" : "apple", "year" : 1976 }
> db.companies.save(apple)
> wrsteve = ({ "name": "Steve", "age": 25, company: new DBRef('companies', apple._id)})
{ steve = ({ "name": "Steve", "age": 25, company: new DBRef('companies', apple._id)})
  "name" : "Steve",
  "age" : 25,
  "company" : DBRef("companies", ObjectId("602dc383b1d3e55e7d0f2e96"))
}
> db.users.save(steve)
WriteResult({ "nInserted" : 1 })
> db.companies.findOne({ _id: steve.company.$id })
{
  "_id" : ObjectId("602dc383b1d3e55e7d0f2e96"),
  "name" : "apple",
  "year" : 1976
}
```



Настройка индексов

При поиске документов в небольших коллекциях мы не испытаем особых проблем. Однако когда коллекции содержат миллионы документов, а нам надо сделать выборку по определенному полю, то поиск нужных данных может занять некоторое время, которое может оказаться критичным для нашей задачи. В этом случае нам могут помочь индексы. Индексы позволяют упорядочить данные по определенному полю, что впоследствии ускорит поиск. Например, если мы в своем приложении или задаче, как правило, выполняем поиск по полю `name`, то мы можем индексировать коллекцию по этому полю. Таким образом с помощью метода `createIndex` устанавливается индекс по полю `name`. MongoDB позволяет установить до 64 индексов на одну коллекцию.

Если мы просто определим индекс для коллекции, например, `db.users.createIndex({"name": 1})`, то мы все еще сможем добавлять в коллекцию документы с одинаковым значением ключа `name`. Однако, если нам потребуется, чтобы в коллекцию можно было добавлять документ с одним и тем же значением ключа только один раз, мы можем установить флаг `unique`.

Теперь, если мы попытаемся добавить в коллекцию два документа с одним и тем же значением `name`, то мы получим ошибку. В тоже время тут есть свои тонкости. Так, документ может не иметь ключа `name`. В этом случае для добавляемого документа автоматически создается ключ `name` со значением `null`. Поэтому при добавлении второго документа, в котором не определен ключ `name`, будет выброшено исключение, так как ключ `name` со значением `null` уже присутствует в коллекции.

Также можно задать уникальный индекс сразу для двух полей, но в этом случае все добавляемые документы должны иметь уникальные значения для обоих полей.

Кроме того, тут есть свои ограничения. Например, значение поля, по которому идет индексация, не должно быть больше 1024 байт.

```
mongo
> db.users.createIndex({"name" : 1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
> db.users.createIndex({"name" : 1}, {"unique" : true})
{
  "ok" : 0,
  "errmsg" : "Index with name: name_1 already exists with different options",
  "code" : 85,
  "codeName" : "IndexOptionsConflict"
}
> db.users.createIndex({"name" : 1, "age" : 1}, {"unique" : true})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 2,
  "numIndexesAfter" : 3,
  "ok" : 1
}
> db.system.indexes.find()
> db.users.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_"
  },
  {
    "v" : 2,
    "key" : {
      "name" : 1
    },
    "name" : "name_1"
  },
  {
    "v" : 2,
    "unique" : true,
    "key" : {
      "name" : 1,
      "age" : 1
    },
    "name" : "name_1_age_1"
  }
]
```



Явное создание коллекции

Коллекцию можно создать явным образом, применив метод `db.createCollection(name, options)`, где `name` - название коллекции, а `options` - необязательный объект с дополнительными настройками инициализации.

В процессе работы, возможно, потребуется изменить название коллекции. Например, если при первом добавлении данных в ее названии была опечатка. И чтобы не удалять и затем пересоздавать коллекцию, следует использовать функцию `renameCollection`.

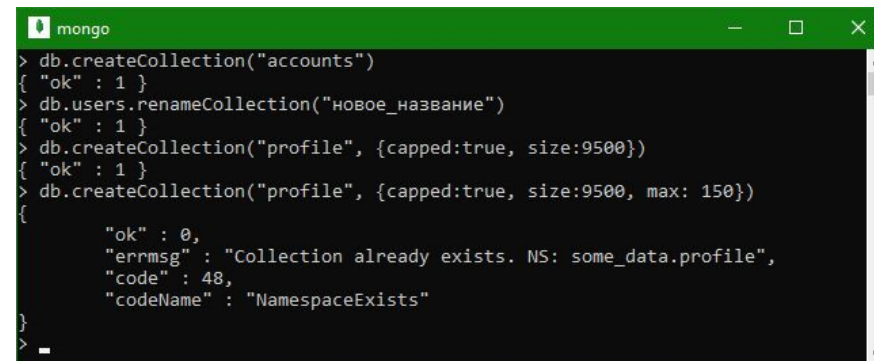
Когда мы отправляем запрос к бд на выборку, то MongoDB возвращает нам документы в том порядке, как правило, в котором они были добавлены. Однако такой порядок не всегда гарантируется, так как данные могут быть удалены, перемещены, изменены. Поэтому в MongoDB существует понятие ограниченной коллекции (`capped collection`). Подобная коллекция гарантирует, что документы будут располагаться в том же порядке, в котором они добавлялись в коллекцию. Ограниченные коллекции имеют фиксированный размер. И когда в коллекции уже нет места, наиболее старые документы удаляются, и в конец добавляются новые данные.

В отличие от обычных коллекций ограниченные мы можем задать явным образом. Например, создадим ограниченную коллекцию с названием `profile` и зададим для нее размер в 9500 байт:

Также можно ограничить количество документов в коллекции, указав его в параметре `max`. Однако при таком способе создания коллекции следует учитывать, что если все место под коллекцию заполнено (например, выделенные нами 9500 байтов), а количество документов еще не достигло максимума, в данном случае 150, то в этом случае при добавлении нового документа самый старый документ будет удаляться, а на его место будет вставляться новый документ.

При обновлении документов в таких коллекциях надо помнить, что документы не должны расти в размерах, иначе обновление не удастся произвести.

Также нельзя удалять документы из подобных коллекций, можно только удалить всю коллекцию.



```
mongo
> db.createCollection("accounts")
{ "ok" : 1 }
> db.users.renameCollection("новое_название")
{ "ok" : 1 }
> db.createCollection("profile", {capped:true, size:9500})
{ "ok" : 1 }
> db.createCollection("profile", {capped:true, size:9500, max: 150})
{
  "ok" : 0,
  "errmsg" : "Collection already exists. NS: some_data.profile",
  "code" : 48,
  "codeName" : "NamespaceExists"
}
```





Ваши вопросы

что необходимо прояснить в рамках
данного раздела



Работа с MongoDB в Python

основы работы с пакетом pymongo



Установка pymongo

Чтобы работать с MongoDB в Python, необходимо установить пакет pymongo:

- `pip install pymongo`



Работа с MongoDB в Python

Чтобы подключиться к MongoDB в Python, нужно создать клиентское соединение с помощью класса MongoClient.

```
1 import pymongo
2
3
4 client = pymongo.MongoClient('localhost', 27017)
```



Получаем объект БД

Чтобы начать работать с некоторой базой данных, ее надо получить с помощью вызова атрибута соединения.

Далее для работы с коллекцией, ее надо также получить по атрибуту.

```
1  import pymongo
2
3
4  client = pymongo.MongoClient('localhost', 27017)
5
6
7  # или database = client['users']
8  database = client.users
9
10
11 # или collection = database['users']
12 collection = database.users
```



Проверка существования базы данных

Чтобы проверить, существует ли некоторая БД, нужно проверить, если ли она в `client.list_database_names()`

```
1 import pymongo
2
3 client = pymongo.MongoClient('localhost', 27017)
4
5
6 db_list = client.list_database_names()
7 if "users" in db_list:
8     print("The database exists.")
```



Проверка существования коллекции в БД

Чтобы проверить, существует ли некоторая коллекция в БД, нужно проверить, если ли она в `database.list_collection_names()`

```
1 import pymongo
2
3 client = pymongo.MongoClient('localhost', 27017)
4
5 # или database = client['users']
6 database = client.users
7
8 collection_list = database.list_collection_names()
9 if "users" in collection_list:
10     print("The collection exists.")
```



Добавляем документ в коллекцию

Чтобы добавить элемент в коллекцию, необходимо вызвать метод `insert_one` объекта `collection`

```
1 import pymongo
2
3
4 client = pymongo.MongoClient('localhost', 27017)
5
6
7 # или database = client['users']
8 database = client.users
9
10
11 # или collection = database['users']
12 collection = database.users
13
14
15 user_data = {
16     'login': 'bestLoginEver',
17     'name': 'Name Surname'
18 }
19 result = collection.insert_one(user_data)
20 print(f"Added user ID: {result.inserted_id}")
```



Добавляем документ в коллекцию

Чтобы добавить несколько документов в коллекцию, необходимо вызвать метод `insert_many` объекта `collection`

```
1 import pymongo
2
3 client = pymongo.MongoClient('localhost', 27017)
4
5 # или database = client['users']
6 database = client.users
7
8 # или collection = database['users']
9 collection = database.users
10
11 users_list = [
12     {
13         'login': 'bestLoginEver',
14         'name': 'Name Surname'
15     },
16     {
17         'login': 'bestLoginEver',
18         'name': 'Name Surname'
19     }
20 ]
21
22 result = collection.insert_many(users_list)
23 print(f"Added user IDs: {result.inserted_ids}")
```



Поиск документ в коллекции

Для поиска можно воспользоваться методами:

- `find_one`
- `find`

```
1 import pymongo
2
3 client = pymongo.MongoClient('localhost', 27017)
4
5 # или database = client['users']
6 database = client.users
7
8 # или collection = database['users']
9 collection = database.users
10
11 user = collection.find_one({'login': 'login1'})
12 print(user)
```



Сортировка найденных значений

Чтобы отсортировать найденные документы, надо к результату find применить метод sort

```
1  import pymongo
2
3  client = pymongo.MongoClient('localhost', 27017)
4
5  # или database = client['users']
6  database = client.users
7
8  # или collection = database['users']
9  collection = database.users
10
11
12 documents = collection.find().sort('login')
13
14 for user in documents:
15     print(user)
16
```



Обновление документов

Для обновления данных документов можно воспользоваться методами:

- `collection.update_one`
- `collection.update_many`

```
1 import pymongo
2
3 client = pymongo.MongoClient("mongodb://localhost:27017/")
4 database = client["mydatabase"]
5 collection = database["customers"]
6
7
8 query = { "address": "Valley 345" }
9 new_values = { "$set": { "address": "Canyon 123" } }
10
11 collection.update_one(query, new_values)
12
13
14 query = { "address": { "$regex": "^S" } }
15 new_values = { "$set": { "name": "Minnie" } }
16
17 x = collection.update_many(query, new_values)
18 print(x.modified_count, "documents updated.")
```



Удаление документов

Для удаления документов из коллекции можно воспользоваться двумя методами:

- `collection.delete_one`
- `collection.delete_many`

```
1 import pymongo
2
3 client = pymongo.MongoClient("mongodb://localhost:27017/")
4 database = client["mydatabase"]
5 collection = database["customers"]
6
7 query = { "address": "Mountain 21" }
8
9 collection.delete_one(query)
10
11 query = {"address": {"$regex": "^S"}}
12
13 x = collection.delete_many(query)
14 print(x.deleted_count, " documents deleted.")
```

