

ЦИФРОВАЯ
КУЛЬТУРА
УНИВЕРСИТЕТ ИТМО

MongoDB

Высшая школа цифровой культуры
Университет ИТМО

dc@itmo.ru

Содержание

1 Основы MongoDB	2
2 Начало работы с MongoDB и управление базами	7
3 Выборка данных из коллекций	13
4 Фильтры, сортировка и агрегирование	19
5 Редактирование данных и индексы	28

1 Основы MongoDB

В этой лекции мы познакомим слушателей с основами документного хранилища данных – системой MongoDB. Почему именно MongoDB? Потому, что именно MongoDB (согласно упомянутому ранее рейтингу) принято считать наиболее ярким представителем группы документных хранилищ.

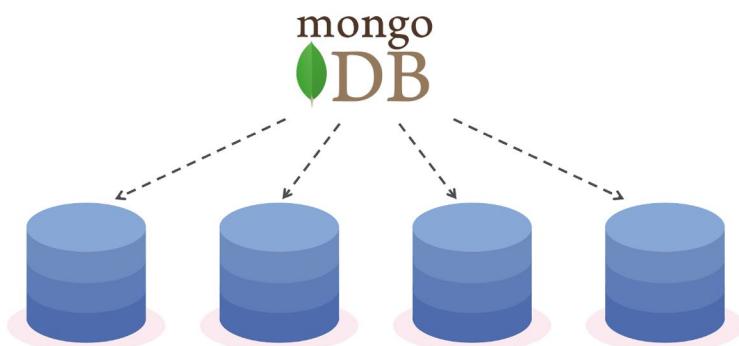
Почему именно MongoDB?

MongoDB - наиболее яркий представитель
группы документных хранилищ.

<https://www.mongodb.com/>
- официальный сайт MongoDB.



MongoDB представляет собой систему для управления документно-ориентированными базами данных, которая реализует новый подход к построению баз данных, где нет таблиц, схем, запросов SQL, внешних ключей и многих других объектов, присущих традиционным реляционным базам данных. MongoDB в некоторых случаях работает быстрее, обладает лучшей масштабируемостью, и, наконец, ее легче использовать. При этом важно понимать, что задачи бывают разные и методы их решения бывают разные. В какой-то ситуации MongoDB действительно улучшит производительность приложения, например, если надо хранить сложные по структуре данные. В



другой же ситуации лучше будет использовать традиционные реляционные базы данных. Вся система MongoDB может представлять не только одну базу данных, находящуюся на одном физическом сервере. Особенность архитектуры MongoDB позволяет расположить несколько баз данных на одном или

нескольких физических серверах, и эти базы данных смогут легко обмениваться данными и даже сохранять целостность.

Базу данных MongoDB можно представить в виде иерархической структуры как это указано на рисунке. Если в реляционных базах данные хранятся в таблицах, то в MongoDB база данных состоит из коллекций. Каждая коллекция имеет свое уникальное имя – произвольный идентификатор, состоящий не более чем 128 различных алфавитно-цифровых символов и знака подчеркивания. Коллекции в свою очередь состоят из документов. Именно

Структура базы данных MongoDB



документы хранят полезную информацию. В некотором смысле документы подобны строкам в реляционных базах, где строки хранят информацию об отдельных объектах.

В отличие от строк документы могут хранить сложную по структуре информацию, которую принято называть слабоструктурированной. Документ

Сравнение основных понятий MongoDB и реляционных баз

Реляционная база	MongoDB
База данных	База данных
Таблица	Коллекция
Строка (запись) в таблице	Документ



можно представить как хранилище ключей и значений. Ключ представляет собой простую метку, с которой ассоциирован определенный фрагмент данных из документа. Для каждого документа имеется свой уникальный ключ,

который называется `_id`. И если явным образом не указать его значение, то MongoDB сгенерирует для него значение автоматически.

Как уже упоминалось ранее, одним из популярных стандартов обмена данными и их хранения является формат **JSON** (Java Script Object Notation). JSON эффективно описывает сложные по структуре данные. Способ хранения данных в MongoDB в этом плане похож на JSON, хотя формально JSON не используется. Для хранения в MongoDB применяется формат, который

Формат хранения данных в MongoDB

JSON - Java Script Object Notation

BSON - Binary Java Script Object Notation



называется **BSON** (БиСон) или сокращение от binary JSON. Формат BSON позволяет работать с данными быстрее: быстрее выполняется поиск и обработка. Хотя надо отметить, что BSON в отличие от хранения данных в формате JSON имеет небольшой недостаток: в целом данные в JSON-формате занимают меньше места, чем в формате BSON, но, с другой стороны, данный недостаток окупается скоростью. На рисунке Вы можете видеть пример документа в формате JSON.

Всему документу соответствует уникальный идентификатор, т.е. ключ `_id` со значением, равным 1. Отдельные поля в документе также представлены в виде набора пар ключ-значение. Например, ключу "CartoonName" соответствует одно значение "The Lion King", а ключу "FilmDirectorName" массив значений: ["Roger Allers", "Rob Minkoff"].

Пример документа MongoDB

```
{  
  "_id": 1,  
  "CartoonName": "The Lion King",  
  "Year": 1994,  
  "Country": "USA",  
  "Duration": 88,  
  "FilmDirectorName": ["Roger Allers", "Rob Minkoff"]  
  "SongWriterName": ["Elton John", "Hans Zimmer"]  
}
```



Следует обратить внимание на то, что ключи в MongoDB являются регистрационно-зависимыми. Два ключа на приведенном рисунке различаются.

Примеры различающихся ключей



Ключи идентифицируют значения. Значения же могут различаться по типу данных. В приведенном ранее примере "CartoonName" соответствует строковому значению, "Year" – целочисленному, "FilmDirectorName" – массиву строковых значений. На рисунке приведены наиболее используемые ти-

Типы данных

- **String:** строковый тип данных (для строк используется кодировка UTF-8)
- **Integer:** используется для хранения целочисленных значений
- **ObjectID:** тип данных для хранения id документа
- **Array (массив):** тип данных для хранения массивов элементов
- **Boolean:** булевый тип данных, хранящий логические значения TRUE или FALSE
- **Regular expression:** применяется для хранения регулярных выражений

и др.



пы значений, среди которых вы можете видеть: строки, целые значения, логические, массивы и даже регулярные выражения.

Следует обратить внимание, что значения в документах обладают строгой типизацией, например, следующие два документа не будут идентичными. Если в первом случае для ключа "Duration" определена в качестве значения строка, то во втором случае значением является число. И именно поэтому значения этих ключей считаются различными.

Примеры документов с различающимися значениями

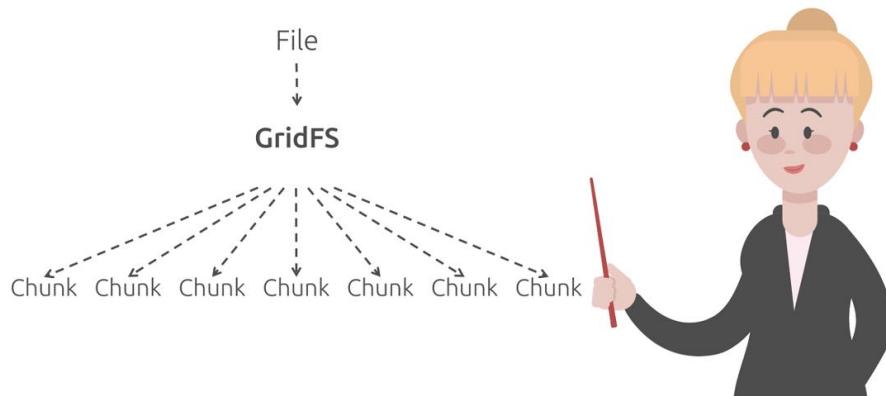
{"Duration": "88"}

{"Duration": 88}



Система MongoDB ориентирована на хранение документов. Однако эти документы иногда оказываются документами очень большого размера. MongoDB позволяет сохранять данные большого размера, однако при этом размер одного документа ограничивается 16 Мб. Но в MongoDB имеется специальное решение – технология GridFS, которая позволяет хранить данные по размеру больше, чем 16 Мб. Суть технологии GridFS заключается в использовании двух специальных коллекций. В первой коллекции, которая называется files, хранятся имена файлов, а также их метаданные, например размер. А в другой коллекции, которая называется chunks, в виде небольших сегментов хранятся данные файлов, обычно сегментами по 256 Кб.

Технология GridFS



2 Начало работы с MongoDB и управление базами

Для установки MongoDB можно загрузить один распространяемых пакетов с официального сайта MongoDB: <https://www.mongodb.com/download-center#community>. Официальный сайт предоставляет пакеты дистрибутивов для различных платформ: Windows, Linux, MacOS, Solaris. Для каждой платформы доступно несколько дистрибутивов разных версий. Кроме того, предлагаются два вида серверов – Community и Enterprise. Для учеб-

Два вида серверов



ных целей вполне достаточно версии Community. Но следует иметь в виду, что Enterprise-версия обладает несколькими большими возможностями.

Для выполнения заданий настоящего курса можно воспользоваться версией MongoDB, установленной на одном из серверов ИТМО: <https://online.ifmo.ru/mongo/>.

MongoDb в ИТМО

<https://online.ifmo.ru/mongo/>



При использовании этого сервера качестве `user-name` нужно указать `itmo`, а в качестве пароля – `online`. В качестве имени базы для всех примеров данной лекции будет использоваться база `globalStudent`.

Начальная аутентификация пользователей на сервере MongoDB в ИТМО

user-name: *itmo*
password: *online*
database name: *globalStudent*



На рисунке вы можете видеть пример аутентификации пользователя на сервере ИТМО. Обратите внимание, что пароль при вводе не отображается.

```
Mongo shell: Release 4.0.13 Production on Sat Nov 16 01:32:05 MSK 2019
Copyright (c) 2009, 2019, MongoDB Inc. All rights reserved.

Enter user-name: itmo
Enter password:
Enter database name: globalStudent

Now you will be connected to the command line interface

Connected to:
MongoDB v4.0.0 Community Edition - 64bit Production
> █
```

Система MongoDB может работать одновременно с несколькими базами. Однако существует понятие текущей базы, а в связи с этим есть набор команд MongoDB, который позволяет уточнить список доступных баз, установить текущую базу, просмотреть статистику по базе и уточнить содержимое текущей базы (т.е. список коллекций). На рисунке приведен список команд, предназначенный для управления базами MongoDB. Это и команды для демонстра-

Команды управления базами MongoDB

show dbs – демонстрирует список доступных баз

use <имя базы> – устанавливает текущую (активную) базы

db – отображает имя текущей базы

db.stats() – демонстрирует статистику текущей базы

show collections – отображает коллекции текущей базы

db.<имя коллекции>.insert(document) – добавляет новый документ в коллекцию



ции доступных баз, и команды для установки текущей базы, и демонстрация статистики по текущей базе, и отображение списка коллекций базы, и добавление новых документов в коллекцию. Зачем нужна текущая база? При работе с MongoDB, надо прежде всего установить требуемую базу данных в качестве текущей, чтобы затем ее использовать. Для этого надо либо с самого начала присоединиться к нужной базе, либо использовать для подключения команду `use`. При этом не важно, существует ли такая база данных или нет. Если ее нет, то MongoDB автоматически создаст ее при добавлении в нее данных.

Как увидеть список доступных баз данных? Для этого существует команда

```
show dbs
```

Приведенный рисунок демонстрирует пример просмотра списка доступных баз.



```
Now you will be connected to the command line interface
Connected to:
MongoDB v4.0.0 Community Edition - 64bit Production

> show dbs
dbuser1      0.000GB
globalStudent 0.001GB
>
```

Теперь можно подключиться к нужной базе (например, `globalStudent`) с помощью команды `use`. Приведенный рисунок демонстрирует такое переключение на базу `globalStudent` и соответствующее сообщение системы.



```
> use globalStudent
switched to db globalStudent
> █
```

Если в команде `use` используется имя, которого нет в списке баз, то будет создана новая база. В качестве имени можно задать любое имя, однако есть некоторые ограничения. Например, в имени не должно быть некоторых специальных символов. Они перечислены на рисунке. Кроме того, имена баз по длине ограничены 64 байтами. А еще есть зарезервированные имена (`local`, `admin` и `config`), которые также нельзя использовать.

Запрещенные символы и слова

/, \, ., ", *, <, >, :, |, ?, \$

local, admin, config



Команды `use` и `show dbs` тесно связаны между собой. Взгляните на пример на рисунке. Команда `use` пытается переключиться на новую базу, не находит ее и вынуждена создать новую базу с именем `newDB`. После этого используется команда `show dbs`. Очевидно, что мы ожидаем увидеть новую базу в списке баз, которые отображаются в результате выполнения команды. Однако это не так. Новая база появится в списке баз только после того, когда в ней появятся реальные данные. Для того, чтобы узнать какая база исполь-

```
switched to db globalStudent
> use newDB
switched to db newDB
> show dbs
dbuser1      0.000GB
globalStudent 0.001GB
> db
newDB
>
```

зуется в текущий момент времени, можно воспользоваться командой `db`. На рисунке приведен пример использования этой команды. Можно увидеть, что текущая база – `newDB`.

Следующий рисунок демонстрирует как с помощью команды

```
db.stats()
```

можно узнать разнообразную статистику о текущей базе (в частности ее размер). Для этого сначала с помощью команды `use` происходит переключение базы, а затем собирается статистика по базе. В частности, можно увидеть,

что имя базы – `globalStudent`, в базе 3 коллекции и 8588 объектов и т.п.

Для того, чтобы увидеть имена коллекций можно воспользоваться командой

```
show collections
```

Приведенный рисунок демонстрирует, как можно уточнить из каких коллекций состоит база. Команда была применена к демонстрационной базе `globalStudent`. Мы видим 3 коллекции: `STOPS`, `UNDREGROUND` и `student`. Следует обратить внимание, что имена коллекций являются регистрозависимыми, а, следовательно, имена `student` и `STUDENT` будут представлять разные коллекции.

```
 }  
> show collections  
STOPS  
UNDERGROUND  
student  
>
```

Как создать новую коллекцию в текущей базе? Казалось бы, что сначала надо бы создать коллекцию, а потом добавлять в нее новые документы.

Однако в MongoDB специально создавать коллекции можно, не обязательно. Новая коллекция будет создана автоматически при добавлении первого документа. Итак, напишем команду, которая добавит новый документ в коллекцию CARTOON в текущей базе.

```
db.CARTOON.insert(  
{  
    "CartoonName": "The Lion King",  
    "Year": 1994,  
    "Country": "USA",  
    "Duration": 88,  
    "FilmDirector": ["Roger Allers", "Rob Minkoff"]  
})
```

В команде надо указать имя коллекции и, собственно, сам добавляемый документ. В нашем примере в качестве документа используется документ в формате JSON, представляющий информацию о мультфильме “Король Лев”. Однако у этой команды есть особенность – она выполнима только в том случае, когда у пользователя достаточно прав для редактирования базы. Но если Вы работаете с базой globalStudent на сервере ИТМО, то у Вас таких прав явно недостаточно, так как эта **база открыта для студентов только на чтение**.

В следующих фрагментах данной лекции будут рассмотрены команды, позволяющие выбирать данные из существующих коллекций.

3 Выборка данных из коллекций

Выборка данных из коллекций в MongoDB осуществляется при помощи команды `find`. Действие этой команды, в конечном счете, во многом аналогично обычному запросу `select` в реляционной базе. Так же, как и в команде `select`, ключевые моменты запроса задаются с помощью условия выборки (т.е. фильтра) и проекции. И, тем не менее, синтаксически они отличаются. На рисунке приведен синтаксис команды `find`.

Команда для выборки данных из коллекции

```
db.<collection-name>.find([data_filter[, projection]])
```

data_filter – конструкция, которая задает условие для выборки данных из коллекции

projection – конструкция, которая задает поля коллекции для выборки



При использовании команды нужно явным образом указать имя коллекции, из которой осуществляется выборка. Обратите внимание на квадратные скобки в описании команды. Они указывают, что соответствующая часть конструкции может быть пропущена. Например, можно написать запрос, в котором не будет ни фильтра для выбираемых записей, ни проекции. В этом случае в результат запроса попадут все данные присутствующие в коллекции. Именно такой запрос мы сейчас и продемонстрируем на конкретном примере. Приведем пример простейшего варианта использования команды

Выборка всех данных из коллекции `student`

MongoDB `db.student.find()`

SQL `select * from student`



`find`. Например, чтобы извлечь все документы из коллекции `student`, можно использовать команду:

```
db.student.find()
```

В языке SQL этой команде бы соответствовал запрос:

```
select * from student
```

Фильтр и проекция в данном запросе не используется. На следующем рисунке видим результат выполнения команды в демонстрационной базе `globalStudent`.

```
> db.student.find()
{ "_id" : ObjectId("5b8418347e20d578c40b6f6d"), "name" : "Michael", "surname" : "Armani", "age" : 21 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f6e"), "name" : "Anna", "surname" : "Ford", "age" : 24 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f6f"), "name" : "Andrei", "surname" : "Ivanov", "age" : 25 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f70"), "name" : "Simon", "surname" : "Smith", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f71"), "name" : "Ksenia", "surname" : "Semenova", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f72"), "name" : "Jonathan", "surname" : "Wolff", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f73"), "name" : "Alisa", "surname" : "Braun", "age" : 23 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f74"), "name" : "Andrei", "surname" : "Smith", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f75"), "name" : "Simon", "surname" : "Armani", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f76"), "name" : "Ksenia", "surname" : "Smith", "age" : 18 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f77"), "name" : "Michael", "surname" : "Ivanov", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f78"), "name" : "Anna", "surname" : "Braun", "age" : 25 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f79"), "name" : "Simon", "surname" : "Smith", "age" : 21 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7a"), "name" : "Andrei", "surname" : "Ford", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7b"), "name" : "Jonathan", "surname" : "Braun", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7c"), "name" : "Ksenia", "surname" : "Smith", "age" : 18 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7d"), "name" : "Michael", "surname" : "Wolff", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7e"), "name" : "Andrei", "surname" : "Smith", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7f"), "name" : "Alisa", "surname" : "Semenova", "age" : 23 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f80"), "name" : "Simon", "surname" : "Ivanov", "age" : 20 }
Type "it" for more
`
```

Для вывода документов в более удобном наглядном представлении мы можем добавить вызов метода `pretty()`.

Форматирование результата выполнения команды

```
db.student.find().pretty()
```



И результат будет выведен совсем в ином стиле, который является более характерным для отображения документов формата JSON.

```
> db.student.find().pretty()
{
  "_id" : ObjectId("5b8418347e20d578c40b6f6d"),
  "name" : "Michael",
  "surname" : "Armani",
  "age" : 21
}
{
  "_id" : ObjectId("5b8418347e20d578c40b6f6e"),
  "name" : "Anna",
  "surname" : "Ford",
  "age" : 24
}
{
  "_id" : ObjectId("5b8418347e20d578c40b6f6f"),
  "name" : "Andrei",
  "surname" : "Ivanov",
  "age" : 25
}
{
  "_id" : ObjectId("5b8418347e20d578c40b6f70"),
  "name" : "Simon",
  "surname" : "Smith",
  "age" : 22
}
{
  "_id" : ObjectId("5b8418347e20d578c40b6f71"),
  "name" : "Ksenia",
  "surname" : "Semenova",
  "age" : 19
}
```

В приведенном примере мы рассмотрели выборку всех документов, однако, если нам надо получить не все документы, а только те, которые удовлетворяют определенному условию, то придется использовать в команде фильтр для отбора требуемых документов.

Например, выведем все документы (из коллекции `student`), для которых будет задан фильтр `{age:19}`. То есть то, что в реляционных базах выводилось бы запросом

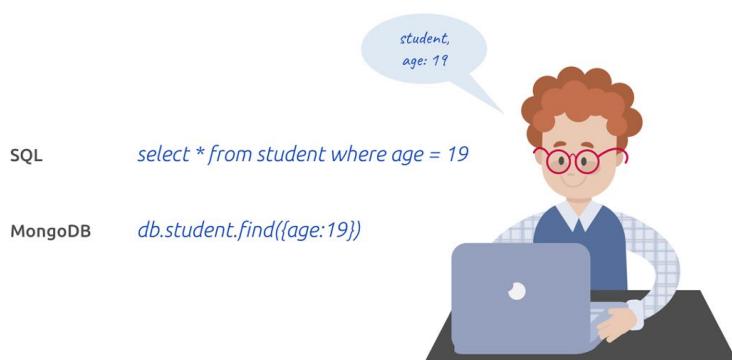
```
select * from student where age = 19
```

В MongoDB для этого придется написать следующий запрос:

```
db.student.find({age:19})
```

Проекция в запросе, по-прежнему, не используется.

Выборка из коллекции `student` 19-летних студентов



На рисунке вы можете видеть результат выполнения запроса и убедиться, что туда, действительно, попали только 19-летние студенты.

```
> db.student.find({age:19})
{ "_id" : ObjectId("5b8418347e20d578c40b6f71"), "name" : "Ksenia", "surname" : "Semenova", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7b"), "name" : "Jonathan", "surname" : "Braun", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f81"), "name" : "Jonathan", "surname" : "Smith", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f89"), "name" : "Simon", "surname" : "Armani", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f8e"), "name" : "Jonathan", "surname" : "Semenov", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f92"), "name" : "Simon", "surname" : "Smith", "age" : 19 }
>
```

Усложним запрос и выведем только **19-летних студентов** по имени **Simon**. В запросе придется задать более сложный фильтр для данных:

```
db.student.find({age:19, name: "Simon"})
```

В реляционной базе этому запросу соответствовал бы запрос вида:

```
select * from student where age = 19 and name = "Simon"
```

То есть запятая в фильтре интерпретируется как логическая операция **and**.

На рисунке вы можете видеть результат выполнения запроса и убедиться, что туда, действительно, попали только **19-летние** студенты по имени **Simon**.

```
> db.student.find({age:19, name: "Simon"})
{ "_id" : ObjectId("5b8418347e20d578c40b6f89"), "name" : "Simon", "surname" : "Armani", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f92"), "name" : "Simon", "surname" : "Smith", "age" : 19 }
>
```

Документ может иметь множество полей, но не все эти поля могут быть нужны в результате запроса. В этом случае в операторе выборки необходимо использовать проекцию, позволяющую указать требуемые в конечной выборке поля.

Выборка имен 19-летних студентов

MongoDB `db.student.find({age:19},{name:1})`

SQL `select _id, name from student where age=19`



Например, запрос

```
db.student.find({age:19},{name:1})
```

выводит **только имена 19-летних студентов**. Единица напротив названия поля в проекции указывает на то, что соответствующее поле должно присутствовать в выборке. При этом одно поле, а именно `_id` по умолчанию будет выведено даже без указания соответствующей единицы, указывающей на необходимость его присутствия.

```
> db.student.find({age:19},{name:1})
{ "_id" : ObjectId("5b8418347e20d578c40b6f71"), "name" : "Ksenia" }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7b"), "name" : "Jonathan" }
{ "_id" : ObjectId("5b8418347e20d578c40b6f81"), "name" : "Jonathan" }
{ "_id" : ObjectId("5b8418347e20d578c40b6f89"), "name" : "Simon" }
{ "_id" : ObjectId("5b8418347e20d578c40b6f8e"), "name" : "Jonathan" }
{ "_id" : ObjectId("5b8418347e20d578c40b6f92"), "name" : "Simon" }
>
```

Поэтому, если мы не хотим видеть поле `_id` в выборке, то надо явным образом упомянуть его в проекции и сопоставить ему значение 0:

```
db.student.find({age:19},{name:1,_id:0})
```

```
> db.student.find({age:19},{name:1,_id:0})
{ "name" : "Ksenia" }
{ "name" : "Jonathan" }
{ "name" : "Jonathan" }
{ "name" : "Simon" }
{ "name" : "Jonathan" }
{ "name" : "Simon" }
>
```

В качестве альтернативы для 0 и 1 в проекции можно использовать логические значения `true` и `false`:

```
db.student.find({age:19},{name:1,_id:false})
```

В некоторых случаях требуется вывести данные из базы в соответствии с каким-то шаблоном, то есть не по точному значению того или иного поля, а лишь какой-то его части. Для этого можно использовать, так называемые, регулярные выражения. Регулярные выражения позволяют задать шаблон, которому должно соответствовать значение строкового поля. В рамках данной лекции мы не будем приводить точное описание синтаксиса для задания регулярных выражений в MongoDB, но приведем несколько наиболее часто используемых шаблонов.

Популярные шаблоны

Шаблон	Пояснение	Примеры строк, соответствующих шаблону
/string/	задает строки, содержащие подстроку "string", с учетом используемого регистра	It is a string All of the strings
/string/i	задает строки, содержащие подстроку "string", без учета используемого регистра	It is a String All of the STRINGS
/string\$/	задает строки, которые заканчиваются на подстроку "string", с учетом используемого регистра	It is a string Long string
/string\$/i	задает строки, которые заканчиваются на подстроку "string", без учета используемого регистра	It is a String Long STRING

Вот, к примеру, запрос, который выведет всех студентов с именами, в которых содержится прописная буква S:

```
db.student.find({name:/S/})
```

А вот – результат этого запроса в среде MongoDB:

```
> db.student.find({name:/S/})
{ "_id" : ObjectId("5b8418347e20d578c40b6f70"), "name" : "Simon", "surname" : "Smith", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f75"), "name" : "Simon", "surname" : "Armani", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f79"), "name" : "Simon", "surname" : "Smith", "age" : 21 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f80"), "name" : "Simon", "surname" : "Ivanov", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f84"), "name" : "Simon", "surname" : "Wolff", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f89"), "name" : "Simon", "surname" : "Armani", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f92"), "name" : "Simon", "surname" : "Smith", "age" : 19 }
```

Приведем в качестве примера еще один запрос

```
db.student.find({name:/s/i})
```

который выводит всех студентов, в имени которых встречается буква s в нижнем или верхнем регистрах:

```
> db.student.find({name:/s/i})
{ "_id" : ObjectId("5b8418347e20d578c40b6f70"), "name" : "Simon", "surname" : "Smith", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f71"), "name" : "Ksenia", "surname" : "Semenova", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f73"), "name" : "Alisa", "surname" : "Braun", "age" : 23 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f75"), "name" : "Simon", "surname" : "Armani", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f76"), "name" : "Ksenia", "surname" : "Smith", "age" : 18 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f79"), "name" : "Simon", "surname" : "Smith", "age" : 21 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7c"), "name" : "Ksenia", "surname" : "Smith", "age" : 18 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7f"), "name" : "Alisa", "surname" : "Semenova", "age" : 23 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f80"), "name" : "Simon", "surname" : "Ivanov", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f83"), "name" : "Ksenia", "surname" : "Smith", "age" : 21 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f84"), "name" : "Simon", "surname" : "Wolff", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f89"), "name" : "Simon", "surname" : "Armani", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f8a"), "name" : "Alisa", "surname" : "Smith", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f8f"), "name" : "Ksenia", "surname" : "Ivanova", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f91"), "name" : "Alisa", "surname" : "Ivanova", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f92"), "name" : "Simon", "surname" : "Smith", "age" : 19 }
```

И еще один запрос

```
db.student.find({name:/se/i})
```

```
> db.student.find({name:/se/i})
{ "_id" : ObjectId("5b8418347e20d578c40b6f71"), "name" : "Ksenia", "surname" : "Semenova", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f76"), "name" : "Ksenia", "surname" : "Smith", "age" : 18 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7c"), "name" : "Ksenia", "surname" : "Smith", "age" : 18 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f83"), "name" : "Ksenia", "surname" : "Smith", "age" : 21 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f8f"), "name" : "Ksenia", "surname" : "Ivanova", "age" : 20 }
>
```

который выбирает всех студентов, в имени которых встречается подстрока **se** независимо от используемого регистра:

И, наконец, последний запрос в этом фрагменте

```
db.student.find({surname:/a$/})
```

позволяющий вывести всех студентов, фамилии которых заканчиваются на букву **a**:

```
> db.student.find({surname:/a$/})
{ "_id" : ObjectId("5b8418347e20d578c40b6f71"), "name" : "Ksenia", "surname" : "Semenova", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7f"), "name" : "Alisa", "surname" : "Semenova", "age" : 23 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f87"), "name" : "Michael", "surname" : "Semenova", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f8f"), "name" : "Ksenia", "surname" : "Ivanova", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f91"), "name" : "Alisa", "surname" : "Ivanova", "age" : 20 }
>
```

4 Фильтры, сортировка и агрегирование

В приведенных выше примерах мы использовали при выборке данных фильтры, в которых значение того или иного поля должно было в точности совпадать с заданным или соответствовать регулярному выражению. Между

Операции сравнения

- **\$gt** (больше чем)
- **\$lt** (меньше чем)
- **\$gte** (больше или равно)
- **\$lte** (меньше или равно)
- **\$ne** (не равно)
- **\$eq** (равно)



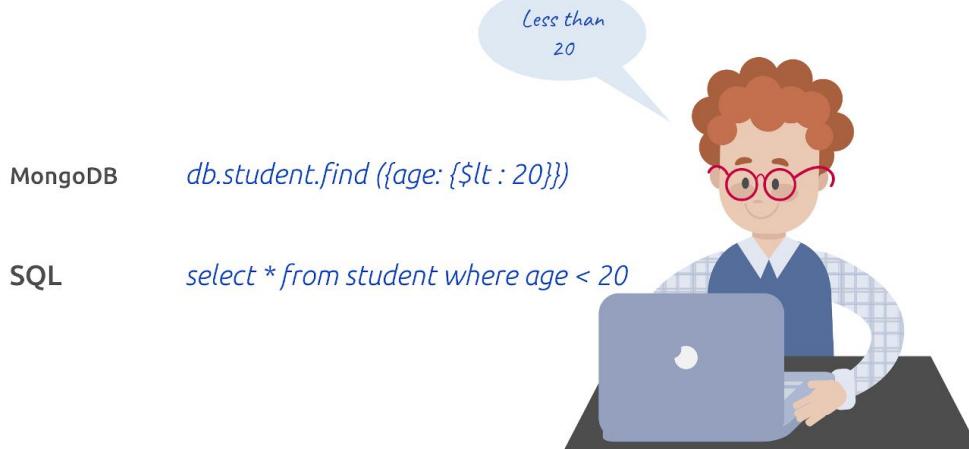
тем фильтры могут быть значительно более разнообразными и использовать разного рода операции. Например, операции сравнения, логические операции

и т.п. Рассмотрим некоторые, наиболее востребованные операции. Начнем с операций сравнения.

Все операции представляют собой аббревиатуру соответствующего английского названия операции. Например, `lt` – less than. Посмотрим, как используются операции сравнения в следующем запросе.

Сравним, как в SQL и в MongoDB записываются запросы, выбирающие всех студентов младше 20 лет. Для соответствующего фильтра понадобится операция less than `$lt`. Наверное, форма записи условия выборки в MongoDB

Выборка всех студентов младше 20 лет



выглядит не так изысканно, как в SQL, но, тем не менее, она достаточно понятна. На рисунке продемонстрирован результат выполнения команды:

```
> db.student.find({age:{$lt : 20}})
{ "_id" : ObjectId("5b8418347e20d578c40b6f71"), "name" : "Ksenia", "surname" : "Semenova", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f76"), "name" : "Ksenia", "surname" : "Smith", "age" : 18 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7b"), "name" : "Jonathan", "surname" : "Braun", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7c"), "name" : "Ksenia", "surname" : "Smith", "age" : 18 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f81"), "name" : "Jonathan", "surname" : "Smith", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f89"), "name" : "Simon", "surname" : "Armani", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f8d"), "name" : "Michaeln", "surname" : "Braun", "age" : 18 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f8e"), "name" : "Jonathan", "surname" : "Semenov", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f92"), "name" : "Simon", "surname" : "Smith", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f94"), "name" : "Anna", "surname" : "Ford", "age" : 18 }
>
```

Если нам нужно выполнить запрос, в котором требуется провести проверку на точное равенство, можно воспользоваться операцией equal – `$eq`.

Эквивалентные запросы

```
db.student.find ({age: {$eq : 20}})
```

```
db.student.find ({age: 20})
```



Но можно написать запрос в манере, которая описывалась в предыдущем фрагменте лекции и не использует явного упоминания операции сравнения, а подразумевает его неявно. На рисунке приведен результат выполнения запроса, записанного в первой манере:

```
> db.student.find({age:{$eq : 20}})
{ "_id" : ObjectId("5b8418347e20d578c40b6f72"), "name" : "Jonathan", "surname" : "Wolff", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f75"), "name" : "Simon", "surname" : "Armani", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f77"), "name" : "Michael", "surname" : "Ivanov", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7a"), "name" : "Andrei", "surname" : "Ford", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7d"), "name" : "Michael", "surname" : "Wolff", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f80"), "name" : "Simon", "surname" : "Ivanov", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f82"), "name" : "Andrei", "surname" : "Braun", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f85"), "name" : "Andrei", "surname" : "Smith", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f8a"), "name" : "Alisa", "surname" : "Smith", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f8f"), "name" : "Ksenia", "surname" : "Ivanova", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f91"), "name" : "Alisa", "surname" : "Ivanova", "age" : 20 }
```

Операторы сравнения можно использовать и в более сложных выражениях, где требуется задать несколько ограничений на одно или несколько полей. Например, нужно вывести студентов, возраст которых больше 20, но меньше 23 лет. Сравним, как такие запросы записываются в SQL и в MongoDB.

Выборка всех студентов старше 20, но меньше 23 лет

MongoDB `db.student.find ({age: {$gt : 20, $lt : 23}})`

SQL `select * from student where age > 20 and age < 23`



В SQL условия выборки `age > 20` и `age < 23` соединены логической операцией `and`. В варианте с MongoDB эта операция явным образом не записана,

но запятая, соединяющая два условия, неявно эту операцию подразумевает. Однако в более сложных условиях логические операции можно и нужно записывать явно. На рисунке приведен результат исполнения команды в среде MongoDB:

```
> db.student.find({age:{$gt: 20, $lt: 23}})
{ "_id" : ObjectId("5b8418347e20d578c40b6f6d"), "name" : "Michael", "surname" : "Armani", "age" : 21 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f70"), "name" : "Simon", "surname" : "Smith", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f74"), "name" : "Andrei", "surname" : "Smith", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f79"), "name" : "Simon", "surname" : "Smith", "age" : 21 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7e"), "name" : "Andrei", "surname" : "Smith", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f83"), "name" : "Ksenia", "surname" : "Smith", "age" : 21 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f84"), "name" : "Simon", "surname" : "Wolff", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f86"), "name" : "Anna", "surname" : "Ford", "age" : 21 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f87"), "name" : "Michael", "surname" : "Semenova", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f8c"), "name" : "Anna", "surname" : "Ford", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f90"), "name" : "Michael", "surname" : "Wolff", "age" : 21 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f93"), "name" : "Andrei", "surname" : "Armani", "age" : 22 }
>
```

В MongoDB так же, как и в SQL, есть возможность явно использовать стандартные логические операции (то есть операции, объединяющие логические значения) для того, чтобы формировать разнообразные условия выборки. Список доступных логических операций приведен на рисунке. Однако

Логические операции

- *\$and* (и)
- *\$or* (или)
- *\$not* (отрицание)



надо признать, что их использование в MongoDB выглядит менее естественно, чем в SQL.

Сравните, как записываются соответствующие запросы, выбирающие студентов по имени **Simon** или **Anna**, в SQL

```
select * from student where name = "Simon" or name = "Anna"
```

и MongoDB

```
db.student.find({$or: [{name:"Simon"}, {name:"Anna"}]})
```

На рисунке приведен результат выполнения такой команды:

```
> db.student.find({$or:[{name:"Simon"}, {name:"Anna"}]})  
[{"_id": ObjectId("5b8418347e20d578c40b6f6e"), "name": "Anna", "surname": "Ford", "age": 24}, {"_id": ObjectId("5b8418347e20d578c40b6f70"), "name": "Simon", "surname": "Smith", "age": 22}, {"_id": ObjectId("5b8418347e20d578c40b6f75"), "name": "Simon", "surname": "Armani", "age": 20}, {"_id": ObjectId("5b8418347e20d578c40b6f78"), "name": "Anna", "surname": "Braun", "age": 25}, {"_id": ObjectId("5b8418347e20d578c40b6f79"), "name": "Simon", "surname": "Smith", "age": 21}, {"_id": ObjectId("5b8418347e20d578c40b6f80"), "name": "Simon", "surname": "Ivanov", "age": 20}, {"_id": ObjectId("5b8418347e20d578c40b6f84"), "name": "Simon", "surname": "Wolff", "age": 22}, {"_id": ObjectId("5b8418347e20d578c40b6f86"), "name": "Anna", "surname": "Ford", "age": 21}, {"_id": ObjectId("5b8418347e20d578c40b6f89"), "name": "Simon", "surname": "Armani", "age": 19}, {"_id": ObjectId("5b8418347e20d578c40b6f8c"), "name": "Anna", "surname": "Ford", "age": 22}, {"_id": ObjectId("5b8418347e20d578c40b6f92"), "name": "Simon", "surname": "Smith", "age": 19}, {"_id": ObjectId("5b8418347e20d578c40b6f94"), "name": "Anna", "surname": "Ford", "age": 18}]>
```

Далее приведем в качестве примера два запроса, выбирающих студентов по имени **Simon** старше 20 лет. В SQL:

```
select * from student where name = "Simon" and age > 20
```

В MongoDB – стандартный оператор **find**, но с достаточно сложным фильтром:

```
db.student.find({$and:[{name:"Simon"}, {age:{$gt:20}}]})
```

Как видно из приведенных примеров – запрос в SQL формулируется значительно проще, чем в MongoDB. На рисунке приведен результат исполнения команды в среде MongoDB:

```
> db.student.find({$and:[{name:"Simon"}, {age:{$gt:20}}]})  
[{"_id": ObjectId("5b8418347e20d578c40b6f70"), "name": "Simon", "surname": "Smith", "age": 22}, {"_id": ObjectId("5b8418347e20d578c40b6f79"), "name": "Simon", "surname": "Smith", "age": 21}, {"_id": ObjectId("5b8418347e20d578c40b6f84"), "name": "Simon", "surname": "Wolff", "age": 22}]>
```

MongoDB не только позволяет задавать фильтр и проекцию при выборке данных, но и указывать дополнительные опции, позволяющие сортировать и факторизовать результат выборки. Рассмотрим эти возможности подроб-

Сортировка результатов запросов

```
db.<collection-name>.find([data_filter[, projection]]).sort({options})
```



нее. MongoDB предоставляет возможность отсортировать выборку с помощью функции **sort**. Передавая в эту функцию в качестве параметра ключ

для сортировки и значение 1 или -1, можем указать, в каком порядке следует производить сортировку по требуемому ключу: по возрастанию (1) или по убыванию (-1).

Сортировка списка студентов по имени

MongoDB `db.student.find().sort({name: 1})`

SQL `select * from student order by name`



Сравним, как записывается в SQL и MongoDB сортировка списка студентов по возрастанию имени. В SQL для указания сортировки используется конструкция `order by` (по умолчанию по возрастанию), а в MongoDB – функция `sort` с параметрами: `name` и `1`. На следующем рисунке приведен пример записи сортировка того же списка по убыванию поля `name`. В SQL для указания сортировки используется конструкция `order by` с опцией `desc`, а в MongoDB – функция `sort` с параметрами: `name` и `-1`.

Сортировка списка студентов по имени (по убыванию)

MongoDB `db.student.find().sort({name: -1})`

SQL `select * from student order by name desc`



На рисунке приведен результат исполнения последнего запроса:

```
> db.student.find().sort({name:-1})
{
  "_id": ObjectId("5b8418347e20d578c40b6f70"), "name": "Simon", "surname": "Smith", "age": 22 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f75"), "name": "Simon", "surname": "Armani", "age": 20 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f79"), "name": "Simon", "surname": "Smith", "age": 21 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f80"), "name": "Simon", "surname": "Ivanov", "age": 20 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f84"), "name": "Simon", "surname": "Wolff", "age": 22 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f89"), "name": "Simon", "surname": "Armani", "age": 19 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f92"), "name": "Simon", "surname": "Smith", "age": 19 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f8d"), "name": "Michaeln", "surname": "Braun", "age": 18 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f6d"), "name": "Michael", "surname": "Armani", "age": 21 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f77"), "name": "Michael", "surname": "Ivanov", "age": 20 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f7d"), "name": "Michael", "surname": "Wolff", "age": 20 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f87"), "name": "Michael", "surname": "Semenova", "age": 22 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f90"), "name": "Michael", "surname": "Wolff", "age": 21 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f71"), "name": "Ksenia", "surname": "Semenova", "age": 19 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f76"), "name": "Ksenia", "surname": "Smith", "age": 18 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f7c"), "name": "Ksenia", "surname": "Smith", "age": 18 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f83"), "name": "Ksenia", "surname": "Smith", "age": 21 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f8f"), "name": "Ksenia", "surname": "Ivanova", "age": 20 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f72"), "name": "Jonathan", "surname": "Wolff", "age": 20 }
{
  "_id": ObjectId("5b8418347e20d578c40b6f7b"), "name": "Jonathan", "surname": "Braun", "age": 19 }
}
Type "it" for more
```

Иногда нам не нужны все результаты запроса, а достаточно получить лишь первые несколько документов. Для этого может использоваться функция `limit`. Ее параметр задает требуемое количество документов для вывода. Например, вывести 3 документа из коллекции `student` можно так, как показано на рисунке. Но особый смысл в использовании этой функции возникает

Выборка 3 документов из коллекции `student`

`db.student.find().limit(3)`



тогда, когда мы уже имеем дело с отсортированным списком.

Например, нужно вывести первых десять студентов из отсортированного по фамилии в лексикографическом порядке списка студентов. Такой запрос может быть записан так:

```
db.student.find().sort({surname:1}).limit(10)
```

На следующем рисунке представлен результат исполнения такого запроса в среде MongoDB:

```
>
> db.student.find().sort({surname:1}).limit(10)
{ "_id" : ObjectId("5b8418347e20d578c40b6f6d"), "name" : "Michael", "surname" : "Armani", "age" : 21 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f75"), "name" : "Simon", "surname" : "Armani", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f89"), "name" : "Simon", "surname" : "Armani", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f93"), "name" : "Andrei", "surname" : "Armani", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f73"), "name" : "Alisa", "surname" : "Braun", "age" : 23 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f78"), "name" : "Anna", "surname" : "Braun", "age" : 25 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f7b"), "name" : "Jonathan", "surname" : "Braun", "age" : 19 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f82"), "name" : "Andrei", "surname" : "Braun", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f8d"), "name" : "Michaeln", "surname" : "Braun", "age" : 18 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f6e"), "name" : "Anna", "surname" : "Ford", "age" : 24 }
>
```

Еще одну любопытную возможность предоставляет функция `skip`. Она позволяет пропустить в результате запроса заданное количество документов. Типичный сценарий ее использования такой: отсортировать список, пропустить в отсортированном списке требуемое количество документов, выбрать заданное количество документов.

Следующий рисунок демонстрирует результат запроса

```
db.student.find().sort({surname:1}).skip(10).limit(5)
```

в котором сортируется список студентов по фамилии, затем пропускаются первые 10 документов и выводятся следующие 5:

```
> db.student.find().sort({surname:1}).skip(10).limit(5)
{ "_id" : ObjectId("5b8418347e20d578c40b6f7a"), "name" : "Andrei", "surname" : "Ford", "age" : 20 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f86"), "name" : "Anna", "surname" : "Ford", "age" : 21 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f8c"), "name" : "Anna", "surname" : "Ford", "age" : 22 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f94"), "name" : "Anna", "surname" : "Ford", "age" : 18 }
{ "_id" : ObjectId("5b8418347e20d578c40b6f6f"), "name" : "Andrei", "surname" : "Ivanov", "age" : 25 }
>
```

В SQL можно подсчитать количество строк с помощью агрегатной функции `count`. Например, так:

```
select count(*) from student
```

Аналогичная возможность есть и в MongoDB. С помощью похожей функции `count()` можно получить число элементов в коллекции. Например, количе-

Агрегирование данных в коллекции/таблице

MongoDB	<code>db.student.count()</code>
SQL	<code>select count(*) from student</code>



ство документов в коллекции `student` можно подсчитать так:

```
db.student.count()
```

Так же, как и в SQL, есть возможность комбинировать условия поиска и функцию count. Например, для того, чтобы подсчитать, сколько есть студентов с именем Anna, в SQL мы напишем запрос

```
select count(*) from student where name = "Anna"
```

В MongoDB:

```
db.student.find({name: "Anna"}).count()
```

На рисунке приведен результат исполнения такого запроса среде MongoDB:

```
> db.student.find({name: "Anna"}).count()
5
>
```

В коллекции могут быть документы, которые содержат одинаковые значения для одного или нескольких полей. Например, в нескольких документах определено `name = "Kat"`. А по условиям запроса надо найти только уникальные различающиеся значения для одного из полей документа. Такую опера-

Использование конструкции `distinct` в SQL и MongoDB

MongoDB	<code>db.student.distinct("name")</code>
SQL	<code>select distinct name from student</code>



цию в математике называют операцией факторизации. Для этого в системах, управляющих хранением данных, принято использовать конструкцию `distinct`. Напомним, что такая возможность существует и в SQL и используется, например, так:

```
select distinct name from student
```

Аналогичная возможность есть и в MongoDB. Запрос

```
db.student.distinct("name")
```

отображает список имен, которые встречаются в коллекции student. Причем каждое имя будет отображаться в результате запроса только один раз.

На рисунке приведен пример исполнения такого запроса. Как видите – повторяющихся имен нет.

```
> db.student.distinct("name")
[
    "Alisa",
    "Andrei",
    "Anna",
    "Jonathan",
    "Ksenia",
    "Michael",
    "Michaeln",
    "Simon"
]
>
```

5 Редактирование данных и индексы

В предыдущих фрагментах данной лекции мы рассмотрели приемы, позволяющие осуществлять добавление данных и их просмотр. Но, разумеется, в MongoDB есть приемы, позволяющие редактировать уже существующие коллекции. Начнем с редактирования отдельного поля. Для выполнения такой операции есть команда `set`. У этой команды есть абсолютный аналог в SQL. Пример записи команды `set` и ее аналог в SQL приведены на рисунке. Обратите внимание на три части команды. Первая – выбирает множе-

Обновление отдельного поля

MongoDB `db.STUDENT.update({name : "Eugene", age: 29}, {$set: {age : 30}}, {multi:true})`

SQL

`UPDATE student SET age = 30 WHERE name = "Eugene" and age = 29`



ство документов, которые будут отредактированы, вторая – указывает какие именно поля и каким образом будут изменены, а третья часть команды (`{multi:true}`) указывает на то, что отредактированы будут все документы, для которых будет выполнено условие выборки. Если третьего фрагмента не

будет – будет отредактирован только первый попавшийся документ из полученной выборки.

Для удаления отдельного поля используется оператор `unset`. Пример использования приведен на рисунке. Первый параметр команды указывает фильтр для выборки, второй – имя поля, а третий – указывает на то, что должны быть удалены соответствующие поля во всех документах выборки. Если вдруг подобного ключа в документах выборки не существует, то оператор не оказывает никакого влияния коллекцию.

Удаление отдельного поля

MongoDB `db.STUDENT.update({ "name": "Tom"}, {$unset: { "left": 1}}, {multi:true})`

SQL

`ALTER TABLE student DROP COLUMN left`



С помощью одной команды `unset` можно удалить сразу несколько полей. На рисунке приведен пример, в котором в указанной выборке удаляются поля `salary` и `left`. Для этого во втором параметре выборки их просто надо перечислить через запятую.

Удаление нескольких полей

MongoDB `db.STUDENT.update({ "name": "Tom"}, {$unset: { "salary": 1, "left": 1}}, {multi:true})`

Можно удалять не только отдельные поля, но и целые коллекции. Для этого, как и в SQL используется команда `DROP`. Только синтаксис у этой команды – немного другой. Пример использования этой команды и сравнение с аналогичной командой SQL можете видеть на рисунке. В MongoDB:

`db.STUDENT.drop()`

в SQL:

`DROP TABLE student`

Как известно, индексы используются в базах данных для ускорения выборки. Кроме того, индекс можно использовать для контроля уникальности значений полей. В этом смысле MongoDB не является исключением. Создавать индексы по полям коллекций можно и нужно. Для этого существует команда `createIndex`. Она работает аналогично тому, как работает команда `CREATE INDEX` в SQL. На рисунке приведен пример создания индекса по полю `name` в MongoDB и аналогичный оператор SQL. Индекс, который создается

Создание индекса для коллекции STUDENT

MongoDB `db.STUDENT.createIndex({ "name" : 1 })`

SQL `CREATE INDEX idx_name ON student(name)`



таким образом, не является уникальным, т.е. он будет способствовать более быстрому выполнению запросов по полю `name`, но при этом не будет контролировать уникальность значения поля `name`.

```
> db.STUDENT.createIndex({"name" : 1})
{
    "createdCollectionAutomatically" : true,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

Для того, чтобы контролировать уникальность значения поля/полей нужно указать это свойство при создании индекса явным образом. На рисунке приведены примеры создания индексов по полям с уникальными значениями в MongoDB и SQL. Несмотря на внешнюю похожесть создания уникальных индексов, в MongoDB все же есть свои особенности. Они связаны с тем, что документы в MongoDB слабо структурированы и не обязаны иметь в составе любого документа поле `name`. В случае наличия такого индекса, для добавляемого документа будет автоматически создаваться поле `name` (даже если оно не было предусмотрено в документе), а в качестве значения с ним будет ассоциировано неопределенное значение `null`. Первый документ с неопределенным значением будет создан без всяких проблем, а вот со вторым –

проблемы будут. Индекс откажется принимать такое значение, так документ со значением `null` уже присутствует в коллекции.

Создание уникального индекса для коллекции STUDENT

MongoDB `db.STUDENT.createIndex({ "name": 1}, { "unique": true})`

SQL `CREATE UNIQUE INDEX idx_name ON student(name)`



Индексы могут быть построены не только по одному полю. Можно строить составные индексы на основе нескольких полей. Пример на рисунке демонстрирует построение аналогичных составных индексов по полям `name` и `age` в MongoDB и SQL. Составные индексы также могут быть уникальными,

Создание составного индекса для коллекции STUDENT

MongoDB `db.STUDENT.createIndex({ "name": 1, "age": 1})`

SQL `CREATE INDEX idx_name ON student(name,age)`



если добавить к их определению соответствующий параметр `unique`.

```
> db.STUDENT.createIndex({ "name" : 1, "age" : 1})
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 2,
    "numIndexesAfter" : 3,
    "ok" : 1
}
```

Все индексы базы данных хранятся в системной коллекции `system.indexes`. Обратившись к ней (при наличии подходящих прав для работы с базой), мы можем получить все индексы и связанную с ними информацию:

```
db.system.indexes.find()
```

Конечно, в SQL-подобных базах тоже есть свои системные таблицы, в которых такие сведения тоже хранятся, но имена соответствующих таблиц сильно различаются в зависимости от СУБД, в контексте которой используется язык SQL. Поэтому мы не можем привести аналогичную команду SQL.

Кроме того, существует возможность просмотра индексов для конкретной коллекции. Для этого есть команда `getIndexes()`:

```
db.STUDENT.getIndexes()
```

На рисунке Вы можете видеть саму команду и результат ее работы. Результат

```
> db.STUDENT.getIndexes()
[ {
    {
        "v" : 2,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "dbuser1.STUDENT"
    },
    {
        "v" : 2,
        "key" : {
            "name" : 1
        },
        "name" : "name_1",
        "ns" : "dbuser1.STUDENT"
    },
    {
        "v" : 2,
        "key" : {
            "name" : 1,
            "age" : 1
        },
        "name" : "name_1_age_1",
        "ns" : "dbuser1.STUDENT"
    }
]
```

интересен тем, что в нем можно увидеть какие именно имена сопоставила система тем или иным индексам (имена для каждого индекса создавались автоматически и в результате запроса им соответствует ключ `name`). В нашем случае мы видим три индекса: `_id_`, `name_1` и `name_1_age_1`. Эти имена еще могут нам понадобиться.

Если индексов в базе становится слишком много или они начинают мешать добавлению новых значений, которые оказываются не уникальными, можно задуматься об операции удаления лишних индексов. Разумеется, такая команда тоже присутствует и имеет подходящий аналог в SQL. На рисунке Вы можете видеть примеры таких команд в MongoDB и SQL. В любом случае – и в MongoDB и в SQL нужно при удалении явно указать имя индекса (даже если он создавался автоматически).

Удаление индексов базы

MongoDB `db.STUDENT.dropIndex("name_1")`

SQL `DROP INDEX name_1 ON student`



Мы рассмотрели в рамках нашей лекции лишь основные приемы для работы с MongoDB. Более глубокое изучение MongoDB не входит в программу нашего курса, а любознательным слушателям можно посоветовать читать официальную документацию MongoDB. Ссылку на нее вы можете видеть на рисунке.

Официальная документация MongoDB

<https://docs.mongodb.com/guides/>

