Технологии работы с большими данными

Лекция 4. NoSQL & MongoDB

нетология

Алексей Кузьмин

Директор разработки; ДомКлик.ру

О спикере:

- Руковожу направлением работы с данными и Data Science
- Преподаю в Нетологии
- Работаю в IT с 2010 года (ABBYY, ДомКлик)
- Окончил МехМат МГУ в 2012 году

Я в Слаке:

@Alexey Kuzmin

Работа с данными

Источники данных

Сбор данных
Лекция 8

SQL-БД
Лекция 1

NoSQL
Лекция 4

MapReduce
Лекция 5-7

Мотивация Big Data
Лекция 3

Data Science
Лекция 2

Отчеты
Лекция 1

Модели
Лекция 2

Люди и процессы
Лекция 9

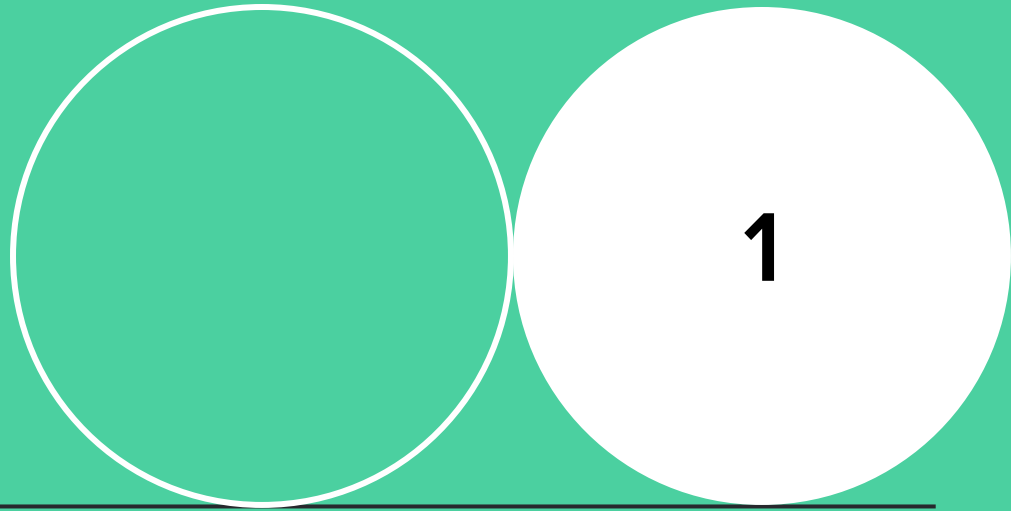
Примеры кейсов
Лекция 10

Что такое базы данных NoSQL?

- Поговорим о NoSQL решениях и базах данных
- Подробно рассмотрим MongoDB

NOSQL

Not Only SQL



Алексей Кузьмин

Технологии работы с большими данными

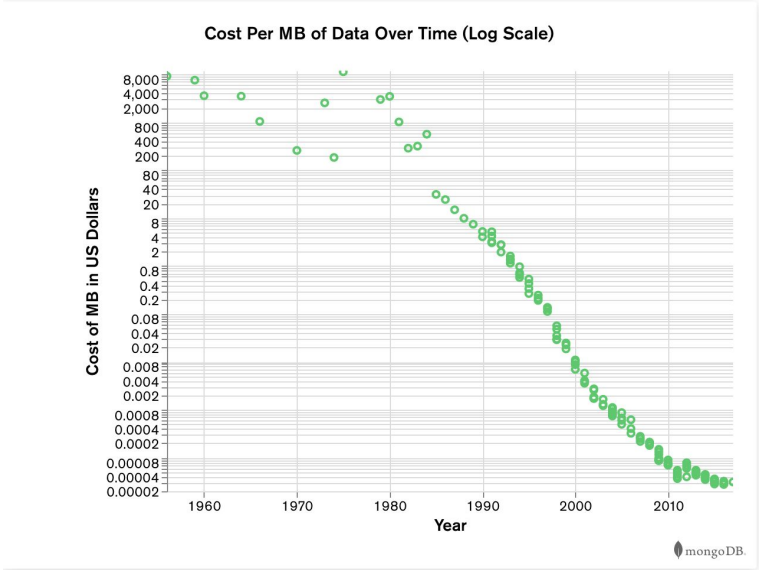
нетология

SQL-базы

- Хранение реляционных (табличных, структурированных) данных
- Нормализованная структура данных (отсутствие дублирования информации)
- Детальное планирование и подготовка при разработке схемы данных

NoSQL

- Not only SQL
- Другой формат хранения данных
- Появились в конце 2000’х с падением стоимости мегабайта хранения



NoSQL

- Оптимизированы не на экономию места, а на продуктивность использования
- Зачастую поддерживают хранение слабоструктурированной информации
- Гибкая схема данных

Виды NoSQL

- Документоориентированные - каждая запись (документ) состоит из множества пар “ключ-значение”. Часто выступают в роли production-баз данных. Иногда - как единое хранилище. Пример - MongoDB.
- Ключ-значение - более простая баз данных. Каждая запись - абстрактные данные, записанные с каким-то ключом. Часто выступают в роли хранения справочной информации. Пример - Redis
- Колоночные - похожи на реляционные. Так же хранят данные в таблицах, однако запись не обязана иметь все доступные колонки. Часто на их основе строят DataLake. Пример - HBase и Cassandra
- Графовые - хранят данные в виде графа связей. Пример - Neo4j.

CAP-теорема

- согласованность данных (англ. consistency) — во всех вычислительных узлах в один момент времени данные не противоречат друг другу;
- доступность (англ. availability) — любой запрос к распределённой системе завершается корректным откликом, однако без гарантии, что ответы всех узлов системы совпадают;
- устойчивость к разделению (англ. partition tolerance) — расщепление распределённой системы на несколько изолированных секций не приводит к некорректности отклика от каждой из секций.
- Из 3-х свойств обеспечить выполнение можно любых двух

3 класса систем

- **CA** - во всех узлах данные согласованы и обеспечена доступность, при этом она жертвует устойчивостью к распаду на секции.
- **CP** - в каждый момент обеспечивает целостный результат и способна функционировать в условиях распада, но достигает этого в ущерб доступности: может не выдавать отклик на запрос.
- **AP** не гарантируется целостность, но при этом выполнены условия доступности и устойчивости к распаду на секции.

SQL vs NoSQL

	SQL	**NoSQL**
CAP	AC	AP, CP, реже AC
Модель хранения	Таблицы с фиксированной структурой	Document: JSON или XML Key-value: ключ-значение, Wide-column: таблицы с динамическими столбцами Graph: узлы и ребра
Появление	1970-ые	2000+
Назначение	Общего назначения	Document: общего назначения Key-value: хранение большого количества данных с быстрым поиском по ключу Wide-column: большое количество данных с разными запросами Graph: анализ связей

SQL vs NoSQL

	SQL	**NoSQL**
Схема данных	Жесткая	Гибкая
Масштабирование	Вертикальное	Горизонтальное
Атомарные операции с несколькими сущностями (транзакции)	Поддерживаются	Редко
Join	Поддерживаются	Редко, обычно просто не нужны

SQL vs NoSQL

Плюсы NoSQL

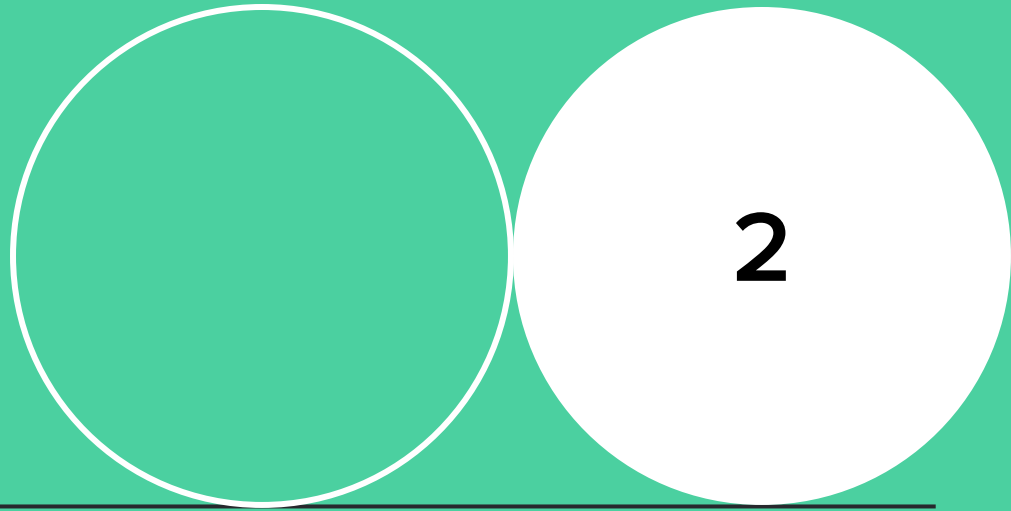
- Гибкая структура данных
- Горизонтальное масштабирование
- Быстрые запросы
- Проще для разработчика

Плюсы SQL

- Поддержка транзакций
- Экономнее расход диска
- Часто привычнее и проще в использовании и сопровождении

MongoDB

Документы и JSON



Алексей Кузьмин

Технологии работы с большими данными

нетология

MongoDB

MongoDB — это мощная, гибкая и масштабируемая база данных общего назначения. Mongo сочетает в себе вторичные индексы, запросы с диапазонами и сортировкой, агрегацией и геопозиционные запросы.

Преимущества

- Поддержка индексов
- Реплицирование и горизонтальное масштабирование
- Поддержка mapReduce (хотя и тормозит)
- Поддержка JavaScript в запросах
- В 2018 году (версия 4.0) добавлена поддержка транзакций
- Отсутствие схемы

JSON

key-value структура

ограничен { }

ключи не могут содержать символ точки и знак $

Типы данных

- null { x: null }
- boolean - true или false { x: true, y: false }
- Числа { x: 3.14 }
- Строки {x: 'string' }
- Дата - 64-битные целые числа, которые показывают количество миллисекунд прошедших с эпохи линукс(1 января, 1970 года). Для работы mongoDB используют класс Date в JS. { x: new Date() }
- Массивы { x: ['string', 3.14, new Date()] }
- Вложенные документы { x: { name: 'Merrick', isAdmin: true } }
- ObjectID - идентификатор объекта { x: ObjectId() }

Документ

Документ - это просто набор key/value значений

case-sensitive и type-sensitive**:**

{ count: 5 }

{ count: '5' }

{ Count: 5 }

{ Count: '5' }

Ключи - произвольный набор символов.

Документ не может содержать поля с одинаковыми ключами

{ greeting: "Hello, world!", greeting: "Hello, MongoDB!" } <- не валидный документ

Коллекция

Коллекция - это группа документов.

Коллекции могут содержать произвольные документы, они не обязаны обладать единой структурой

users:

{ name: 'Merrick', views: 5 }

{ name: 'John', views: 15 }

{ weather: 'rain', walk: false }

Коллекции

Одна коллекция для всего - плохо:

1. Не унифицированная работа с документами
2. Быстрее получить список коллекций, чем извлечь список из коллекции и только после отфильтровать.
3. Объединение документов одного и того же типа в коллекцию позволяет оптимизировать работу с данными.
4. Индексы требуют наличия некоторой структуры

База данных

Коллекции хранятся в базах данных

Mongo Shell

Mongo shell это инструмент для управления базой с помощью консоли, который имеет доступ к API базы + интерпретатор JS.

Основные команды:

- db — выведет имя текущей выбранной базы, если база еще не выбрана то покажет test
- use <database name> — с помощью данной команды мы можем выбрать любую базу данных, если база не существует, то она просто создастся. После чего команда db указывает на выбранную базу.
- help — выводит вспомогательную информацию о дополнительных методах.
- show dbs — показывает все существующие базы данных
- show collections — показывает существующие коллекции в выбраной базе данных.

Создание документов

Вставить документ - при помощи insertOne

db.users.insertOne({ name: 'Alexey', email: 'test@gmail.com'})

добавляет к документу поле _id если оно отсутствует и добавляет документ в коллекцию.

Возвращает:

{

"acknowledged" : true,

"insertedId" : ObjectId("5a06d38ed438c69bc223b7b2")

}

Создание документов

Вставить несколько документов - при помощи insertMany

```
db.users.insertMany([
 { name: 'Vasya', email: 'vasya@gmail.com' },
 { name: 'Petya', email: 'petya@gmail.com' }
])
```

добавляет к документу поле _id если оно отсутствует и добавляет документ в коллекцию. Возвращает:

```
{
"acknowledged" : true,
"insertedIds" : [  ObjectId("5a06d473d4sdfsc67bc223b7b3"),  ObjectId("5a06d473d4sfdsdf7bc223b7b4") ]
}
```

Чтение данных

find или findOne

db.users.find() -> все документы из коллекции

Можно передать объект filter.

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

filter - объект, по которому производится сравнение и фильтрация документов. Можно несколько полей

db.users.find({ nick: 'John', age: 27 })

Чтение данных

Выборка полей

db.users.find({}, { nick: true, age: true }) -> в результате будут только поля с ником и возрастом

Можем исключить какие то поля, вместо того, чтобы указывать, что возвращать:

db.users.find({}, { age: false }) -> все кроме возраста

Одновременно использовать true и false, для разных полей нельзя.

Условные операторы

```
db.users.find({ age : { $gte : 18, $lte : 30 }})
```

- $lt — оператор меньше(<)
- $lte — оператор меньше либо равно(≤)
- $gt — оператор больше (>)
- $gte — оператор больше либо равно (≥)
- $ne — Оператор отрицания не равно(!=)

$in

Проверка на множество значений

```
db.users.find({ age: { $in: [27, 28, 29] } })
```

$or

Одно из условий

```
db.users.find({ $or: [{ age: 27}, { nick: 'John' }] })
```

Можно усложнять :-)

```
db.users.find({ $or: [
 { age: { $in: [27, 28, 29] } },
 { nick: 'John' }
] })
```

усложняя запрос, мы усложняем поиск и задействуем больше ресурсов.

Поиск поддокумента

```
{
 "name" : {
   "first" : "Joe"
 },
 "age" : 45
}
```

Используем точку для указания “вложенности”:

```
db.users.find({ 'name.first': 'Joe' })
```

Ключ не может содержать символ точки, mongoDB выдаст ошибку если вы попытаетесь записать свойство с точкой.

Удаление документов

deleteOne, deleteMany, drop

Первым аргументом принимают объект filter. В нем указывается поле, по которому фильтруются все документы в коллекции.

db.movies.deleteOne({ _id: 4 })

db.movies.deleteMany({ year: 1984 })

db.movies.drop() <- удаляет коллекцию

Замещение документов

replaceOne

Метод replaceOne полностью заменяет документ на переданный вторым аргументом.

db.movies.replaceOne({ title: “Some title” }, new_movie)

Первый параметр - фильтр, второй - новый json-документ

Обновление документов

updateOne и updateMany

Методы updateOne и updateMany позволяют обновить документ без его полного считывания, но для этого нужно использовать mongoDB operators(Операторы)

Операторы

Сайт, который учитывает количество посещений

```
{
 "_id" : ObjectId("4b253b067525f35f94b60a31"),
 "url" : "www.example.com",
 "pageviews" : 52
}
```

```
db.stats.updateOne({url: 'www.example.com'},
...  {$inc: {pageview: 1}
...})
```

$inc - оператор инкремента

Операторы

$set - оператор вставки

```
db.movies.updateOne({ _id: 0 },
... { $set: { subtitle: 'Lord of the ring' } }
...)
db.movies.updateOne({ _id: 0 },
... { $unset: { subtitle: 1 } }
...)
```

$set - обновить только определенные поля в документе, не затрагивая остальные

$unset - удалит поле если оно есть.

Операторы

$set можно использовать для вложенных документов

```
{
"_id" : ObjectId("4b253b067525f35f94b60a31"),
"title" : "A Blog Post",
"author" : {
   "name" : "joe"
 }
}
> db.blog.updateOne({"author.name" : "joe"},
... {"$set" : {"author.name" : "david"}})
```

Массивы

В mongodb могут быть массивы - коллекции элементов

```
{
"_id" : ObjectId("4b253b067525f35f94b60a31"),
"title" : "A Blog Post",
"author" : {
   "name" : "joe"
 },
“comments”: [ { name: ‘Rock’ }, { name: ‘Rick’ } ]
}
```

Модификация массивов

$push - добавить элемент в конец

```
> db.blog.posts.updateOne({ title" : "A blog post" }, {
...    $push  : {
...      comments" : {
...        name : "joe",
...        email : "joe@example.com",
...        content : "nice post."
...      }
...    }
..   })
```

$addToSet - добавить элемент если его нет

Модификация массивов

```
> db.users.updateOne({ _id: 0 }, { $pop : { comments: 1 }})
```

Если цифра в key 1 то удалиться последний элемент списка, если -1, то удалиться первый элемент.

```
db.users.updateOne({ _id: 0 }, {
...  $pull: { comments: { name: 'Rick' } }
...})
```

$pull удалит все попавшиеся в массиве элементы которые совпадут с переданным.

Агрегация

Агрегация — это группировка значений многих документов.

3 способа:

- pipeline
- mapReduce
- одноцелевые методы

Pipeline

Фреймворк для агрегации. Является предпочтительным способом делать агрегацию в mongo

db.collection.aggregate()

По сути - многоэтапный конвейер преобразования документов в агрегированный результат.

Pipeline

Наша коллекция

{ "_id" : ObjectId("5a5779894a531b514a44e7c9"), "name" : "Toster", "spec" : "prog", "lvl" : 5 }

{ "_id" : ObjectId("5a5779894a531b514a44e7ca"), "name" : "Johny", "spec" : "prog", "lvl" : 2 }

{ "_id" : ObjectId("5a5779894a531b514a44e7cb"), "name" : "Kostya", "spec" : "cook" }

Самый простой агрегатор

db.authors.aggregate({ $match: { spec: "prog" } })

Добавим группировку

```
db.authors.aggregate(
 { $match: { spec: "prog" }},
 { $group: { _id: ‘result’, level: { $sum: '$lvl' } } }
)
{ "_id" : "level", "level" : 7 }
```

{ $group: { _id: ‘result’, level: { $sum: '$lvl' } } }

$group - агрегатор (команда группировки), вернет новые документы с результатами

_id - поле, по которому происходит группировка (если константа - агрегируем все строчке, если поле - то в рамках значения поля).

- _id: ‘result’ - константа, все строчки
- _id: ‘$spec’ - группируем строчки по полю spec

level - название поля с итоговым значением. В результирующем документе тут будет результат

$sum - оператор суммирования

$lvl - обращение к полю lvl текущего документа

Pipeline

Список всех операторов которые могут быть использованы в $group

- $sum — Возвращает сумму всех численных полей.
- $avg — Рассчитывает среднее значение между числовыми полями.
- $min — получит минимальное значение из числовых полей
- $max — Получить максимальное значение из числовых полей
- $push — Помещает значение поля в результирующий массив
- $addToSet — Вставляет значение в массив в результирующем документе, но не создаёт дубликаты.
- $first — Получает только первый документ из сгрупированных, обычно используется с сортировкой.
- $last — Получает последний документ

Pipeline

Если документы большие - то может не хватить памяти. Ее можно сэкономить, оставив только те поля в документе, которые нужны

```
db.authors.aggregate(
 { $match: { spec: "prog" }},
 { $project: { lvl: 1 } },
 { $group: { _id: 'level', level: { $sum: '$lvl' } } }
)
```

MapReduce

Строится вокруг двух функций:

- map - выбирает нужные поля
- reduce - сворачивает значения отдельных документов

MapReduce

коллекция:

{ status: 'Frontend', salary: 1000, work: 'programmer'}

{ status: 'Backend', salary: 1300, work: 'programmer'}

{ status: 'Analitycs', salary: 1000, work: 'CTO'}

Считаем среднюю зарплату по профессии:

```
const map = function () { emit( this.work, this.salary ) }

const reduce = function (key, values) {

 return (Array.sum(values) / values.length);

}

db.worker.mapReduce(map, reduce, { out: 'mapReduceCollections' })
```

MapReduce

Функция map внутри вызывает функцию emit, которая добавляет для “ключа” значение в “массив”. На выходе из функции map у нас будет такая коллекция:

```
[
 { "_id" : "CTO", "value" : [1000] }
 { "_id" : "programmer", "value" : [1000, 1300] }
]
```

MapReduce

Функция reduce заменяет массив на одно значение (аггрегат) - в нашем случае на среднее арифметическое

```
[
 { "_id" : "CTO", "value" : 1000 }
 { "_id" : "programmer", "value" : 1150}
]
```

MapReduce

Затем результат помещается в коллекцию, где его можно просмотреть

Одноцелевая агрегация

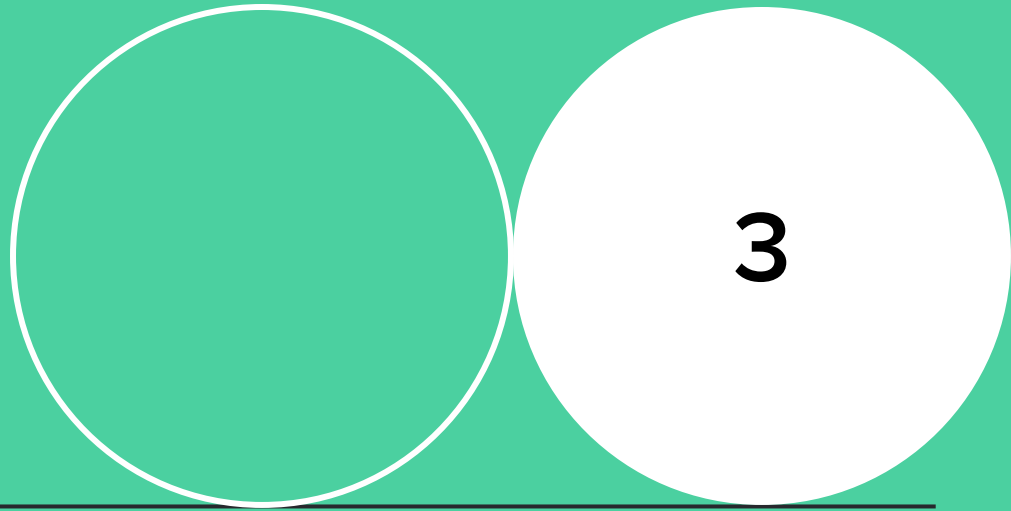
Агрегация одной коллекции по определенному ключу. Например:

db.users.count() -> количество элементов

db.products.distinct('tag') -> уникальные значения ключа

и тд

Домашнее задание



Алексей Кузьмин

Технологии работы с большими данными

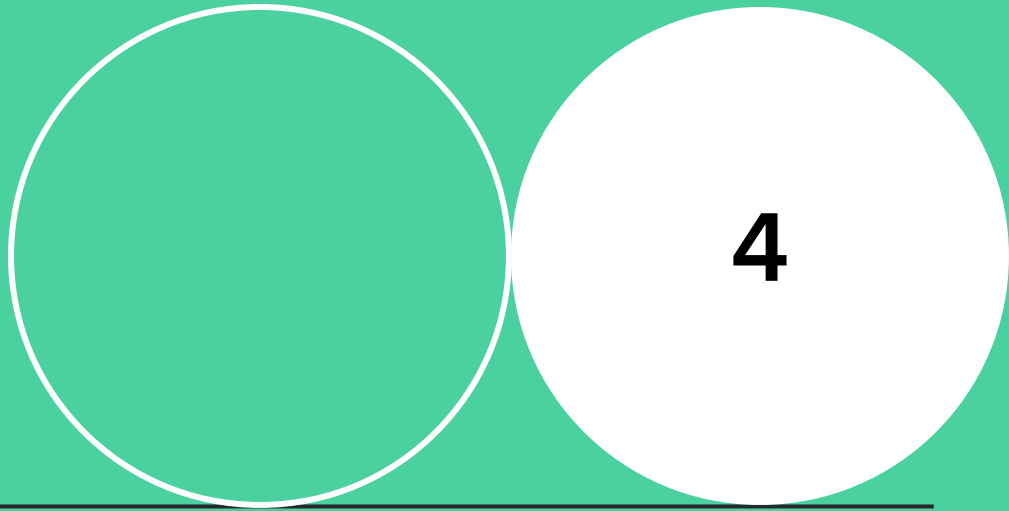
нетология

Домашнее задание

1. Взять online терминал mongodb - https://docs.mongodb.com/manual/tutorial/getting-started/
2. Создать базу данных
3. Вставить 4 документа по товарам на сайте. Атрибуты:
   a. Название
   b. Категория (2 товара из одной категории, 2 товара из другой)
   c. Цена
   d. Количество товаров на складе
4. Рассчитать остаточную стоимость товаров в каждой категории (сумма цены, умноженной на остаток)
5. Уменьшить количество товара на 1
6. Вывести top-2 самых дорогих товара

Результат прислать в виде скриншотов терминала с выполненными командами и их результатом

Полезные ссылки



Алексей Кузьмин

Полезные ссылки

- http://profyclub.ru/docs/167
- https://resources.mongodb.com/getting-started-with-mongodb
- https://www.ozon.ru/context/detail/id/8688130/
- https://www.guru99.com/create-read-update-operations-mongodb.html

Спасибо за внимание

Алексей
Кузьмин