НЕТОЛОГИЯ
групп

занятие 4

NoSQL & MongoDB

Алексей Кузьмин

Директор разработки; Data Scientist

ДомКлик.ру

aleksej.kyzmin@gmail.com

ЧТО СЕГОДНЯ ИЗУЧИМ

Что такое базы данных NoSQL?

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

NOSQL DB

CAP-теорема

- Consistency (Согласованность). Как только мы успешно записали данные в наше распределенное хранилище, любой клиент при запросе получит эти последние данные.
- Avalability (Доступность). В любой момент клиент может получить данные из нашего хранилища, или получить ответ об их отсутствии, если их никто еще не сохранял.
- Partition Tolerance (Устойчивость к разделению системы). Потеря сообщений между компонентами системы (возможно даже потеря всех сообщений) не влияет на работоспособность системы. Здесь очень важный момент состоит в том, что если какие-то компоненты выходят из строя, то это тоже подпадает под этот случай, так как можно считать, что данные компоненты просто теряют связь со всей остальной системой.

Теорема CAP (известная также как теорема Брюера) — эвристическое утверждение о том, что в любой реализации распределённых вычислений возможно обеспечить не более двух из трёх следующих свойств

Классы систем

В системе класса CA во всех узлах данные согласованы и обеспечена доступность, при этом она жертвует устойчивостью к распаду на секции.

Система класса 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.

В системе класса AP не гарантируется целостность, но при этом выполнены условия доступности и устойчивости к распаду на секции. Хотя системы такого рода известны задолго до формулировки принципа CAP (например, распределённые веб-кэши или DNS).

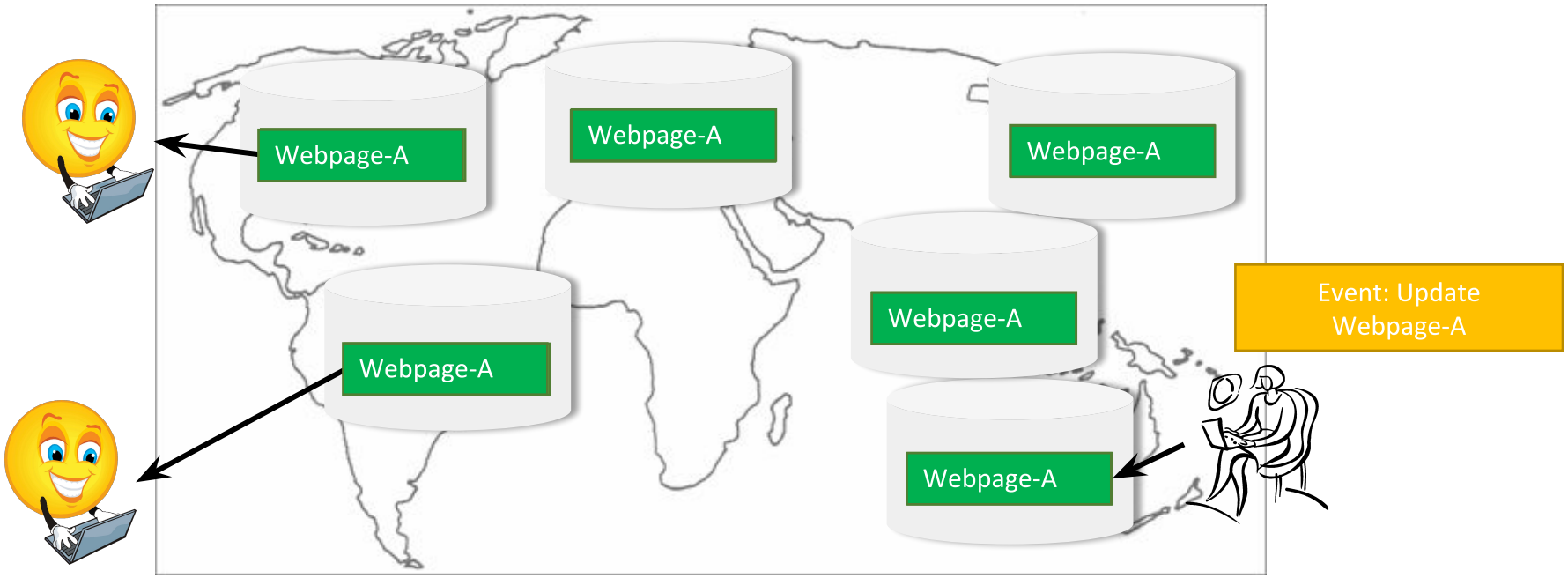
Свойства BASE

- Теорема CAP доказывает, что невозможно гарантировать строгую целостность и доступность, и при этом сохранять устойчивость к разделению.

- В частности, к таким БД применимы свойства BASE:
  - **Доступность в целом (BA)**: система обеспечивает доступность
  - **Гибкость (S)**: состояние системы может меняться со временем, даже без воздействия извне
  - **Согласованность в конечном счет (E)**: система, рано или поздно, придет в согласованное состояние

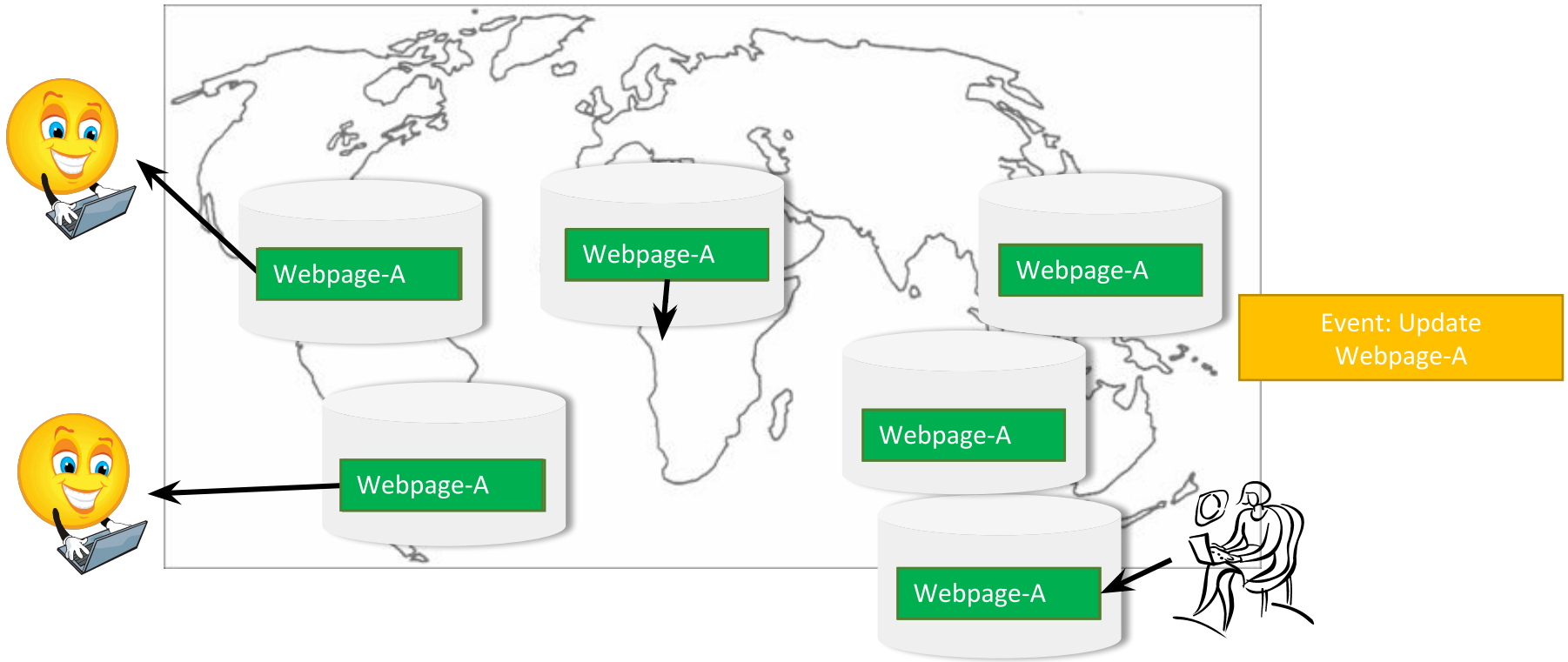
Согласованность в конечном счете

- Все копии постепенно станут согласованными в отсутствие обновлений



Согласованность в конечном счет

- Но, что если клиент обращается к данным из разных копий?



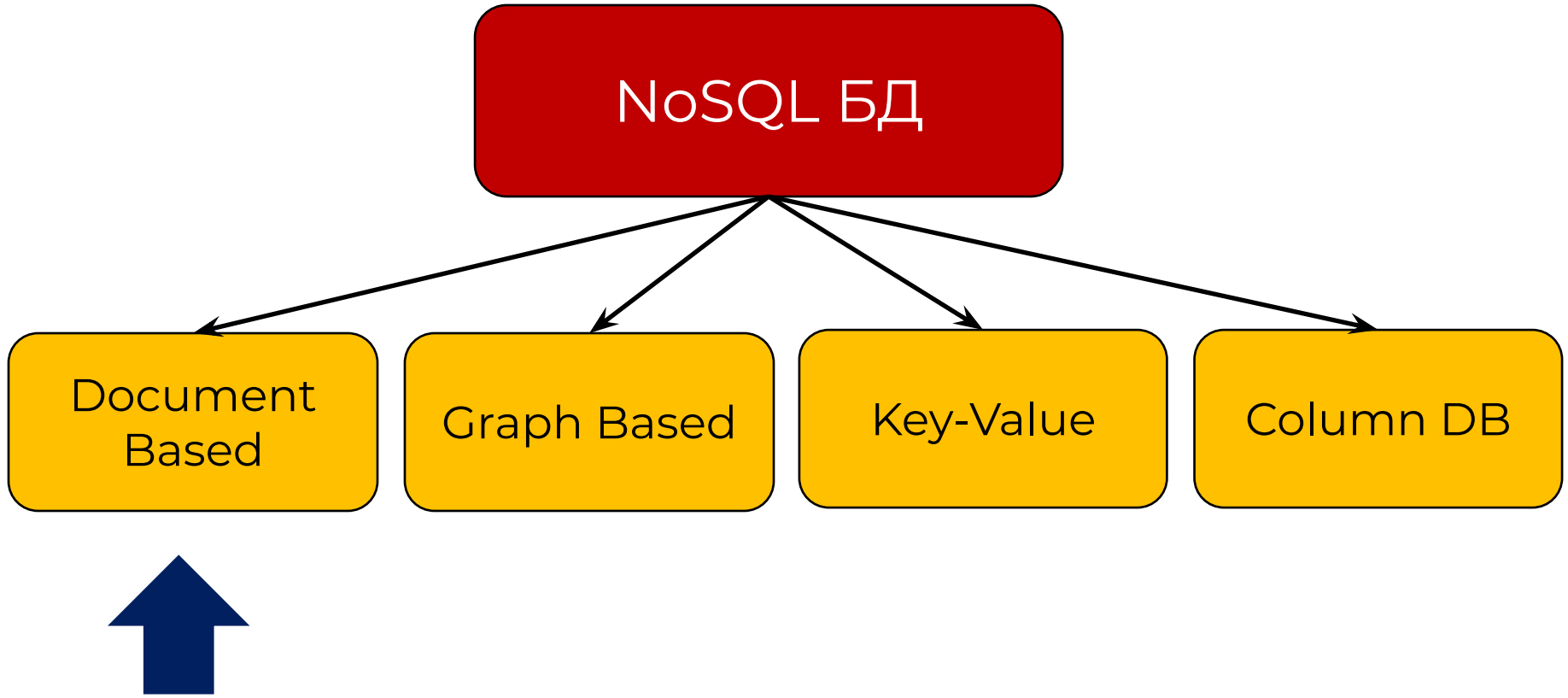
Протоколы формата (RYOW) - “читай то, что сам записал”

Для чего можно использовать базы данных NoSQL?

- Гибкость.
- Масштабируемость.
- Высокая производительность.
- Широкие функциональные возможности.

Типы NoSQL БД

Ограниченная классификация БД на NoSQL:



Хранилища документов

- Документы хранятся в каком-то стандартном формате (например, XML, JSON, PDF или документы формата Office)
  - В их отношении применим термин большие двоичные объекты (BLOB)

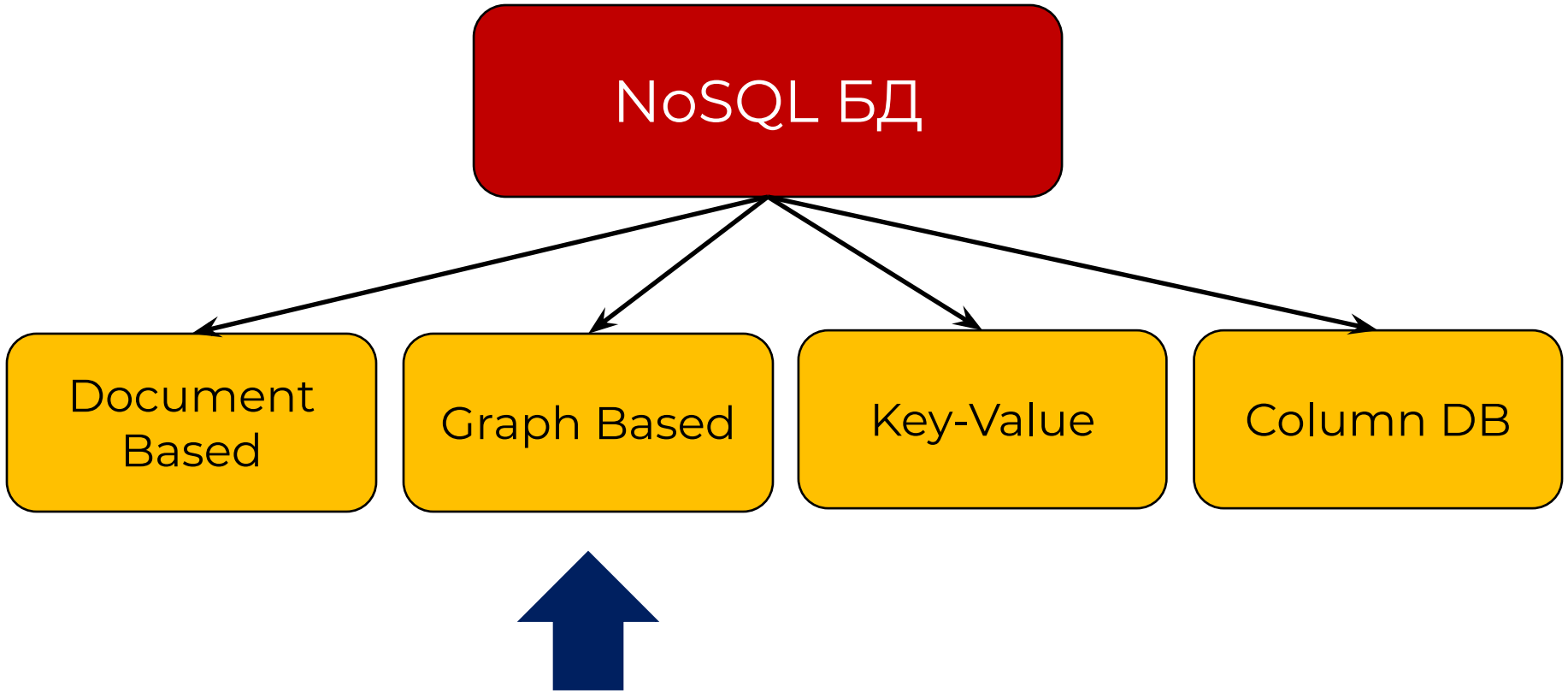
- Документы могут быть проиндексированы
  - Это хранилище документов, превосходящее традиционные файловые системы.
  - Например, MongoDB и CouchDB (к обеим можно обратиться с помощью методов MapReduce)

Место в мире больших данных

- Часто используют как замену традиционных баз данных
- Могут выступать первичным слоем сбора данных, который потом сбрасывается в долгосрочное хранилище
- На малых объемах может выступать как “единое” хранилище для работы с данными

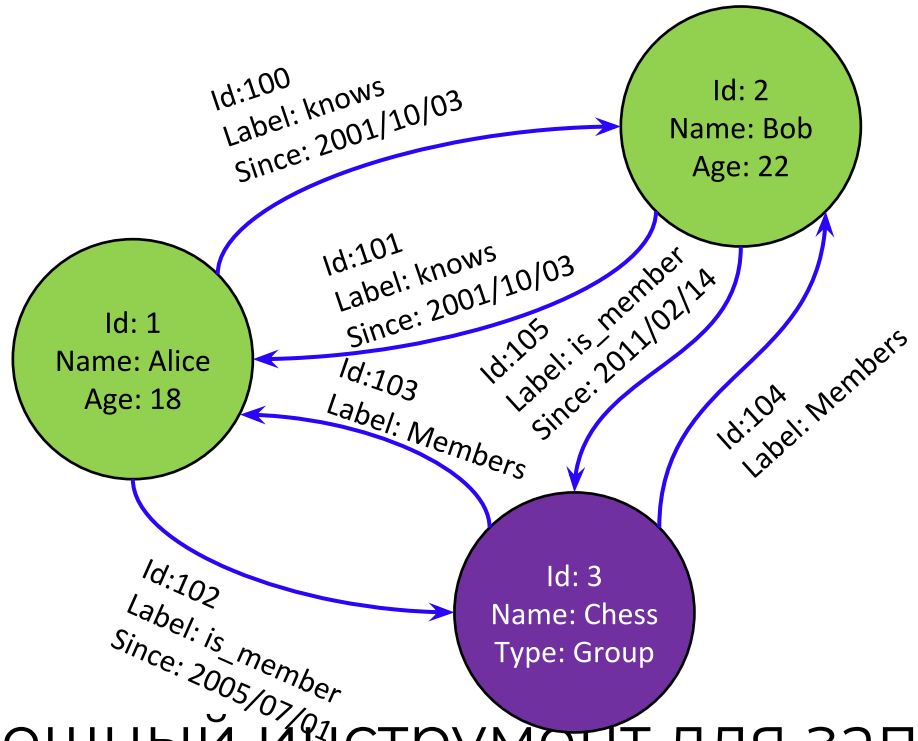
Типы NoSQL БД

Ограниченная классификация БД на NoSQL:



Графовые БД

- Данные представлены в виде вершин и углов



- Графовые БД - мощный инструмент для запросов, которые можно представить в виде отношений

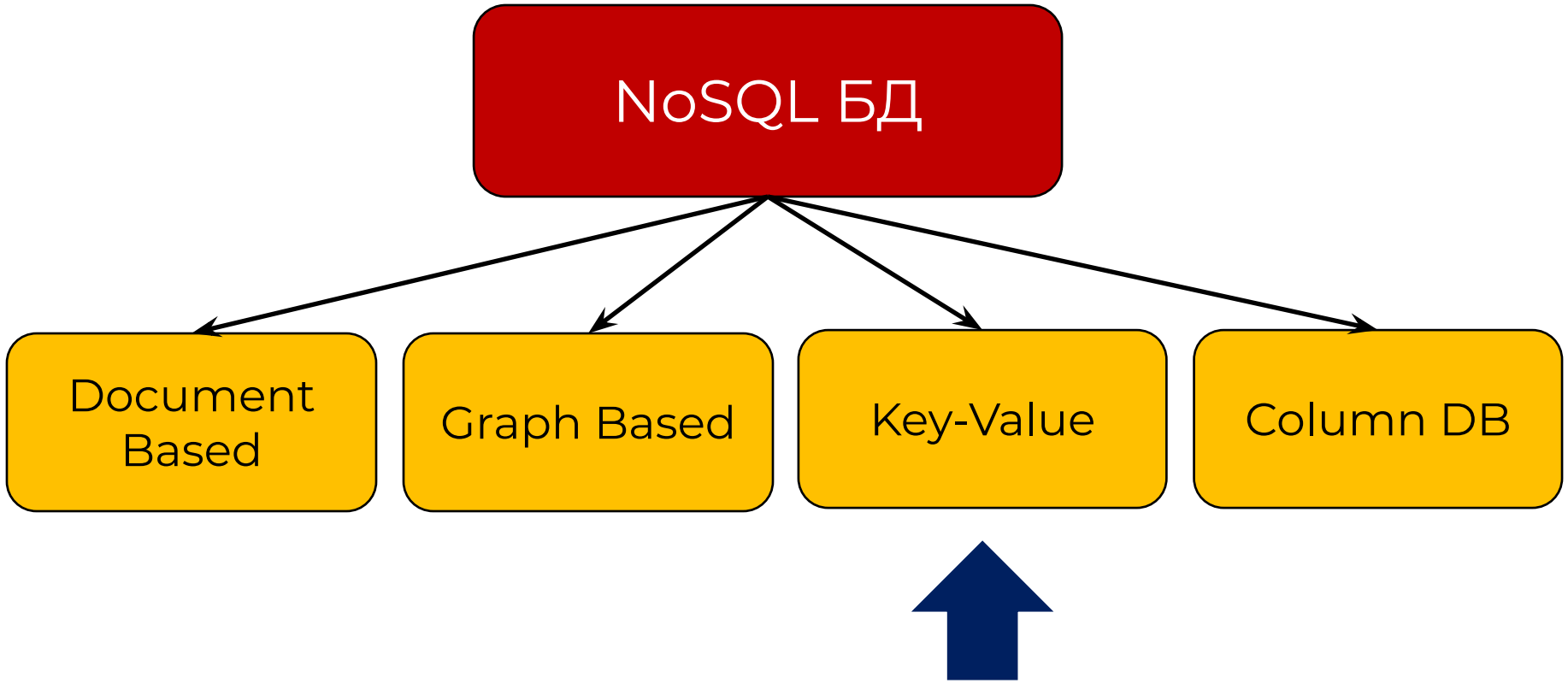
- E.g., Neo4j and VertexDB

Место в мире больших данных

- Для прикладной задачи, где есть отношения вида “сущность” и “связи между ними”

Типы NoSQL БД

Ограниченная классификация БД на NoSQL:



Хранилища ключей

- Ключи относятся, по возможности, к более комплексным значениям (например, спискам)
- Ключи могут храниться в таблице хэшей и легко распространяться
- Такие хранилища поддерживают, как правило, операции CRUD (создавать (C), читать (R), обновлять (U), и удалять (D))

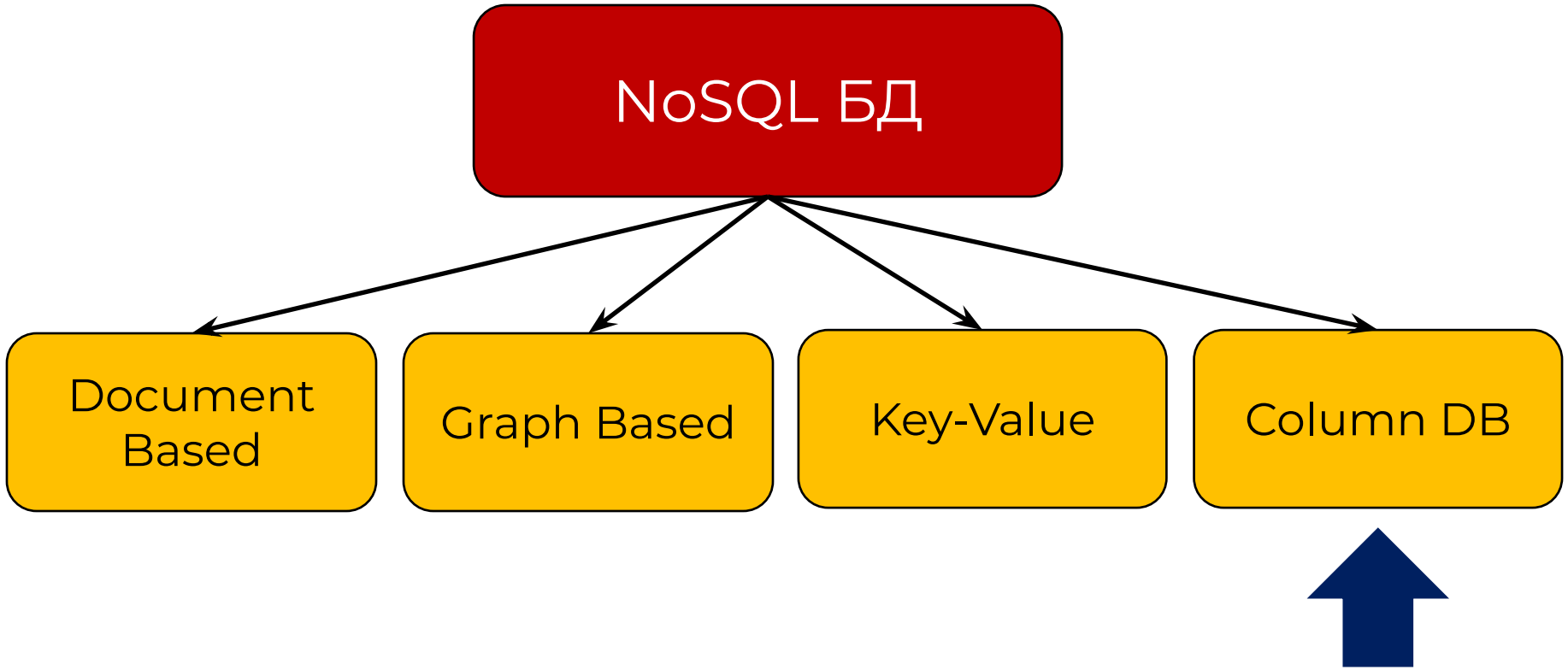
благодаря чему, нет ни соединений, ни агрегирующих функций

Место в мире больших данных

- Если живут в оперативной памяти - то справочники, к которым можно быстро получить доступ
- Если живут на диске - то долгосрочные хранилища слабоструктурированной информации

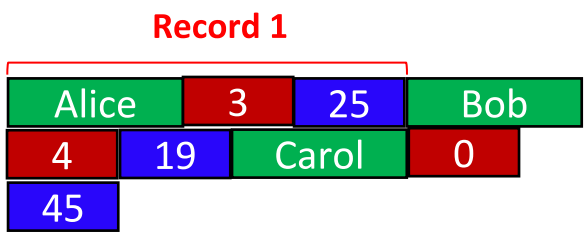
Типы NoSQL БД

Ограниченная классификация БД на NoSQL:

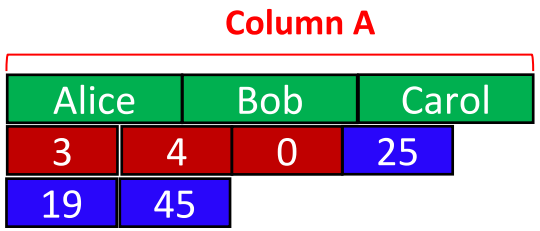


Колоночные БД

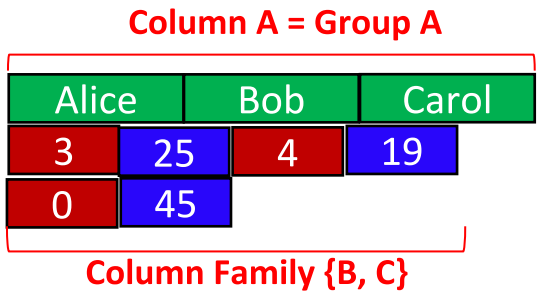
- Колоночные БД – гибрид реляционных СУБД и key-value
  - Значения хранятся в группах от 0 и более колонок, но в порядке колонок (в отличие от порядка “строчного”)



Row-Order



Columnar (or Column-Order)



Columnar with Locality Groups

  - Значения вызываются по подходящим ключам

- Например, HBase и Vertica

Место в мире больших данных

- Долгосрочные хранилища слабоструктурированной информации
- Batch-processing

MongoDB
документная база данных

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
}
```

```
db.people.find({ name: { first: "Joe" }}) -> явный поиск
db.users.find({ 'name.first': 'Joe' }) -> другой синтаксис
```

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: movie.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’ }
]
}
```

Поиск в массиве

```
db.blog.find({ comments: { name: 'Rock' } }) -> найдет наш коммент
```

`$all` -> поиск нескольких вхождений

```
db.blog.find({ comments:
 { $all: [{ name: 'Rock'}, {name: 'John' }] }
})
```

Если нам нужно сравнить только один определенный элемент массива:

```
db.blog.find({ 'comments.2': { name: 'John' } })
```

Количество элементов в массиве:

```
db.blog.find({ comments: { $size: 2} })
```

Поиск в массиве

Рассмотрим такой документ:

```
{ x: { $gt: 10, $lt: 20 } }
```

И запрос:

```
db.test.find({ x: { $gt: 10, $lt: 20}})
```

Подойдет ли масссив?

Поиск в массиве

Рассмотрим такой документ:

```
{ x: { $gt: 10, $lt: 20 } }
```

И запрос:

```
db.test.find({ x: { $gt: 10, $lt: 20}})
```

Подойдет ли масссив? да!

Ни 5, ни 25 не находится между 10 и 20, но документ возвращается, потому что 25 соответствует первому оператору (оно больше 10) и 5 соответствует второму оператору (оно меньше 20).

Решение - $elemMatch

```
db.test.find({ x: { $elemMatch: { $gt: 10, $lt: 20 }})
```

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" } })
```

Pipeline

Добавим группировку

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

`$group` - агрегатор, который вернет новый документ

`$sum` - аккумулятор, сохраняющий свое состояние

`$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
```

ВОПРОСЫ

Домашнее задание

Домашнее задание

1. Взять любой online терминал mongodb
   a. https://www.tutorialspoint.com/mongodb_terminal_online.php
   b. https://www.mplay.run/mongodb-online-terminal
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

Полезные материалы

- 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

НЕТОЛОГИЯ
групп

Спасибо за внимание!

Алексей Кузьмин

aleksej.kyzmin@gmail.com