



**Московский ордена Ленина, ордена Октябрьской Революции  
и ордена Трудового Красного Знамени  
государственный технический университет им. Н.Э. Баумана**

Реферат  
по курсу «Постреляционные СУБД»  
кафедры ИУ-5  
«Системы обработки информации и управления»  
на тему “MongoDB”

Проверила:  
Виноградова М.В.  
Выполнил:  
Студент группы ИУ5-38М  
\_\_\_\_\_/ Чернобровкин С.В.

Москва 2016 г.

<b>Описание</b>	<b>2</b>
<b>Краткий обзор основных возможностей</b>	<b>2</b>
<b>Формат хранения. Сравнение с реляционными СУБД</b>	<b>3</b>
<b>Простые операции</b>	<b>4</b>
Операция insert	4
Операция find	4
Операция update	6
Операция remove	7
<b>Агрегационный фреймворк Mongo</b>	<b>7</b>
\$Match	7
\$Group	8
Комбинация этапов в pipeline	9
\$Project	10
\$Unwind	10
Агрегаторы	11
<b>Join</b>	<b>11</b>
<b>Геоданные</b>	<b>12</b>
<b>Map-Reduce</b>	<b>12</b>
<b>Особенности репликации</b>	<b>14</b>
<b>Выводы</b>	<b>16</b>
<b>Литература</b>	<b>16</b>

## Описание

MongoDB - документоориентированная СУБД, написанная на C++.

Одним из популярных стандартов обмена данными и их хранения является JSON (JavaScript 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 называется BSON (БиСон) или сокращение от binary JSON.

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, с другой стороны, данный недостаток с лихвой окупается скоростью.

MongoDB хранит все данные в оперативной памяти для увеличения скорости работы, однако при этом является персистентной, то есть данные периодически сбрасываются на диск.

В Mongo не существует понятия транзакций, однако их можно реализовать вручную, используя несколько таблиц.

## Краткий обзор основных возможностей

1. Документо-ориентированная СУБД, хранение в формате BSON, чтение в JSON
2. Использование JavaScript для написания функций и запросов
3. Возможность выполнения операций, аналогичных операциям в реляционной СУБД
4. Наличие агрегационного фреймворка для написания сложных агрегационных запросов
5. Поддержка концепции map-reduce
6. Валидация полей документов
7. Поддержка хранимых процедур на JavaScript
8. Возможность репликации
9. Возможность использования в качестве основного хранилища для BigData, поддержка ведущих BigData-инструментов
10. Встроенная работа с гео-координатами.

## Формат хранения. Сравнение с реляционными СУБД

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```

← field: value  
← field: value  
← field: value  
← field: value

Рис. 1. Пример данных

Данные отображаются в формате JSON.

Аналогом реляционных “таблиц” выступают коллекции. Аналогом “кортежей” - документы. Коллекции в отличие от таблиц не имеют четкой схемы. В этом и недостаток и преимущество. С одной стороны это ускоряет скорость разработки и делает хранение более удобным, с другой стороны может возникать множество ошибок из-за рассинхронизации данных, так как нет никаких проверок полей.

Любая СУБД подходит только для определенного класса задач, в частности MongoDB отлично подходит, если:

- Данные слабо связаны между собой (нет соединений коллекций)
- Данные не структурированы (нельзя четко выделить схему бд)
- Схема данных все время обновляется (например на старте проекта)

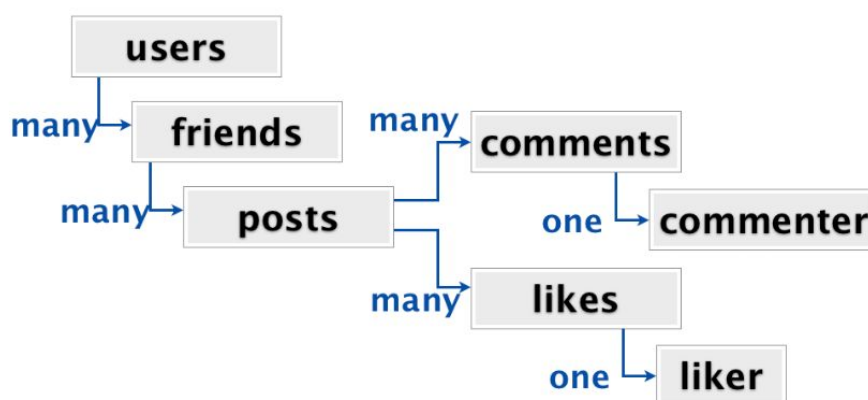


Рис. 2. Предметная область соцсети

На рисунке рассмотрена предметная область сущностей соцсети - пользователи, посты, комментарии, лайки. Как видно из рисунка каждая сущность связана с какой либо еще. Это отрицательный пример использования

MongoDB. При такой структуре предметной области хранение данных в mongo может только усложнить разработку.

## Простые операции

### Операция insert

Как и в классических реляционных СУБД insert отвечает за вставку документа в коллекцию.

Вызов функции делается у коллекции:

Обращаемся к коллекции users. Она будет создана автоматически.

```
> db.user.insert({name: "admin"})
```

```
WriteResult({ "nInserted" : 1 })
```

Внутри в функцию insert передается json-документ, он будет добавлен в коллекцию.

### Операция find

Операция find - аналог реляционного select. Служит для получения данных из коллекции. В аргументы find передаются “части” запроса, как это делается в классическом SQL (проекция, селекция, сортировка).

```
db.users.find(  
  { age: { $gt: 18 } },  
  { name: 1, address: 1 }  
) .limit(5)
```

- ← collection
- ← query criteria
- ← projection
- ← cursor modifier

Рис. 3. Пример операции find

В этом примере в аргументы find передается селекция (аналог WHERE в SQL) и проекция (список полей, которые необходимо вывести). Кроме этого в конце применяется модификатор limit, который ограничит итоговую выборку пятью записями.

Пример выше можно представить в SQL следующим образом:

```
SELECT _id, name, address  
FROM users  
WHERE age > 18  
LIMIT 5
```

- ← projection
- ← table
- ← select criteria
- ← cursor modifier

Рис. 4. Аналог find в SQL

Каждый из аргументов представляет собой JSON-документ с некоторыми встроенными функциями, как, например, \$gt в примере выше.

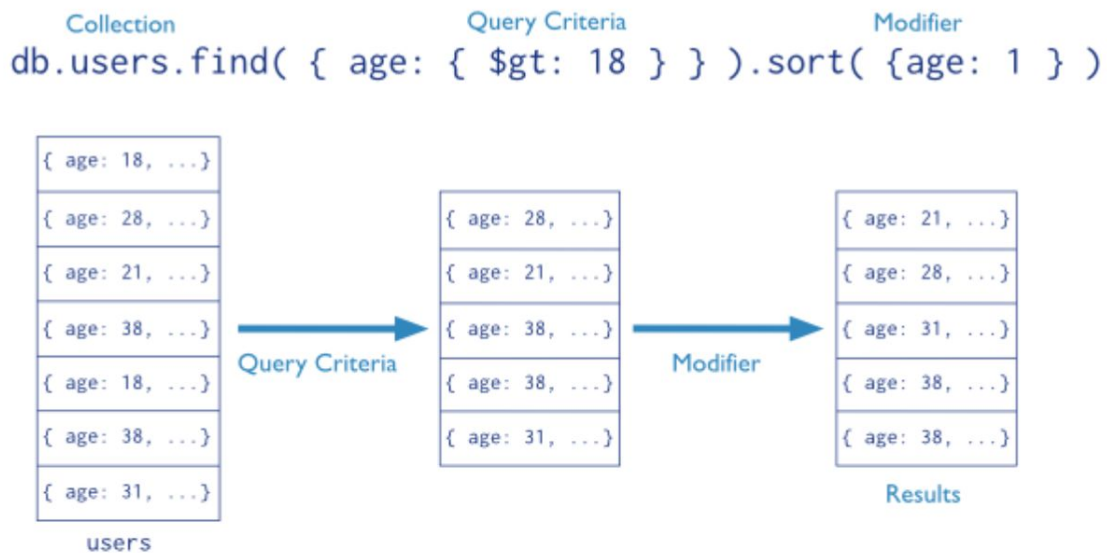


Рис. 5. Ход выполнения операции find

Результат запроса выглядит следующим образом: Выдается набор JSON-документов, состоящих из запрашиваемых полей.

```
db.users.find({age: 23}, {tags: 1}).limit(2)
```

```
{ "_id" : "56dd4dea02775f51f6f4d40c", "tags" : [ "tempor", "magna", "aliquip",
"aliqua", "et", "pariatur", "dolore" ] }
```

```
{ "_id" : "56dd4dea77d7e5cd2c03a8b7", "tags" : [ "proident", "non", "ut", "id",
"pariatur", "aliqua", "velit" ] }
```

ID в MongoDB представляет собой специальный ObjectId, который генерируется автоматически и есть у каждого документа коллекции. \_id представляет собой хеш от множества параметров сервера.

Кроме простых запросов в MongoDB можно комбинировать условия через операторы \$and или \$or:

```
db.users.find({name: "Hebert Franco", gender: "male"}, {_id: 1})
{ "_id" : "56dd51a06b83e49799fed610" }
```

```
db.users.find({$or: [{name: "Hebert Franco"}, {gender: "male"}]}, {_id: 1}).count()
512
```

Если перечислять условия через запятую, как в первом примере, то это автоматически преобразуется в соединение условий по логическому И. В

случае с ИЛИ необходимо явно указывать оператор `$or` и перечислять условия в массиве.

Одним из преимуществ отсутствия схемы является возможность использования вложенных полей и полей-массивов.

```
{
  "_id" : "56dd51a0ab304f8dde29b38a",
  "balance" : {
    "dollar" : 172.13,
    "ruble" : 3133.54
  }
}
```

В примере выше документ содержит вложенное поле `balance`, которое в свою очередь имеет собственную структуру.

Для поиска по таким полям необходимо в условии указывать полный “путь” до поля от корня документа, например:

```
db.users.findOne({"balance.ruble": {$gt: 1000}}, {balance: 1})
```

Если поле содержит в себе массив, то можно использовать специальный оператор `$elemMatch`. В таком случае запрос выдаст только те документы, в которых в поле-массиве хранится структура, удовлетворяющая условию в `elemMatch`.

```
db.users.find({friends: {$elemMatch: {name: "Lindsey Maynard"}}}).count()
```

## Операция update

Операция `update` служит для обновления документа.

В аргументы функции необходимо передать условие выборки и документ, на который будет заменен попавший под условие документ коллекции.

```
db.users.update({}, {is_active: false})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

В таком запросе будет найден документ, удовлетворяющий пустому условию (любой документ) и такой документ будет полностью заменен на документ `{is_active: false}`.

В этом запросе есть 2 неочевидные вещи:

- Обновится только 1 документ
- Обновится не одно поле документа (`is_active`), а весь документ

Чтобы обновить все документы, необходимо передать в опции `multi: true`. Чтобы обновить только одно поле, а не все, нужно использовать ключевое слово `$set`:

```
db.users.update({}, {$set: {is_active: false}}, {multi: true})
WriteResult({ "nMatched" : 1000, "nUpserted" : 0, "nModified" : 999 })
```

## Операция remove

Операция remove используется для удаления документов из коллекции:

Она принимает условие выборки. Все документы, удовлетворяющие этому условию, будут удалены

```
db.users.remove({"name" : "McCray Lopez"})
WriteResult({ "nRemoved" : 1 })
```

## Агрегационный фреймворк Mongo

Для агрегации в mongo используется отдельная функция - aggregate.

В классическом SQL агрегация производится с помощью группировки и агрегаторов:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;
```

В данном примере кортежи будут сгруппированы по column\_name и над каждой группой будет применена aggregate\_function.

Можно выделить несколько этапов запроса:

- Выборка нужных записей
- Группировка выбранных записей
- Применение агрегационной функции к группам

В MongoDB используется подобный механизм. В начале необходимо указать условие выборки, затем условие группировки, возможно некоторые дополнительные этапы, например сортировки и проекции. Все эти этапы собираются в массив, который называется pipeline. Документы коллекции проходят через pipeline и на выходе получается результат агрегации.

## \$Match

\$Match - один из основных операторов в агрегации. В этот этап передается условие выборки, аналогично условию в простых запросах find.

```
db.users.aggregate([
{
  $match: {
    name: "MARYJANE"
```



```

    }
  }
])

```

В данном примере из коллекции `users` будут выбраны все документы, у которых в поле `name` находится значение `maryjane`. Если в `pipeline` не добавить никаких других этапов, то такой запрос будет аналогичен простому запросу `db.users.find({name: "maryjane"})`

## \$Group

`$group` - оператор группировки, имеет следующий синтаксис:

```
{ $group: { _id: <expression>, <field1>: { <accumulator1> : <expression1> }, ... } }
```

В поле `_id` нужно передать выражение, по которому будет происходить группировка, например, это может быть название поля. Дальше в документе перечисляются поля, которые будут содержать агрегированные значения, которые получаются с помощью аккумуляторов, таких как `$sum`, `$max`, `$min`, `$avg`, `$push` и т.д.

```

db.getCollection('emp').aggregate([
  {
    $group: {
      _id: "$position",
      avg_age: {$avg: "$age"}
    }
  }
])

```

Этот запрос вернет документы из коллекции `emp` (сотрудники), сгруппированные по должности и для каждой должности будет посчитан средний возраст сотрудников:

```

{ "_id" : "intern", "avg_age" : 39.62658227848101 }
{ "_id" : "top-manager", "avg_age" : 41.572463768115945 }
{ "_id" : "director", "avg_age" : 42.457142857142856 }
{ "_id" : "senior", "avg_age" : 39.9453125 }
{ "_id" : "manager", "avg_age" : 40.474074074074075 }
{ "_id" : "teamlead", "avg_age" : 41.62328767123287 }
{ "_id" : "programmer", "avg_age" : 39.710743801652896 }
{ "_id" : "middle-programmer", "avg_age" : 41.1294964028777 }

```

Пример группировки по нескольким полям:

```
db.getCollection('emp').aggregate([
```

```

{
  $group: {
    _id: {position: "$position", chief_id: "$chief_id"},
    avg_age: {$avg: "$age"}
  }
}
})

```

## Комбинация этапов в pipeline

Как было сказано выше, этапы агрегации складываются в pipeline, который затем и проходят документы коллекции.

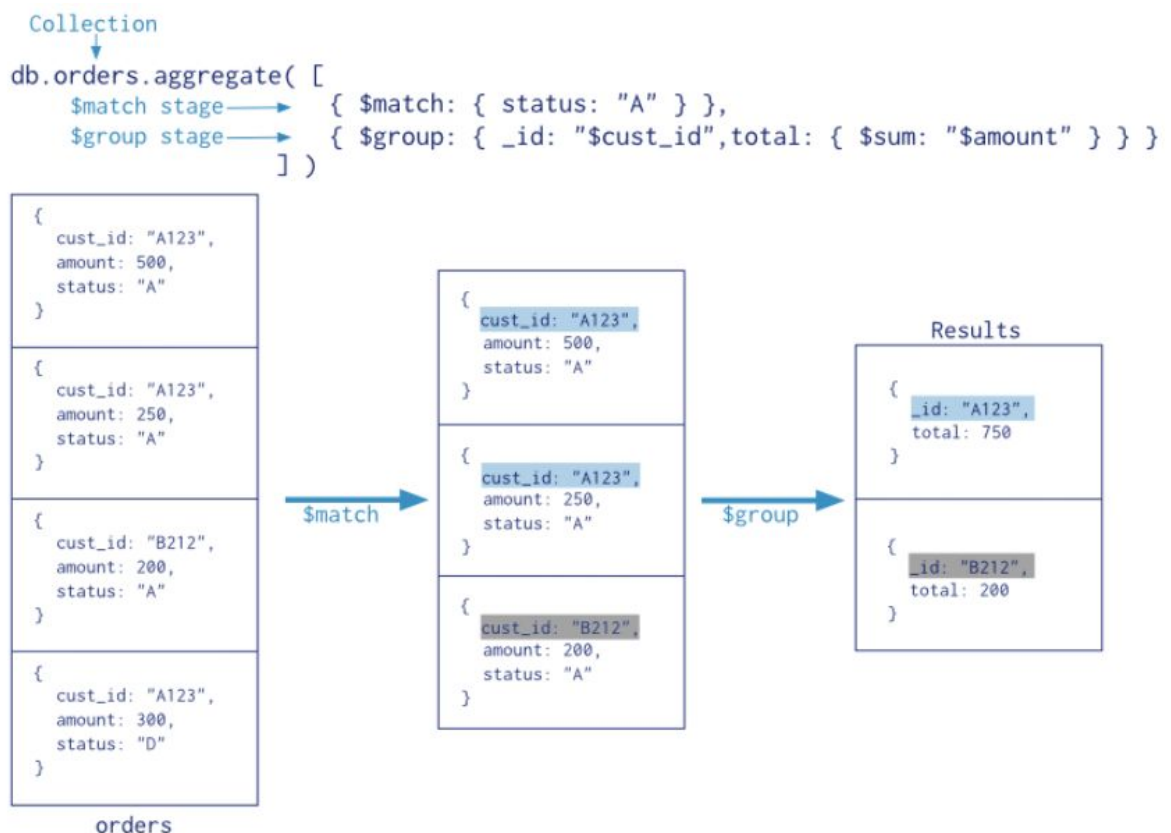


Рис. 6. Пример выполнения комбинированной агрегации

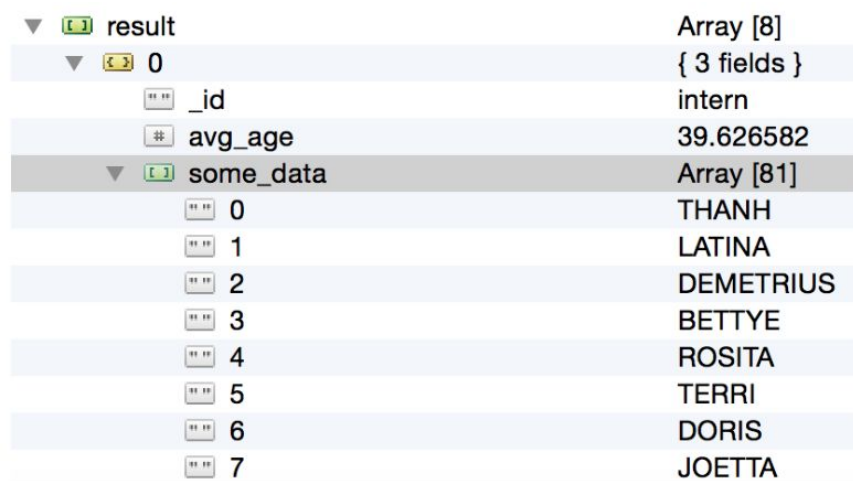
На рисунке рассматривается пример агрегации, где сначала выбираются документы, в которых поле статус = "А", а затем группируются по cust\_id и для каждой группы суммируются значения из поля amount.

## \$Project

Кроме выборки и группировки в агрегации можно использовать проекцию, также как и в простых запросах.

```
db.getCollection('emp').aggregate([
{
  $project: {
    "age": 1,
    "position": 1,
    "name": 1
  }
},
{
  $group: {
    _id: "$position",
    avg_age: {$avg: "$age"},
    some_data: {$addToSet: "$name"}
  }
}
])
```

В проекции можно указать поля, которые потребуются в запросе. Это сэкономит память, используемую при выполнении запроса.



▼ [3] result	Array [8]
▼ [2] 0	{ 3 fields }
_id	intern
avg_age	39.626582
▼ [3] some_data	Array [81]
0	THANH
1	LATINA
2	DEMETRIUS
3	BETTYE
4	ROSITA
5	TERRI
6	DORIS
7	JOETTA

Рис. 7. Пример результата группировки в программе RoboMongo

## \$Unwind

\$Unwind используется для преобразования массивов в отдельные документы. Например, как в примере на рисунке, в объекте есть поле

some\_data, которое представляет собой массив. В агрегации может потребоваться использовать значения массива и агрегировать их. Unwind создаст N новых документов из одного, где в поле с названием поля-массива будет находиться i-ое значение массива.

```
{ "_id" : 1, "item" : "ABC", "sizes" : [ "S", "M", "L" ] }
{ "_id" : 2, "item" : "EFG", "sizes" : [ ] }
{ "_id" : 3, "item" : "IJK", "sizes" : "M" }
{ "_id" : 4, "item" : "LMN" }
{ "_id" : 5, "item" : "XYZ", "sizes" : null }
```

Рис. 8. Пример данных с массивом

```
db.inventory.aggregate( [ { $unwind: "$sizes" } ] )
```

```
{ "_id" : 1, "item" : "ABC", "sizes" : "S" }
{ "_id" : 1, "item" : "ABC", "sizes" : "M" }
{ "_id" : 1, "item" : "ABC", "sizes" : "L" }
{ "_id" : 3, "item" : "IJK", "sizes" : "M" }
```

Рис. 9. Результат операции \$unwind

## Агрегаторы

В MongoDB поддерживается много агрегаторов, выделим основные из них:

- \$sum
- \$avg
- \$first - взять первый элемент группы
- \$last - взять последний элемент группы
- \$max
- \$min
- \$push - добавить элемент в массив
- \$addToSet - добавить элемент в множество

## Join

Начиная с версии 3.2 в MongoDB поддерживаются соединения коллекций. До этой версии соединения можно было сделать вручную в программном коде.

Join можно выполнить с помощью этапа \$lookup в агрегации:

\$lookup:

```
{
  from: "inventory",
```

```
    localField: "specs",  
    foreignField: "size",  
    as: "inventory_docs"  
  }  
}
```

from - коллекция, с которой происходит соединение.

localField - поле, в котором хранится ссылка на соединяемую коллекцию.

foreignField - поле, на которое ссылается ссылка в соединяемой коллекции.

as - поле, в которое будет записан документ присоединенной коллекции.

Пример выше можно было бы переписать на SQL так

A JOIN inventory on specs=size

## Геоданные

MongoDB поддерживает работу с геоинформацией.

В агрегационном фреймворке существует этап \$geoNear, который подбирает документы в заданной дальности. Это может быть удобно для приложений, которые активно работают с геоданными, например карты и навигаторы.

## Map-Reduce

В MongoDB поддерживается парадигма распределенных вычислений MapReduce. Можно рассматривать это как еще один тип агрегации, однако возможности MapReduce куда больше.

Mongo использует JavaScript для написания функций Map и Reduce, что очень удобно и дает свободу программисту решать, как агрегировать данные на более низком уровне нежели в aggregate.

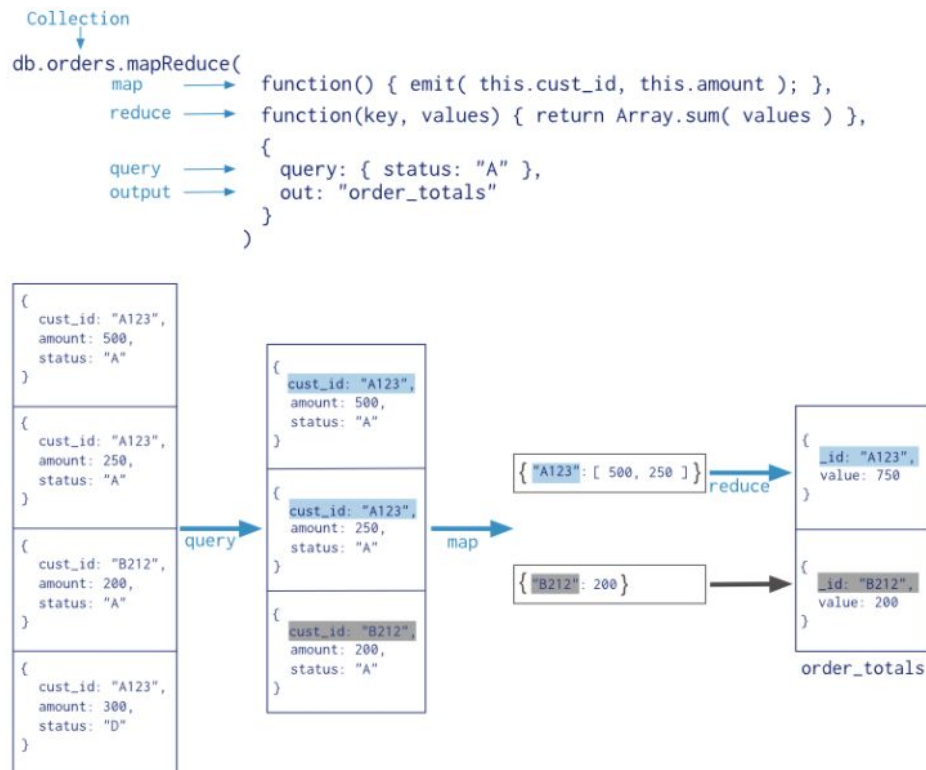


Рис. 10. Пример выполнения MapReduce

Как видно на рисунке в функцию `mapReduce` у коллекции передается функция `map`, `reduce` и блок опций.

Функция `map` применяется для каждого аргумента коллекции и может возвращать  $\geq 0$  документов с помощью оператора `emit`. Один оператор `emit` возвращает одну пару ключ-значение. Операторов `emit` в одной функции `map` может быть неограниченное количество, включая ноль.

После того, как на стадии `map` будут получены новые документы состоящие из ключей и значений, они будут сгруппированы по ключу и переданы следующей функции - `reduce`. Ей на вход поступает ключ и массив сгруппированных значений. Эта функция должна возвращать агрегированное значение для группы.

В `mapReduce` можно применить множество опций, например `query` (в результат попадут только те документы, которые удовлетворяют условию) и `out` (куда выводить результаты, на рисунке выше это коллекция `order_totals`).

Кроме этого может быть передана опциональная функция `finalize`. Она будет применяться для результата и формировать итоговую выдачу.

## Особенности репликации

Рассмотрим стандартную схему асинхронной репликации на примере MySQL:

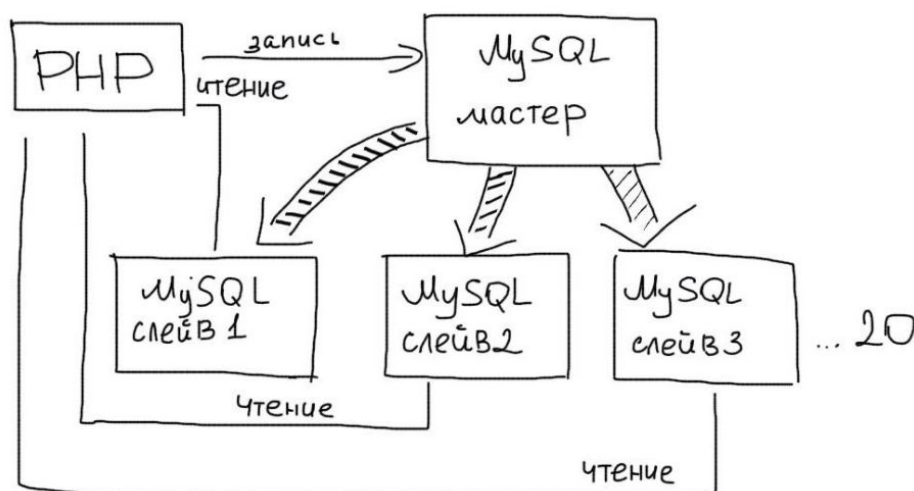


Рис. 11. Схема репликации MySQL

В этой схеме есть 2 типа узлов:

- Мастер-узел - используется для записи
- Слейв-узел - используется для чтения, дублирует записи мастер-узла.

В MongoDB используется похожая схема.

Группа узлов репликации в Mongo называется ReplicaSet. В ReplicaSet всегда есть Primary и Secondary узлы. Primary используются для записи аналогично мастер-узлам в стандартной модели репликации, а secondary аналогично slave-узлам.

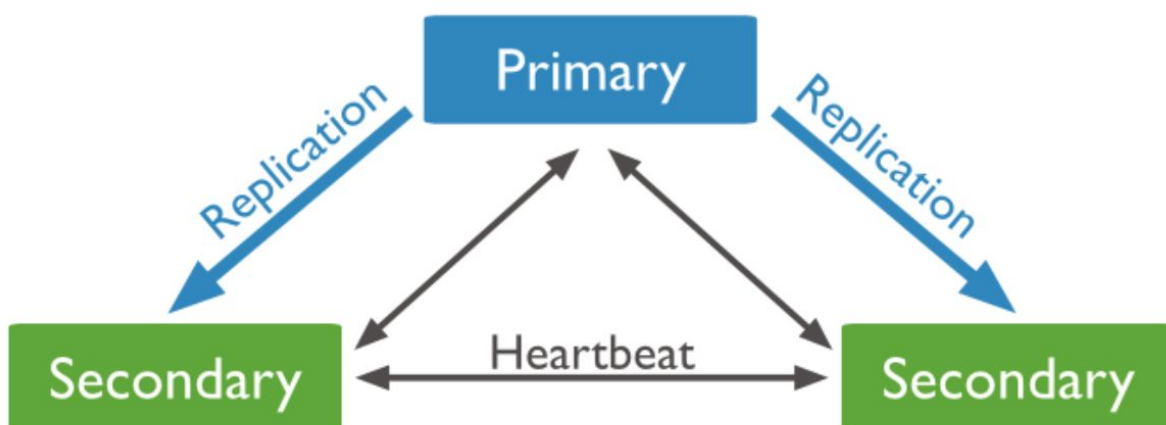


Рис. 12. Схема репликации в MongoDB

Secondary-узлы реплицируют данные Primary-узла, при этом обмениваясь специальными сигналами Heartbeat. Если один из узлов перестал отвечать на heartbeat, значит это отказавший узел. Если отказывает secondary-узел, то никаких проблем не возникает. В случае отказа Primary-узла весь кластер становится недоступным для записи, поэтому необходим новый Primary-узел.

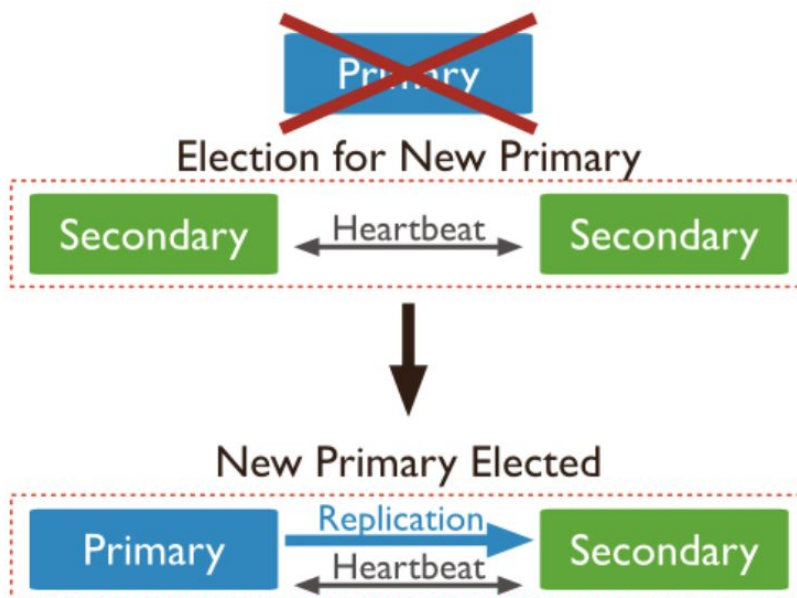


Рис. 13. Пример отказа Primary-узла

Определение нового primary-узла в mongo происходит с помощью специального механизма "выборов". Каждый из оставшихся узлов голосует за один из узлов, чтобы тот стал primary. Узел, набравший кворум голосов, становится Primary-узлом.

Отсюда следует, что минимальным работоспособным кластером будет replicaSet из трех узлов. Получается, что отказоустойчивость кластера =  $N/2 - 1$ , например, если кластер состоит из 6 узлов, то для выбора нового Primary-узла необходимо хотя бы 4 работоспособных узлов, чтобы набрать кворум голосов. Значит только 2 узла могут отказаться. Поэтому рекомендуется выбирать количество узлов нечетным.

Кроме secondary узлов в голосовании может участвовать еще один тип узла - Arbiter. Он не хранит данных, но может голосовать. Таким образом решается проблема необходимости большого числа голосов, так как не всегда оптимально делать множество реплик.

При конфигурировании replica set'a можно указывать приоритеты становления primary-узлом. Узел с нулевым приоритетом никогда не станет primary.

Кроме этого существует возможность указывать приоритеты для чтения данных. Например, можно указать, что чтение должно производиться только с primary, тогда secondary-узлы фактически работают в режиме бекапа. Узел



для чтения также можно выбирать в зависимости от географического положения сервера и клиента. Это может уменьшить накладные расходы передачи данных по сети.

## Выводы

В настоящее время MongoDB получила широкое распространение и используется во множестве проектов. Она поддерживает множество функций, сложных агрегаций и возможностей манипулирования данными. Является классической документоориентированной СУБД, не имеет четкой схемы данных. В последнее время MongoDB развивается в сторону HiLoad и BigData проектов, в частности не так давно появился коннектор для Spark. MongoDB хорошо подойдет как для быстрого старта MVP-проекта, так и для сложных BigData решений.

## Литература

1. Документация по MongoDB. <https://docs.mongodb.com/manual/>
2. Курс лекций “Структуры данных в web”. МАМИ, 2016
3. Статья “Почему вы никогда не должны использовать MongoDB”. <https://habrahabr.ru/post/231213/>