электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.655

ОСОБЕННОСТИ ПОСТРОЕНИЯ СЛОЖНЫХ ЗАПРОСОВ В MONGODB

Сафт А.А., *студент*
кафедра «Системы обработки информации и управления»,
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана

Научный руководитель: Гапанюк Ю. Е., к. т. н., доцент
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
chernen@bmstu.ru

Нереляционная СУБД MongoDB – это документо-ориентированная СУБД, которая не требует описания схемы базы данных. Разрабатывая данную СУБД, авторы хотели заполнить пустующую нишу между простыми, быстрыми и легко масштабируемыми хранилищами типа «ключ-значение» и сложными, поддерживающими сложноструктурированные запросы реляционными СУБД. На сегодняшний день MongoDB продолжает развивать парадигму хранения данных без жестко заданной схемы, в то же время обеспечивая превосходное горизонтальное масштабирование и поддержку сложных запросов практически на уровне РСУБД использующих SQL. MongoDB в качестве формата хранения данных использует BSON (Binary JSON) – бинарный вариант формата JSON (JavaScript Object Notation). Запросы строятся непосредственно в коде программы на любом языке программирования, для которого есть соответствующий драйвер (практически все распространенные языки имеют такие драйверы). В качестве языка администрирования используется JavaScript. Помимо отсутствия схемы, важным отличием формата хранения данных является использование деревьев данных и поддержка массивов. Основной элемент СУБД – база данных. База данных содержит коллекции, аналогом которых в РСУБД являются таблицы. Коллекции содержат документы, состоящие из полей. Данные слаботипизированы, 4 основных типа: число, строка, объект, временная отметка. Каждый документ включает в себя поле с названием “_id”, которое содержит первичный ключ и является уникальным для всей базы данных. Типичным примером документа может служить такой вариант:

```
{
    "_id" : ObjectId("512e35cd878ca53d831c7da0"),
```

```
    "int" : 1,
    "float" : 1.5,
    "string" : "Пример строки",
    "subdocument" : {
        "var1" : 1,
        "var2" : "var2"
    },
    "array" : [
        1,
        1.5,
        "Строка в массиве"
    ]
}
```

Единственным недостатком такой организации хранения данных является отсутствие возможности создавать внешние ключи на другие коллекции, иными словами операция JOIN в MongoDB невозможна. Эта проблема во многом решается использованием вложенных документов при грамотном проектировании базы данных.

При сложных выборках используются следующие 3 типа конструкций: aggregation framework, map-reduce и простые запросы агрегации данных: group, distinct, count.

Aggregation framework представляет более полный инструмент выборки, чем стандартная функция find(). Следующий вызов: db.test.find({field: “value”});

будет аналогичен такому:

db.test.aggregate($match: {field: “value”});

Здесь и далее в качестве рабочей коллекции будем использовать коллекцию test.

В целом, следующие вызовы идентичны:

db.test.find({cat_id: 5, price: {$lt: 1000}}).sort({price:-1}).limit(10);

db.test.aggregate({$match: {cat_id: 5, price: {$lt: 1000}}}, {$sort{price:-1}}, {$limit:10});

Более интересными представляются операторы, которые не применимы в обычной выборке: $project, $unwind. Оператор $project позволяет добавлять в финальную выборку поля по выбору, а оператор $unwind «разматывает» массив. Для коллекции, состоящей из таких документов:

```
{
"_id" : ObjectId("512e35cd878ca53d831c7da1");
```

consumer_id: ObjectId("512e35cd878ca53d831c81a5");
cart_id : 155,
goods: ["player", "phone", "tablet"]
}
такой запрос:
db.test.aggregate({$project: {cart_id: 1, goods:1, _id:0}}, {$unwind: "$goods"});
вернет следующий результат:

```
{
        "result" : [
                {
                        "cart_id" : 155,
                        "goods" : "player"
                },
                {
                        "cart_id" : 155,
                        "goods" : "phone"
                },
                {
                        "cart_id" : 155,
                        "goods" : "tablet"
                }
        ],
        "ok" : 1
}
```

В отличие от прочих операторов, которые могут принять на вход название поля без дополнительного описания, оператор $unwind требует представить название поля как строку (описать ее в кавычках), начинающуюся с символа “$”.

При необходимости группировки результатов по некоторому полю используется оператор $group, однако этой функцей намного эффективнее пользоваться при помощи операции map-reduce или специальной функции group.

Операция map-reduce является реализацией одноименной технологии, основанной на обработке информации в два этапа. Первый этап – map – собирает данные, проходя по коллекции и обрабатывая каждый документ, в результате на выходе создавая пару вида ключ-значение. Второй этап – reduce – последовательно обрабатывает все пары,

созданные функцией map. В качестве эталонной задачи для такой технологии можно рассмотреть задачу поиска суммы квадратов натуральных чисел в некотором диапазоне. В таком случае функция map поэлементно возведет входные числа в квадрат, а функция reduce сложит вместе все квадраты, полученные на первом шаге. В MongoDB дано такое определение функции mapReduce:

db.collection.mapReduce(
<mapfunction>,
<reducefunction>,
{
out: <collection>,
query: <document>,
sort: <document>,
limit: <number>,
finalize: <function>,
scope: <document>,
jsMode: <boolean>,
verbose: <boolean>
}
), где

out – коллекция, куда записываются выходные данные,

query – документ, по которому строится входная выборка,

sort – поля, по которым производится сортировка,

limit – ограничение по количеству документов,

finalize – функция, на вход которой идет результат функции reduce, обрабатывающая этот результат,

scope – документ, содержащий глобальные переменные относительно функций map, reduce и finalize,

jsMode – параметр, определяющий нужно ли сериализовывать данные между выполнением функций в BSON

verbose – параметр, включающий в результат данные о времени выполнения.

Функция map должна содержать вызов функции emit, которая составляет пары вида ключ-значение, используемые дальше. Примером выполнения функции mapReduce может служить такой вариант:

```
var mapFunction = function() {
    emit(this.cat_id, this.price);
```

```
};
var reduceFunction = function(keyCustId, valuesPrices) {
return Array.sum(valuesPrices);
};
db.test.mapReduce(
mapFunction,
reduceFunction,
{
out:  {inline:1}
}
);, где
```

out: {inline:1} направляет вывод в оболочку mongo.

В данном примере была использована группировка по полю cat_id, и для каждой категории подсчитана суммарная стоимость входящих в нее элементов.

Для агрегации данных после выполнения функции reduce, применяется функция finalize. Она принимает ключ, созданный функцией map и результат работы reduce. Таким образом можно посчитать среднее арифметическое некоторого поля. Для примера можно взять предыдущий случай и считать не суммарную стоимость товаров в категории, а среднюю стоимость товара в каждой из категорий:

```
var mapFunction =     function()       {
emit(this.cat_id,         this.price);
};
var reduceFunction = function(keyCatId, valuesPrices)      {
returnValues   = {price:0, count:0};
returnValues.price = Array.sum(valuesPrices)
returnValues.count = valuesPrices.length;
return  returnValues;
};
var finalizeFunction = function(key, reducedValue) {
return reducedValue.avgPrice = reducedValue.price/reducedValue.count;
}
db.test.mapReduce(
mapFunction,
reduceFunction,
```

```
{
out: {inline:1},
finalize: finalizeFunction,
jsMode: true,
verbose: true
}
);
```

Результатом будет документ следующего вида:

```
{
"results" : [
{
"_id" : 1,
"value" : 51.1
},
{
"_id" : 2,
"value" : 50.76
},
{
"_id" : 3,
"value" : 50.62
}
],
"timeMillis" : 16,
"timing" : {
"mapTime" : 11,
"emitLoop" : 14,
"mode" : "mixed",
"total" : 16
},
"ok" : 1,
};
```

Параметр jsMode говорит о том, что данные между функциями map, reduce и finalize не нужно сериализовывать в BSON. Это дает серьезную экономию вычислительных ресурсов и, как следствие, запрос выполняется быстрее. При отсутствии

сериализации, запросы, как правило выполняются на 25-30 % быстрее. Такой запрос на 15000 элементов выполняется в среднем 650 мс с сериализацией и 450 мс без нее. Параметр verbose добавляет в выборку поле timing, содержащее более подробные данные о времени выполнения запроса.

Таким образом, функция mapReduce позволяет выполнять сложные запросы, в том числе с группировкой и проводить анализ полученных данных. Однако функции map и reduce имеют некоторое количество собственных ограничений. Функция map не должна влиять на данные, содержащиеся в базе данных, то есть не должна иметь посторонних эффектов. Функция reduce, в свою очередь также не должна иметь посторонних эффектов, должна быть идемпотентной относительно принимаемого значений (то есть вызов reduce(key, [reduce(key, valuesArray)]) должен иметь один и тот же результат с вызовом reduce(key, valuesArray)). MongoDB не вызовет функцию reduce если значение, соответствующее некоторому ключу единственно.

Важной особенностью операции mapReduce является то, что она автоматически будет выполнена без необходимости менять запрос на любом кластере, в том числе реплицированном и сегментированном. В этом и заключается ее преимущество: функция позволяет обрабатывать данные параллельно, большими объемами без необходимости задействовать мощные сервера. Легкая настройка репликации серверов и сегментирования баз данных и коллекций вместе с мощными инструментами агрегации и анализа данных позволяют эффективно использовать даже небольшие вычислительные ресурсы там, где для РСУБД потребовались бы более объемные расходы, в том числе на обслуживание. В среднем, сложные запросы проще описываются операцией mapReduce (в виду ее высокой и однозначной структурированности), чем сложным SQL-запросом. Также следует заметить, что тонкая настройка и профилирование запросов РСУБД – задача гораздо более сложная, зачастую требующая отдельного специалиста высокой квалификации.

Для выполнения частых задач, таких как группировка, поиск уникальных значений и подсчет количества элементов в MongoDB предусмотрены отдельные функции: group, distinct и count.

Функция group похожа на mapReduce, но вместо функции map используется документ, содержащий поля группировки, либо функция keyf, которая находит нужные поля (например, если они динамически изменяются). Основное отличие от mapReduce состоит в том, что функция group не работает с сегментированными кластерами, а также ограничена двадцатью тысячами уникальных ключей в результирующей выборке. Общий синтаксис функции group:

```
db.collection.group({
key,
reduce,
initial,
 [keyf,]
 [cond,]
finalize
});
```

Поля reduce и finalize аналогичны таким же полям в функции mapReduce, поле initial содержит объявление переменных, используемых в функции reduce, поле cond содержит запрос, фильтрующий входные данные (аналог поля query в mapReduce).

Запрос, аналогичный примеру с подсчетом средней цены категории:

```
var reduceFunction = function(doc, ret) {
ret.price+=doc.price;
ret.count++;
return ret;
};
var finalizeFunction = function(ret) {
if (ret.count>0) return ret.avgPrice=ret.price/ret.count;
}
db.test.group({
key: {cat_id:1},
initial: {price:0, count:0, avg_price:0},
reduce: reduceFunction,
finalize: finalizeFunction
});
```

Результатом которого будет массив: [51.1, 50.76, 50.62]

Аналогом такого запроса на языке SQL будет следующий запрос:

```
SELECT AVG(price)
FROM test
GROUP BY cat_id;
```

При этом следует отметить, что при возрастании сложности запроса, SQL-запрос в размере и сложности будет увеличиваться очень быстро, в то время как вызов group поменяется слабо, в виду его жесткой и универсальной структуры.

Функция reduce при внешней схожести с одноименной функцией в mapReduce, имеет несколько иные параметры:

Первый параметр (в примере "doc") является документом агрегируемой коллекции.

Второй параметр (в примере "ret") является документом, все поля которого инициализируются в поле initial. Этот же документ возвращается в качестве результата и используется в качестве входного и выходного параметра функции finalize.

Функция disctinct возвращает массив уникальных значений указанного поля. В качестве второго параметра может выступать фильтрующий документ:

```
db.test.distinct("cat_id")
[ 1, 2, 3 ]
db.test.distinct("cat_id", {cat_id: {$lt: 3}})
[ 1, 2 ]
```

Функция count возвращает число найденных документов по заданному условию:

```
db.test.count({cat_id: {$lt: 3}})
100
```

Также она может применяться к курсору:

```
db.test.find({cat_id: {$lt: 3}}).count()
100
```

В данной статье были рассмотрены ключевые особенности построения сложных запросов в MongoDB, даны их краткие характеристики и сравнения с аналогами в РСУБД, а также примеры использования таких запросов.

Список литературы

1. Кайл Бэнкер. MongoDB в действии. / Пер. с англ. Слинкина А. А. – М.: ДМК Пресс, 2012. – 394с.: ил.
2. Aggregation. MongoDB.org. URL: http://docs.mongodb.org/manual/aggregation/ (дата обращения: 27.02.13).