Name : Rushikesh Kazbhari Palve
Roll No. 31258

Date : 1/20 | Page No :

Assignment No. B3

DOP :- 24-11-2021 DOS :- 30-11-2021

Title :- MongoDB - Map-reduces operations -

Problem Definition :-

Implement Map reduces operation with suitable example using MongoDB :

Objectives :-

To understand concept of Map-reduce as data processing paradigm for condensing large volumes of data into useful aggregated results.

Learning Outcomes :-

After completion of the assignment, students will be able to understand concept of Map-reduce.

Theory :-

As per the MongoDB documentation, Map-Reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. MongoDB uses

mapReduce command for map-reduce operations.

mapReduce command :-

Following is the syntax of the basic mapReduce command -

```
>db.collection.mapReduce(
    function() { emit(key, value); }, //map function
    function(key, values) { return reduceFunction }, //reduce function
    {
        out : collection,
        query : document,
        sort : document,
        limit : number
    })
```

The Map-reduce function first queries the collection, then maps the result documents to emit key-value pairs, which is then reduced based on the keys that have multiple values.

In the above syntax -

i) map is a javascript function that maps a value with a key and emits a key-value pair.
ii) reduce is a javascript function that reduces or groups all the documents having the same key.
iii) out specifies the location of the map-reduce query result.
iv) query specifies the optional selection criteria

for selecting documents.

v) sort specifies the optional sort criteria.

vi) limit specifies the optional maximum number of documents to be returned using MapReduce.

Consider the following document structure storing user posts. The document stores user_name of the user and the status of post.

```
{"post_text": "tutorialspoint is an awesome
                website for tutorials",
 "user_name": "mark",
 "status": "active"
}
```

We will use a mapReduce function on our posts collection to select all the active posts, group them on the basis of user_name and then count the number of posts by each user using the following code

```
> db.posts.mapReduce(
    function() { emit(this.user_id, 1); },
    function(key, values) {return Array.sum(values)
                                    },
    {
      query: {status: "active"},
      out: "post_total"})
```

the above mapReduce query outputs the following result -

```
{
  "result" : "post_total",
  "timeMillis" : 9,
  "counts" :
    {
      "input" : 4,
      "emit" : 4,
      "reduce" : 2,
      "output" : 2
    }
  "ok" : 1,
}
```

the result shows that a total of 4 documents matched the query (status : "active"), the map function emitted 4 documents with key-value pairs and finally the reduce function grouped mapped documents having the same keys into 2.

To see the result of this mapReduce query, use the find operator -

```
> db.posts.mapReduce( function() { emit ( this.user-id, 1) ; }, function (key, values) { return Array.sum(values) }, { query : { status : "active" } , out : "post_total" }).find()
```

The above query gives the following result which indicates that both users tom and mark

have two parts in active states -

{"_id": "tom", "value": 2}
{"_id": "mack", "value": 2}

In a similar manner, MapReduce queries can be used to construct large complex aggregation queries. The use of custom Javascript functions make use of MapReduce which is very flexible and powerful.

CONCLUSION :-

Thus we have studied Map-Reduce function.

OUTPUT :-

```
> show dbs
admin              0.000GB
assignment_no_10   0.000GB
assignment_no_9    0.000GB
config             0.000GB
local              0.000GB


> use assignment_no_11
switched to db assignment_no_11


> db.createCollection("Orders")
{ "ok" : 1 }


> db.Orders.insert({_id:1, custId:101, ordDate:new Date("2021-11-01"), price:25})
WriteResult({ "nInserted" : 1 })
> db.Orders.insert({_id:2, custId:101, ordDate:new Date("2021-11-08"), price:70})
WriteResult({ "nInserted" : 1 })
> db.Orders.insert({_id:3, custId:102, ordDate:new Date("2021-11-08"), price:50})
WriteResult({ "nInserted" : 1 })
> db.Orders.insert({_id:4, custId:102, ordDate:new Date("2021-11-18"), price:25})
WriteResult({ "nInserted" : 1 })
> db.Orders.insert({_id:5, custId:102, ordDate:new Date("2021-11-19"), price:50})
WriteResult({ "nInserted" : 1 })
> db.Orders.insert({_id:6, custId:103, ordDate:new Date("2021-11-19"), price:35})
WriteResult({ "nInserted" : 1 })
> db.Orders.insert({_id:7, custId:103, ordDate:new Date("2021-11-20"), price:25})
WriteResult({ "nInserted" : 1 })
> db.Orders.insert({_id:8, custId:104, ordDate:new Date("2021-11-20"), price:75})
WriteResult({ "nInserted" : 1 })
> db.Orders.insert({_id:9, custId:104, ordDate:new Date("2021-11-20"), price:55})
WriteResult({ "nInserted" : 1 })
> db.Orders.insert({_id:10, custId:104, ordDate:new Date("2021-11-23"), price:25})
WriteResult({ "nInserted" : 1 })


> db.Orders.find()
{ "_id" : 1, "custId" : 101, "ordDate" : ISODate("2021-11-01T00:00:00Z"), "price" : 25 }
{ "_id" : 2, "custId" : 101, "ordDate" : ISODate("2021-11-08T00:00:00Z"), "price" : 70 }
{ "_id" : 3, "custId" : 102, "ordDate" : ISODate("2021-11-08T00:00:00Z"), "price" : 50 }
{ "_id" : 4, "custId" : 102, "ordDate" : ISODate("2021-11-18T00:00:00Z"), "price" : 25 }
{ "_id" : 5, "custId" : 102, "ordDate" : ISODate("2021-11-19T00:00:00Z"), "price" : 50 }
{ "_id" : 6, "custId" : 103, "ordDate" : ISODate("2021-11-19T00:00:00Z"), "price" : 35 }
{ "_id" : 7, "custId" : 103, "ordDate" : ISODate("2021-11-20T00:00:00Z"), "price" : 25 }
{ "_id" : 8, "custId" : 104, "ordDate" : ISODate("2021-11-20T00:00:00Z"), "price" : 75 }
{ "_id" : 9, "custId" : 104, "ordDate" : ISODate("2021-11-20T00:00:00Z"), "price" : 55 }
{ "_id" : 10, "custId" : 104, "ordDate" : ISODate("2021-11-23T00:00:00Z"), "price" : 25 }


> var mapFun = function(){emit(this.custId, this.price);}

> var reduceFun = function(keycustId, valuesPrices){return Array.sum(valuesPrices);}

> db.Orders.mapReduce(mapFun, reduceFun, {out:"Result"})
{ "result" : "Result", "ok" : 1 }

> db.Result.find()
{ "_id" : 102, "value" : 125 }
{ "_id" : 104, "value" : 155 }
```

```
{ "_id" : 103, "value" : 60 }
{ "_id" : 101, "value" : 95 }


> db.Orders.mapReduce(
   function(){
      emit(this.custId, this.price);
   },
   function(key, values){
      return Array.avg(values)
   },
   {query:   {
      ordDate:{$gte:new Date("2021-11-19")
      }
   },
   out:"Result2"
   })

{ "result" : "Result2", "ok" : 1 }

> db.Result2.find()
{ "_id" : 102, "value" : 50 }
{ "_id" : 104, "value" : 51.666666666666664 }
{ "_id" : 103, "value" : 30 }



> db.Orders.mapReduce(function(){emit(this.custId, this.price);}, function(key, values){return
Array.avg(values)}, {query:{custId:{$gt:102}}, out:"Result2"})
{ "result" : "Result2", "ok" : 1 }
> db.Result2.find()
{ "_id" : 104, "value" : 51.666666666666664 }
{ "_id" : 103, "value" : 30 }


> db.Orders.mapReduce(function(){emit(this.custId, this.price);}, function(key, values){return
Array.avg(values)}, {query:{custId:{$eq:102}}, out:"Result2"})
{ "result" : "Result2", "ok" : 1 }
> db.Result2.find()
{ "_id" : 102, "value" : 41.666666666666664 }
```