**COMP1835**

**Lab 8. Working with MongoDB database**

**MapReduce**

**Overview:**

In this lab you will be practicing mapReduce in MongoDB.

**Lab 7.1. Enter data into MongoDB**

In order to perform mapReduce we need some data in your database.

Open mongo shell and create collection called **orders** in your **test** database.

(Hopefully you remember how to do it, if not refer to your previous week MongoDB lab)

Now Insert the following 5 documentsinto your collection **orders** using **db.orders.insert:**

**1. {**

**\_id: 1,**

**cust\_id: "abc123",**

**ord\_date: new Date("Oct 04, 2012"),**

**status: 'A',**

**price: 25,**

**items: [ { sku: "mmm", qty: 5, price: 2.5 },**

**{ sku: "nnn", qty: 5, price: 2.5 } ]**

**}**

**2. {**

**\_id: 2,**

**cust\_id: "abc234",**

**ord\_date: new Date("Oct 10, 2015"),**

**status: 'A',**

**price: 50,**

**items: [ { sku: "mmm", qty: 10, price: 2.5 },**

**{ sku: "nnn", qty: 10, price: 2.5 } ]**

**}**

**3. {**

**\_id: 3,**

**cust\_id: "abc777",**

**ord\_date: new Date("Oct 12, 2015"),**

**status: 'B',**

**price: 50,**

**items: [ { sku: "mmm", qty: 10, price: 2.5 },**

**{ sku: "nnn", qty: 10, price: 2.5 } ]**

**}**

**4. {**

**\_id: 4,**

**cust\_id: "abc234",**

**ord\_date: new Date("Feb 12, 2015"),**

**status: 'B',**

**price: 20,**

**items: [ { sku: "mmm", qty: 1, price: 2.5 },**

**{ sku: "nnn", qty: 1, price: 2.5 },**

**{ sku: "ddd", qty: 1, price: 5 },**

**{ sku: "xxx", qty: 1, price: 10 }]**

**}**

**5. {**

**\_id: 5,**

**cust\_id: "xyz345",**

**ord\_date: new Date("Feb 10, 2016"),**

**status: 'A',**

**price: 40,**

**items: [ { sku: "ddd", qty: 2, price: 5 },**

**{ sku: "xxx", qty: 3, price: 10 }]**

**}**

**Lab 7.2**. **Perform mapReduce and return the number of orders per customer**

In order to perform a **mapReduce** function on our **orders** collection, we would need to group documents on the basis of customer id and then count the number of orders by each customer.

The syntax for **mapReduc**e would be:

db.orders.**mapReduce**

(

**mapFunction**, // *which would need to be defined*

**reduceFunction**, // *which would need to be defined*

{out: "map\_reduce\_result" } //*outputs the results to a collection*

*named* map\_reduce\_result

)

1. First, we define the map function to process each input document:
   * The function doesn’t have arguments.
   * In the function, **this**refers to the document that the map-reduce operation is processing.
   * 1 means we are going to take each customer once.

**mapFunction** = **function()**

**{**

**emit(this.cust\_id,1);**

**}**

1. Second, we define the corresponding reduce function with two arguments **keyCustId**and **numOrders:**
   * The **numOrders** is an array whose elements are the orders values emitted by the map function and grouped by k**eyCustId**.
   * The function reduces the **numOrders** array to the sum of its elements.

**reduceFunction** = **function(keyCustId, numOrders)**

**{**

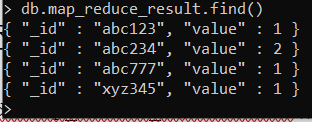
**return Array.sum(numOrders);**

**}**

1. Now we are ready to perform the map-reduce on all documents in the **orders** collection using our **mapFunction** and **reduceFunction**, by replacing the code for **mapFuntion** and **reduceFunction** accordingly.

Run this command in mongo shell and it should complete successfully.

The results of this operation will be in the collection named **map\_reduce\_result.**



**Lab 7.3**. **Perform mapReduce and return the total price per customer**

You need to perform the map-reduce operation on the **orders** collection to group by the **cust\_id**, and calculate the sum of the **price** for each **cust\_id**:

The syntax for mapReduce would be:

db.orders.**mapReduce**

(

**mapFunction**, // *which would need to be defined*

**reduceFunction**, // *which would need to be defined*

{out: "map\_reduce\_example" } //*outputs the results to a collection*

*named* map\_reduce\_example

)

1. First, we define the map function to process each input document:
   * The function doesn’t have arguments.
   * In the function, **this** refers to the document that the map-reduce operation is processing.
   * The function maps the price to the **cust\_id** for each document and emits the **cust\_id** and **price** pair.
2. Second, we define the corresponding reduce function with two arguments **keyCustId** and **valuesPrices:**
   * The **valuesPrices** is an array whose elements are the **price** values emitted by the map function and grouped by **keyCustId.**
   * The function reduces the **valuesPrice**array to the sum of its elements.
   * The function should return

{

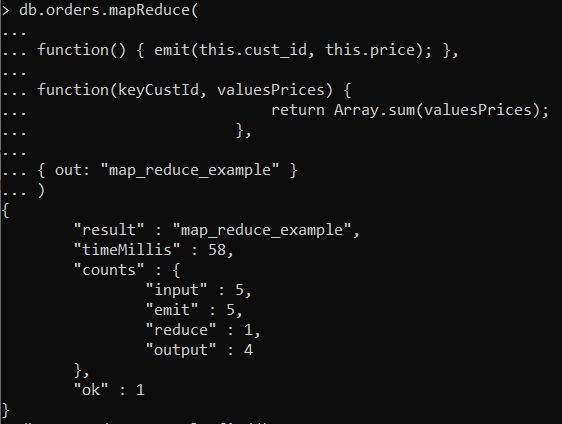
return Array.sum(valuesPrices);

}

1. Now we are ready to perform the map-reduce on all documents in the **orders** collection using our **mapFunction** and **reduceFunction**, by replacing the code for **mapFuntion** and r**educeFunction** accordingly:

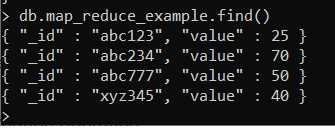
Run this command in mongo shell and it should complete successfully.

The results of this operation will be in the collection named **map\_reduce\_example.**



If the **map\_reduce\_example** collection already exists, the operation will replace the contents with the results of this map-reduce operation.

In order to see the results in mongo shell, you need to query this collection with



### Lab 7.4. Calculate the total quantity ordered of each item.

### Now we have to group all items from all orders by their item code (sku) and then calculate the total in each group. In this case only map function needs to change.

### Since items is an Array, we would need to loop through it using forEach function and for each item collect its quantity (qty):

### function() {

### this.items.forEach(function(item){ emit(item.sku, item.qty); });

### }

### The reduce function is still the same, counting number of elements in each bucket.

### Put your map and reduce functions together into mapReduce and run this in mongo shell and then query the results:

### 

### You can look at your collection to check if the results are correct.

### Optional Lab 7.5. Calculate order and total Quantity with average quantity per Item

Now you perform a map-reduce operation on the **orders** collection for all documents that have an **ord\_date** value greater than **01/01/2012**. The operation groups by the **item.sku** field, and calculates the number of orders and the total quantity ordered for each **sku**. The operation concludes by calculating the average quantity per order for each **sku**value:

1. Map Reduce should look like this:

db.orders.mapReduce

(

**mapFunction2**,

**reduceFunction2,**

{

out: {merge: "map\_reduce\_example"},

query: {ord\_date: {$gt: new Date('01/01/2012') }

},

finalize: **finalizeFunction2**

}

)

This operation uses the **query** field to select only those documents with **ord\_date** greater than **new Date(01/01/2012).** Then it outputs the results to a collection **map\_reduce\_example**.

If the **map\_reduce\_example** collection already exists, the operation will **merge** the existing contents with the results of this map-reduce operation.

1. Now we need to define map function, reduce function and finalize function:
2. Define the map function to process each input document:
   * In the function, **this** refers to the document that the map-reduce operation is processing.
   * For each item, the function associates the **sku** with a new object **value**that contains the **count** of **1** and the item **qty** for the order and emits the **sku** and **value** pair.

var **mapFunction2** = function()

{

for (var idx = 0; idx < **this**.items.length; idx++)

{ var **key** = **this**.items[idx].sku;

var **value** = {count: 1,qty: this.items[idx].qty};

emit(key, **value**);}

};

1. Now we define the corresponding reduce function with two arguments **keySKU** and **countObjVals**:
   * **countObjVals** is an array whose elements are the objects mapped to the grouped **keySKU** values passed by map function to the reducer function.
   * The function reduces the **countObjVals** array to a single object **reducedValue** that contains the **count** and the **qty** fields.
   * In **reducedVal**, the **count**field contains the sum of the **count** fields from the individual array elements, and the **qty** field contains the sum of the **qty** fields from the individual array elements.

var r**educeFunction2** = function(**keySKU**, **countObjVals**)

{

var reducedVal = {count: 0, qty: 0};

for (var idx = 0; idx < countObjVals.length; idx++)

{

reducedVal.count += countObjVals[idx].count;

reducedVal.qty += countObjVals[idx].qty;

}

**return reducedVal**;

};

1. Finally, we define a **finalize** function with two arguments **key** and **reducedVal**. The function modifies the **reducedVal** object to add a computed field named **avg** and returns the modified object:

var **finalizeFunction2** = function(key, reducedVal)

{

reducedVal.avg = reducedVal.qty/reducedVal.count;

return reducedVal;

};

1. Now we ready to assemble the map-reduce operation on the **orders** collection using the **mapFunction2**, **reduceFunction2**, and **finalizeFunction2** functions.

db.orders.mapReduce

(

**mapFunction2**,

**reduceFunction2,**

{

out: {merge: "map\_reduce\_example"},

query: {ord\_date: {$gt: new Date('01/01/2012') }

},

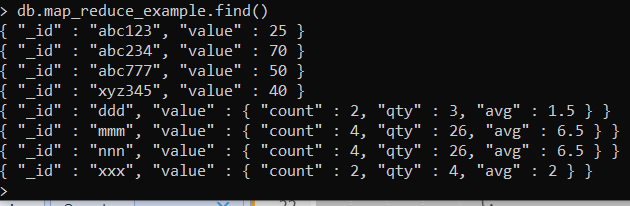
finalize: **finalizeFunction2**

}

)

After mapReduce completes, in order to see the results, you would need to query the map\_reduce\_example collection, by using:

db.map\_reduce\_example.find()



If you do not want to merge results from previous mapReduce you need to change the code by providing **a new name** for your result collection, for example, **map\_reduce\_example2**:

db.orders.mapReduce

(

**mapFunction2**,

**reduceFunction2,**

{

**out: "map\_reduce\_example2",**

query: {ord\_date: {$gt: new Date('01/01/2012') }

},

finalize: **finalizeFunction2**

}

)

Run the code for all functions and mapReduce again with updated mapReduce method and then query collection **map\_reduce\_example2:**

