# MES College of Engineering Pune-01

## Department of Computer Engineering

| | |
|---|---|
| **Name of Student:** | **Class:** |
| **Semester/Year:** | **Roll No:** |
| **Date of Performance:** | **Date of Submission:** |
| **Examined By:** | **Experiment No: Part B-01** |

**GROUP: B) ASSIGNMENT NO: 01**

**AIM: MongoDB Queries:**

Design and Develop MongoDB Queries using CRUD operations.(Use CRUD operations, SAVE method, logical operators etc.).

**OBJECTIVES:**

- To develop basic, intermediate and advanced Database programming skills.
- To develop basic Database administration skill.
- To Study NoSQL database.
- To Study document oriented database-Mongodb.

**APPRATUS:**

- Operating System recommended: 64-bit Open source Linux or its derivative.
- Front End: Java/PHP/Python
- Back End: MongoDB

**THEORY:**

**1. What is MongoDB?**

- MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.
- A record in MongoDB is a document, which is a data structure composed of field and value pairs.
- MongoDB documents are BSON documents. BSON is a binary representation of JSON with additional type information.
- A document is the basic unit of data for MongoDB and is roughly equivalent to a row in a relational database management system.

- A collection is a group of documents. If a document is the MongoDB analog of a row in a relational database, then a collection are analog to a table.
- MongoDB is type-sensitive and case-sensitive.
- Every document has a special key, "_id", that is unique within a collection.
- In the documents, the value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents.

```
{
        Name: “Sagar”,
        Class: “TE Comp”,
        College: “MESCOE”,
        Age: 26,
        Subject: [“DMSA”, “OSD”, “FCA”, “DCWSN”, “TOC”]
}
```

**Fig: MongoDB document**

**2. MongoDB Create Database**

- **The use Command**
  - ✓ MongoDB use DATABASE_NAME is used to create database. The command will create a new database; if it doesn't exist otherwise it will return the existing database.

**Syntax:** >use DATABASE_NAME

**Example:**
```
>use mydb
switched to db mydb
```

- ✓ To check your currently selected database uses the command db.

```
>db
mydb
```

- ✓ If you want to check your databases list, then use the command show dbs.

```
>show dbs
local    0.78125GB
test     0.23012GB
```

- ✓ Your created database (mydb) is not present in list. To display database you need to insert atleast one document into it.
- ✓ In mongodb default database is test. If you didn't create any database then collections will be stored in test database.

3. **MongoDB Drop Database**

- **The dropDatabase () Method**

  MongoDB db.dropDatabase () command is used to drop a existing database.

  **Syntax:** db.dropDatabase()

  This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

4. **MongoDB Create Collection**

- **The createCollection() Method**

  MongoDB db.createCollection(name, options) is used to create collection. In the command, name is name of collection to be created. Options are a document and used to specify configuration of collection. Options parameter is optional, so you need to specify only name of the collection.

  **Syntax:** db.createCollection(name, options)

  **Examples:** Basic syntax of createCollection() method without options is as follows:

```
>use test
switched to db test
>db.createCollection("mycollection")
{ "ok" : 1 }
```

5. **MongoDB Drop Collection**

- **The drop() Method**

  MongoDB's db.collection.drop() is used to drop a collection from the database.

  **Syntax:** db.COLLECTION_NAME.drop()

6. **MongoDB - Insert Document**

- **The insert() Method**

  To insert data into MongoDB collection, you need to use MongoDB's insert() , update() or save () method.

  **Syntax:** db.COLLECTION_NAME.insert(document)

7. **MongoDB - Query Document**

- **The find() Method**

  To query data from MongoDB collection, you need to use MongoDB's find() method. Find() method will display all the documents in a non-structured way.

  **Syntax:** db.COLLECTION_NAME.find()

- **The pretty() Method**

  To display the results in a formatted way, you can use pretty() method.

**Syntax:** db.COLLECTION_NAME.find().pretty()

- **The forEach() Method**

  To display the results in a JSON format, you can use forEach() method.

  **Syntax:** db.COLLECTION_NAME.find().forEach(printjson)

8. **MongoDB Update Document**

   MongoDB's update() and save() methods are used to update document into a collection. The update() method update values in the existing document while the save() method replaces the existing document with the document passed in save() method.

- **MongoDB Update() method**

  The update() method updates values in the existing document.

  **Syntax:**

  db.COLLECTION_NAME.update(SELECTIOIN_CRITERIA,UPDATED_DATA)

- **MongoDB Save() Method**

  The save() method replaces the existing document with the new document passed in save() method.

  **Syntax:** db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})

9. **MongoDB Delete Document**

- **The remove() Method**

  MongoDB's remove() method is used to remove document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag

  1. Deletion criteria: (Optional) deletion criteria according to documents will be removed.
  2. justOne: (Optional) if set to true or 1, then remove only one document.

  Syntax: db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)

**IMPLEMENTATION:**

A. Create Empdb database
B. Create Employee collection by considering following Fields:
   i. Empid: Number
   ii. Name: Embedded Doc (FName, LName)
   iii. Company Name: String
   iv. Salary: Number
   v. Designation: String

vi. Age: Number
vii. Expertise: Array
viii. DOB: String or Date
ix. Email id: String
x. Contact: String
xi. Address: Array of Embedded Doc (PAddr, LAddr)

C. Insert at least 10 documents in Employee Collection and execute following statements:

1. Select all documents where the Designation field has the value "Programmer" and the value of the salary field is greater than 30000.
2. Creates a new document if no document in the employee collection contains
3. {Designation: "Tester", Company_name: "TCS", Age: 25}
4. Selects all documents in the collection where the field age has a value less than 30 or the value of the salary field is greater than 40000.
5. Matches all documents where the value of the field Address is an embedded document that contains only the field city with the value "Pune" and the field Pin_code with the value "411001".
6. Finds all documents with Company_name: "TCS" and modifies their salary field by 2000.
7. Find documents where Designation is not equal to "Developer".
8. Find _id, Designation, Address and Name from all documents where Company_name is "Infosys".
9. Selects all documents in the employee collection where the value of the Designation is either "Developer" or "Tester".
10. Find all document with Exact Match on an Array having Expertise: ['Mongodb','Mysql', 'Cassandra']
11. Drop Single documents where designation="Developer"

**CONCLUSION:**

**QUESTIONS:**

1. What is NoSQL and enlist its benefits.
2. Shows the relationship of RDBMS terminology with MongoDB.

3. Explain CRUD operations in MongoDB database with suitable Example
4. What are Advantages of MongoDB over RDBMS?
5. Enlist Basic datatypes of MongoDB.
6. What is different between SAVE and UPDATE method.
7. What is ObjectId in Mongodb?
8. Explain different method to insert document in Mongodb.
9. Explain CAP & BASE Theorem in NoSQL with Suitable Example.
10. What are different key feature of MongoDB.

**MES College of Engineering Pune-01**

**Department of Computer Engineering**

| | |
|---|---|
| **Name of Student:** | **Class:** |
| **Semester/Year:** | **Roll No:** |
| **Date of Performance:** | **Date of Submission:** |
| **Examined By:** | **Experiment No: Part B-02** |

**GROUP: B) ASSIGNMENT NO: 02**

**AIM: MongoDB – Aggregation and Indexing:**

Design and Develop MongoDB Queries using aggregation and indexing with suitable example using MongoDB.

**OBJECTIVES:**

- To develop basic, intermediate and advanced Database programming skills.
- To develop basic Database administration skill.
- To learn aggregation and indexing for NoSQL database.

**APPRATUS:**

- Operating System recommended: 64-bit Open source Linux or its derivative.
- Front End: Java/PHP/Python.
- Back End: MongoDB.

**THEORY:**

**A. MongoDB Aggregation**

- Aggregations operations process data records and return computed results.
- Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.
- In SQL count(*) and with group by is an equivalent of mongodb aggregation.
- Running data aggregation on the mongod instance simplifies application code and limits resource requirements.
- Like queries, aggregation operations in MongoDB use collections of documents as an input and return results in the form of one or more documents.

- MongoDB provides three ways to perform aggregation: the aggregation pipeline, the map-reduce function and single purpose aggregation methods and commands.

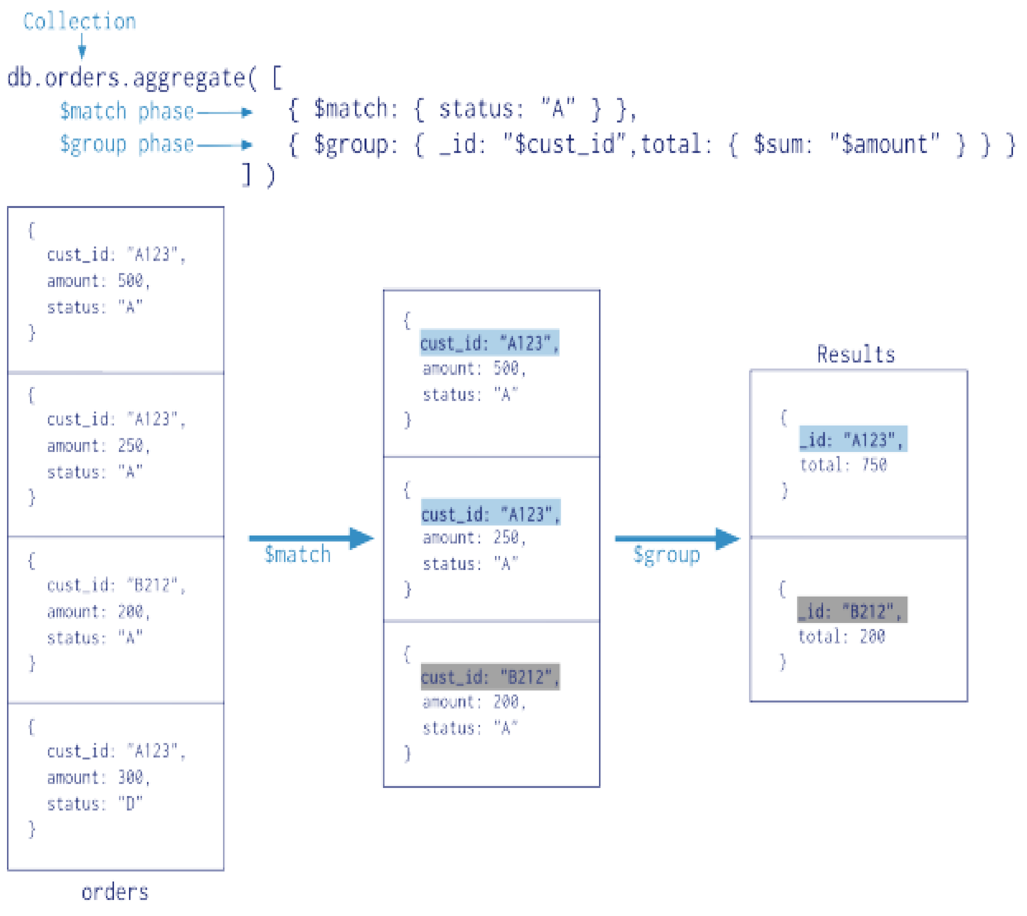
## 1. The aggregate() Method

For the aggregation in mongodb you should use **aggregate()** method.

**Syntax:** db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)

## 2. Aggregation Pipeline

- The aggregation pipeline is a framework for data aggregation modeled on the concept of data processing pipelines.
- The aggregation pipeline provides an alternative to map-reduce and may be the preferred solution for many aggregation tasks where the complexity of map-reduce may be unwarranted.
- In UNIX command shell pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on.
- There is a set of possible stages and each of those is taken a set of documents as an input and is producing a resulting set of documents (or the final resulting JSON document at the end of the pipeline). This can then in turn again be used for the next stage and so on.



```
db.orders.aggregate( [
    { $match: { status: "A" } },
    { $group: { _id: "$cust_id",total: { $sum: "$amount" } } }
] )
```

Collection
$match phase
$group phase
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
{ cust_id: "A123", amount: 300, status: "D" }
orders
$match
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
$group
Results
{ _id: "A123", total: 750 }
{ _id: "B212", total: 200 }

3. **Pipeline Operators**
   - **$project:** Reshapes a document stream. $project can rename, add, or remove fields as well as create computed values and sub-documents.
   - **$match:** Filters the document stream, and only allows matching documents to pass into the next pipeline stage. $match uses standard MongoDB queries.
   - **$redact:** Restricts the content of a returned document on a per-field level.
   - **$limit:** Restricts the number of documents in an aggregation pipeline.
   - **$skip:** Skips over a specified number of documents from the pipeline and returns the rest.
   - **$unwind:** Takes an array of documents and returns them as a stream of documents.
   - **$group:** Groups documents together for the purpose of calculating aggregate values based on a collection of documents.
   - **$sort:** Takes all input documents and returns them in a stream of sorted documents.
   - **$geoNear:** Returns an ordered stream of documents based on proximity to a geospatial point.
   - **$out:** Writes documents from the pipeline to a collection. The $out operator must be the last stage in the pipeline.
4. **Expression Operators**
   a) **$group Operators**
      - **$addToSet:** Returns an array of all the unique values for the selected field among for each document in that group.
      - **$first:** Returns the first value in a group.
      - **$last:** Returns the last value in a group.
      - **$max:** Returns the highest value in a group.
      - **$min:** Returns the lowest value in a group.
      - **$avg:** Returns an average of all the values in a group.
      - **$push:** Returns an array of all values for the selected field among for each document in that group.
      - **$sum:** Returns the sum of all the values in a group.

   b) **Comparison Operators**
      - **$cmp:** Compares two values and returns the result of the comparison as an integer.
      - **$eq:** Takes two values and returns true if the values are equivalent.
      - **$gt:** Takes two values and returns true if the first is larger than the second.

- **$gte:** Takes two values and returns true if the first is larger than or equal to the second.
- **$lt:** Takes two values and returns true if the second value is larger than the first.
- **$lte:** Takes two values and returns true if the second value is larger than or equal to the first.
- **$ne:** Takes two values and returns true if the values are not equivalent.

**c) Boolean Operators**

- **$and:** Returns true only when all values in its input array are true.
- **$or:** Returns true when any value in its input array are true.
- **$not:** Returns the Boolean value that is the opposite of the input value.

**d) Arithmetic Operators**

- **$add:** Computes the sum of an array of numbers.
- **$divide:** Takes two numbers and divides the first number by the second.
- **$mod:** Takes two numbers and calculates the modulo of the first number divided by the second.
- **$multiply:** computes the product of an array of numbers.
- **$subtract:** Takes an array that contains two numbers or two dates and subtracts the second value from the first.

**e) Array Operators**

- **$size:** Returns the size of the array.

**f) Date Operators**

- **$dayOfYear:** Converts a date to a number between 1 and 366.
- **$dayOfMonth:** Converts a date to a number between 1 and 31.
- **$dayOfWeek:** Converts a date to a number between 1 and 7.
- **$year**:Converts a date to the full year.
- **$month:** Converts a date into a number between 1 and 12.
- **$week:** Converts a date into a number between 0 and 53
- **$hour:** Converts a date into a number between 0 and 23.
- **$minute:** Converts a date into a number between 0 and 59.
- **$second:** Converts a date into a number between 0 and 59. May be 60 to account for leap seconds.
- **$millisecond:** Returns the millisecond portion of a date as an integer between 0 and 999.

## B. MongoDB Indexing

- Indexes support the efficient resolution of queries.
- Indexes provide high performance read operations for frequently used queries.
- Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. This scan is highly inefficient and requires the mongod to process a large volume of data.
- Indexes are special data structures that store a small portion of the data set in an easy to traverse form.
- The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in index.
- Indexes support the efficient execution of queries in MongoDB.
- Without indexes, MongoDB must scan every document in a collection to select those documents that match the query statement.
- These collection scans are inefficient because they require mongod to process a larger volume of data than an index for each operation.

### 1. The ensureIndex() Method

- To create an index you need to use ensureIndex() method of mongodb.
- **Syntax:** db.COLLECTION_NAME.ensureIndex({KEY:1})
- Here key is the name of filed on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.
- **Example:**

```
db.mycol.ensureIndex({"title":1})
```

- In ensureIndex() method you can pass multiple fields, to create index on multiple fields.

```
db.mycol.ensureIndex({"title":1,"description":-1})
```

- The ensureIndex() method also accepts list of options (which are optional), whose list is given below:
  - ✓ **Background:** (type: Boolean) Builds the index in the background so that building an index does not block other database activities. Specify true to build in the background. The default value is **false**.
  - ✓ **Unique:** (type: Boolean) Creates a unique index so that the collection will not accept insertion of documents where the index key or keys match an existing value in the index. Specify true to create a unique index. The default value is **false**.
  - ✓ **Name:** (type: string) the name of the index. If unspecified, MongoDB generates an index name by concatenating the names of the indexed fields and the sort order.

- ✓ **dropDups** (type: Boolean) Creates a unique index on a field that may have duplicates. MongoDB indexes only the first occurrence of a key and removes all documents from the collection that contain subsequent occurrences of that key. Specify true to create unique index. The default value is **false**.
- ✓ **Sparse:** (type: Boolean) If true, the index only references documents with the specified field. These indexes use less space but behave differently in some situations (particularly sorts). The default value is **false**.
- ✓ **expireAfterSeconds:** (type: integer) Specifies a value, in seconds, as a TTL to control how long MongoDB retains documents in this collection.
- ✓ **v:** (type: index version) The index version number. The default index version depends on the version of mongod running when creating the index.
- ✓ **Weights:** (type: document) The weight is a number ranging from 1 to 99,999 and denotes the significance of the field relative to the other indexed fields in terms of the score.
- ✓ **default_language:** (type:  string) for a text index, the language that determines the list of stop words and the rules for the stemmer and tokenizer. The default value is **english**.
- ✓ **language_override:** (type:  string) for a text index, specify the name of the field in the document that contains, the language to override the default language. The default value is language.

**2. Unique Indexes**

- A unique index causes MongoDB to reject all documents that contain a duplicate value for the indexed field.
- To create a unique index, use the db.collection.ensureIndex() method with the unique option set to true.
- By default, unique is false on MongoDB indexes.

```
db.members.ensureIndex( { "user_id": 1 }, { unique: true } )
```

**3. Drop Duplicates**

Force MongoDB to create a unique index by deleting documents with duplicate values when building the index.

```
db.collection.ensureIndex( { a: 1 }, { unique: true, dropDups: true } )
```

**4. Remove Index**

- To remove an index from a collection use the dropIndex() method and the following procedure.

- Remove a Specific Index

  db.accounts.dropIndex( { “user_id": 1 } )
- Remove All Indexes except for the _id index from a collection

  db.collection.dropIndexes()

**5. Return a List of All Indexes**

- List all Indexes on a Collection
- To return a list of all indexes on a collection, use the db.collection.getIndexes() method.
- Example: To view all indexes on the user collection:

  db.user.getIndexes()
- List all Indexes for a Database
- To return a list of all indexes on all collections in a database:

  db.system.indexes.find()

**IMPLEMENTATION:**

A. Use Employee database created in Assignment B-01 and perform following aggregation operation

1. Return Designation with Total Salary is Above 200000
2. Find Employee with Total Salary for Each City with Designation="DBA"
3. Find Total Salary of Employee with Designation="DBA" for Each Company
4. Returns names and _id in upper case and in alphabetical order.
5. Count all records from collection
6. For each unique Designation, find avg Salary and output is sorted by AvgSal
7. Return separates value in the Expertise array where Name of Employee="Swapnil"
8. Return separates value in the Expertise array and return sum of each element of array
9. Return Array for Designation whose address is "Pune"
10. Return Max and Min Salary for each company.

B. Use Employee database created in Assignment B-01 and perform following indexing operation

1. To Create Single Field Indexes on Designation
2. To Create Compound Indexes on Name: 1, Age: -1
3. To Create Multikey Indexes on Expertise array
4. Return a List of All Indexes on Collection

5. Rebuild Indexes
6. Drop Index on Remove Specific Index
7. Remove All Indexes except for the _id index from a collection

**CONCLUSION:**

**QUESTIONS:**

1. Which are different aggregation commands and aggregation methods?
2. Enlist user-defined and system variables in aggregation.
3. Describe SQL to aggregation Mapping Chart.
4. Explain Indexing Methods in the mongo Shell.
5. What is different option for indexing?
6. Enlist different Pipeline Operators, Expression Operators, and Comparison Operators.
7. What is use of Drop Duplicates option in Indexing?
8. Write method to return a list of all indexes on a collection and databases.

**MES College of Engineering Pune-01**

**Department of Computer Engineering**

| | |
|---|---|
| **Name of Student:** | **Class:** |
| **Semester/Year:** | **Roll No:** |
| **Date of Performance:** | **Date of Submission:** |
| **Examined By:** | **Experiment No: Part B-03** |

## GROUP: B) ASSIGNMENT NO: 03

**AIM: MongoDB – Map-reduces operations:**

Implement Map reduces operation with suitable example using MongoDB.

**OBJECTIVES:**

- To develop basic, intermediate and advanced Database programming skills.
- To develop basic Database administration skill.
- To understand concepts of map reduce in aggregation of NoSQL database MongoDB.

**APPRATUS:**

- Operating System recommended: 64-bit Open source Linux or its derivative.
- Front End: Java/PHP/Python.
- Back End: MongoDB.

**THEORY:**

**1. MongoDB Aggregation**

- Aggregations are operations that process data records and return computed results.
- MongoDB provides a rich set of aggregation operations that examine and perform calculations on the data sets.
- Running data aggregation on the mongod instance simplifies application code and limits resource requirements.
- Like queries, aggregation operations in MongoDB use collections of documents as an input and return results in the form of one or more documents.

**2. Map-Reduce**

- Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results.
- For map-reduce operations, MongoDB provides the mapReduce database command.

**Syntax:** db.COLLECTION_NAME.mapReduce()

- In general, map-reduce operations have two phases: a map stage that processes each document and emits one or more objects for each input document, and reduce phase that combines the output of the map operation.
- Optionally, map-reduce can have a finalize stage to make final modifications to the result. Like other aggregation operations, map-reduce can specify a query condition to select the input documents as well as sort and limit the results.
- All map-reduce functions in MongoDB are JavaScript and run within the mongod process. Map-reduce operations take the documents of a single collection as the input and can perform any arbitrary sorting and limiting before beginning the map stage.
- MapReduce can return the results of a map-reduce operation as a document, or may write the results to collections. The input and the output collections may be sharded.

**3. MongoDB and MapReduce**

In MapReduce we use mapreduce, map, and reduce keys. These three keys are required, but there are many optional keys that can be passed to the MapReduce command:

- "*finalize" : function*
  - ✓ A final step to send reduce's output to.
- "*keeptemp" : boolean*
  - ✓ If the temporary result collection should be saved when the connection is closed.
- *"out" : string*
  - ✓ Name for the output collection. Setting this option implies keep temp : true.
- *"query" : document*
  - ✓ Query to filter documents by before sending to the map function.
- *"sort" : document*
  - ✓ Sort to use on documents before sending to the map (useful in conjunction with the limit option).
- *"limit" : integer*
  - ✓ Maximum number of documents to send to the map function.
- *"scope" : document*
  - ✓ Variables that can be used in any of the JavaScript code.
- *"verbose" : boolean*
  - ✓ Whether or not to use more verbose output in the server logs.

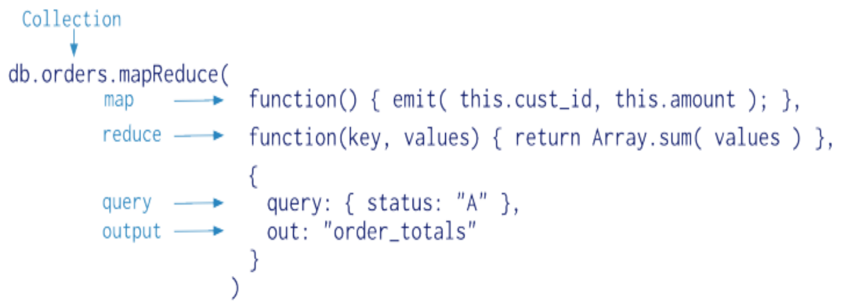
**4. Aggregation Commands**

- **Aggregate:** Performs aggregation tasks such as group using the aggregation framework.

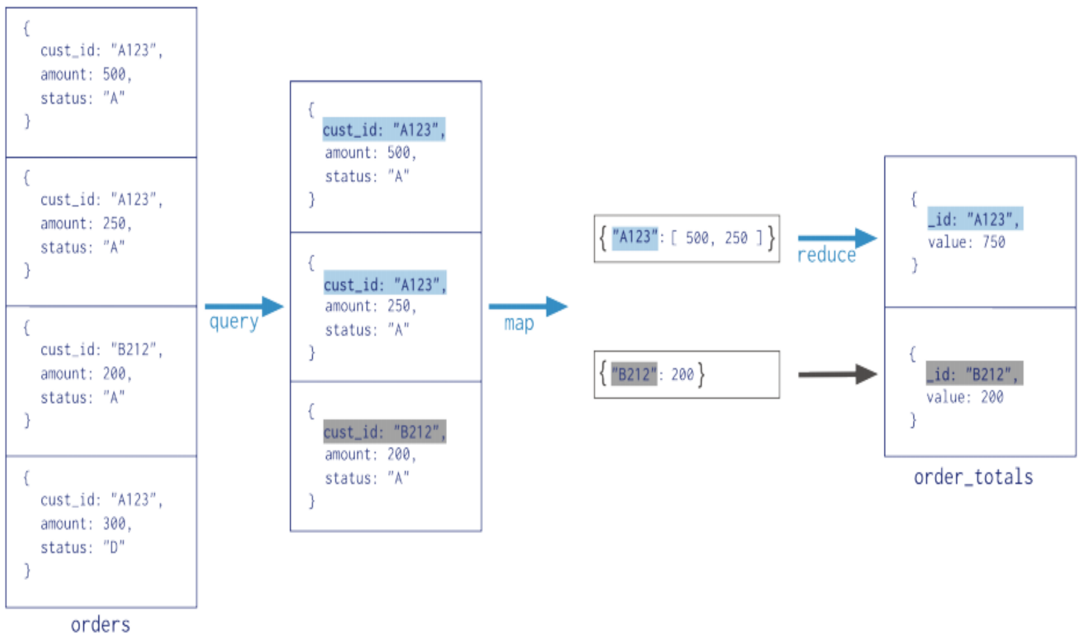
- **Count:** Counts the number of documents in a collection.
- **Distinct:** Displays the distinct values found for a specified key in a collection.
- **Group:** Groups documents in a collection by the specified key and performs simple aggregation.
- **mapReduce:** Performs map-reduce aggregation for large data sets.

5. **Aggregation Methods**

- **db.collection.aggregate():** Provides access to the aggregation pipeline.
- **db.collection.group():** Groups documents in a collection by the specified key and performs simple aggregation.
- **db.collection.mapReduce():** Performs map-reduce aggregation for large data sets.



```
db.orders.mapReduce(
             function() { emit( this.cust_id, this.amount ); },
             function(key, values) { return Array.sum( values ) },
             {
               query: { status: "A" },
               out: "order_totals"
             }
)
```



{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
{ cust_id: "A123", amount: 300, status: "D" }
orders
query
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
map
{ "A123": [ 500, 250 ] }
{ "B212": 200 }
reduce
{ _id: "A123", value: 750 }
{ _id: "B212", value: 200 }
order_totals

6. **Example for MapReduce Operation**

In the mongo shell, the **db.collection.mapReduce()** method is a wrapper around the mapReduce command.

The following examples use the **db.collection.mapReduce() method**:

Consider the following map-reduce operations on a collection orders that contains documents of the following prototype:

```
{
        _id: ObjectId("50a8240b927d5d8b5891743c"),
        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 } ]
}
```

**Return the Total Price Per Customer**

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:

a) **Define the map function to process each input document:**
   - 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.

```
var mapFunction1 = function() {
                        emit(this.cust_id, this.price);
                        };
```

b) **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.

```
var reduceFunction1 = function(keyCustId, valuesPrices) {
                                return Array.sum(valuesPrices);
                        };
```

c) **Perform the map-reduce on all documents in the orders collection using the mapFunction1 map function and the reduceFunction1 reduce function.**

```
db.orders.mapReduce(
                                mapFunction1,
                                reduceFunction1,
                                { out: "map_reduce_example" }
```

)

This operation outputs the results to a 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.

**IMPLEMENTATION:**

Use Employee database created in Assignment B-01 and perform Map reduces operation for following statements:

1. Return the Total Salary of per Company
2. Return the Total Salary of Company Name:"TCS"
3. Return the Avg Salary of Company whose address is “Pune".
4. Return the Total Salary for each Designation of Infosys.
5. Return total count for “State=AP”
6. Return Count for State AP and Age greater than 40.

**CONCLUSION:**

**QUESTIONS:**

1. What are different Aggregation commands?
2. What is map and reduce phase?
3. Explain Map Reduce Concurrency.
4. Write Step for MapReduce Operation with example.
5. What is Map-Reduce JavaScript Function?