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.

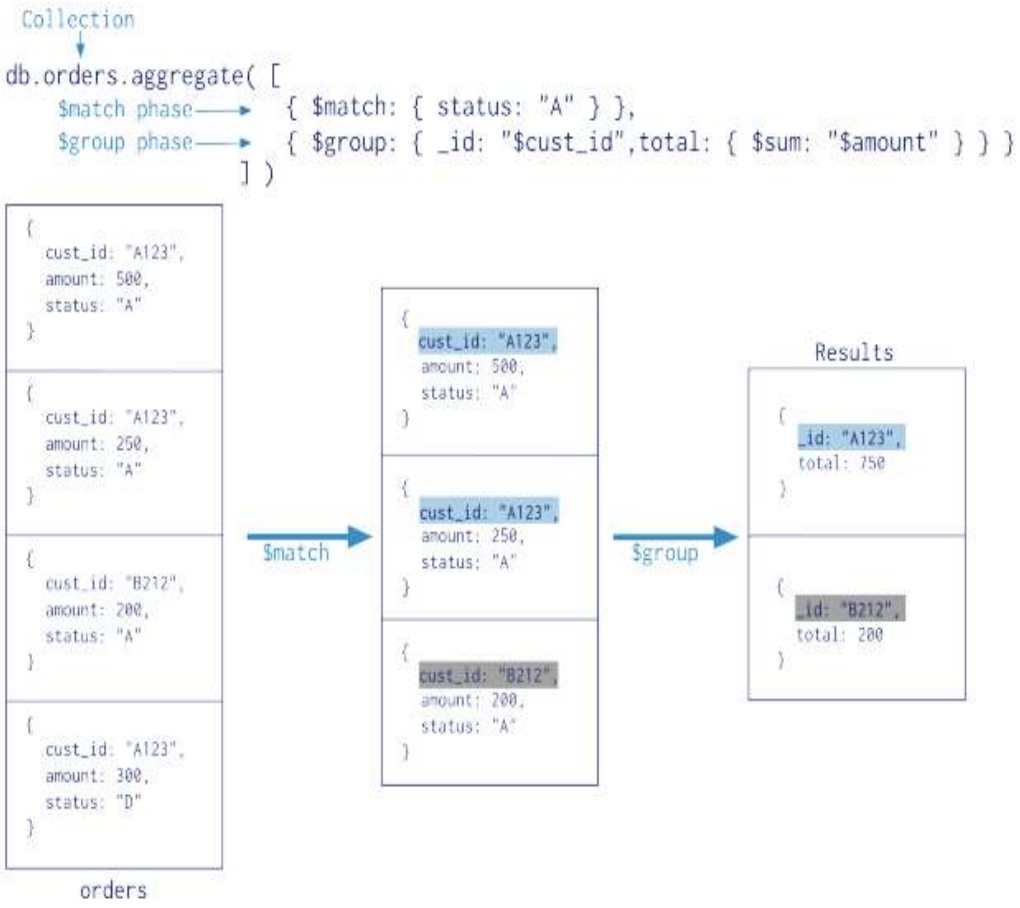
1. **The aggregate() Method**

   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.



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.