Assignment No. 11

Title of Assignment: Aggregation and indexing in MongoDB

Problem Definition: Aggregation and indexing with suitable example using MongoDB.

1.1 Pre-requisite : Basics of **MongoDB**

1.2 Learning Objective:

To learn all type aggregation and indexing commands and their uses.

1.3 Relevant Theory / Literature Survey:

1.3.1 Introduction

[I] 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.
- 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.

The aggregate() Method

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

Syntax:

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

Example:

```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
```

In the above example we have grouped documents by field **by_user** and on each occurance of by_user previous value of sum is incremented. There is a list available aggregation expressions .

Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])
$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

Pipeline Concept

- **$project:** Used to select some specific fields from a collection.
- **$match:** This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.
- **$group:** This does the actual aggregation as discussed above.
- **$sort:** Sorts the documents.
- **$skip:** With this it is possible to skip forward in the list of documents for a given amount of documents.
- **$limit:** This limits the amount of documents to look at by the given number starting from the current positions.
- **$unwind:** This is used to unwind document that are using arrays. when using an array the data is kind of prejoinded and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.

[II] MongoDB Indexing

- Indexes support the efficient resolution of 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 require 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.

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 dropIndex() Method

To drop an index you need to use dropIndex() method of mongodb.

Syntax:

>db.COLLECTION_NAME.dropIndex({KEY:1})

The getIndexes() Method

It is used to display the list of indexes

Syntax:

>db.COLLECTION_NAME.getIndexes({KEY:1})

Conclusion

Thus we studied the commands of Aggregation and indexing .