# Assignment No .10

**Title of Assignment: MongoDB – Aggregation and Indexing:**
Design and Develop MongoDB Queries using aggregation and indexing with suitable example using MongoDB.
**Course Objective:**
To acquire the skills to use a powerful, flexible, and scalable general-purpose databases to handle Big Data

**Course Outcome:**

Implement NoSQL queries using MongoDB

**Software Required: -** 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 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.

**The ensureIndex() Method**

To create an index you need to use ensureIndex() method of mongodb.

**SYNTAX:**

Basic syntax of **ensureIndex()** method is as follows()

```
>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})
```

**Drop Index :**

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

**AGGREGATION** :Aggregations operations process data records and return computed results. operations on the grouped data to return a single result. In sql count(*) and with group by is an equivalent of MongoDB

Now from the above collection if you want to display a list that how many tutorials are written by each user then you will use **aggregate()** method as shown below:

method.SYNTAX: Basic syntax of aggregate() method is as follows

```
{_id: ObjectId(7df78ad8902c) title: 'MongoDB Overview',
  description: 'MongoDB is no sql database', by_user:

  'tutorials point',

  url: 'http://www.tutorialspoint.com', tags:
  ['mongodb', 'database', 'NoSQL'], likes: 100},

{_id: ObjectId(7df78ad8902d) title: 'NoSQL Overview',
  description: 'No sql database is very fast', by_user:

  'tutorials point',

  url: 'http://www.tutorialspoint.com', tags:
  ['mongodb', 'database', 'NoSQL'], likes: 10},
```

```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
{"result" : [{"_id" : "tutorials point", "num_tutorial" : 2},
    {"_id" : "tutorials point", "num_tutorial" : 1}],"ok" : 1}
```

SQL equivalent query for the above use case will be **select by_user, count(*) from mycol group by by_user.** 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 expression.

| **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"}}}]) |
| $first | Gets the first document from the source documents according to the grouping. | db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}]) |
| $last | Gets the last document from the source documents according to the grouping. | db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}]) |

**Pipeline Concept**

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. MongoDB also support same concept in aggregation framework. 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 an so on.

Possible stages in aggregation framework are following:

- **$project:** Used to select some specific fields from a collection.
- **$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 position.s
- **$unwind:** This is used to unwind document that are using arrays. when using an array the data is kind of pre-joined 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.

**Conclusion:** We have implemented aggregation and indexing using Mongodb