Aim:
Aggregation and indexing with suitable example using MongoDB.

Theory:

Indexing:

Indexes support the efficient resolution of the queries.Without indexes MongoDB must scan every document of collection to select those documents that match the query statement.This scan is highly efficient and require the MongoDB to process a large volume of data.Indexes are special data structures that stores a small portion of data set in an easy way to transverse form. The index stores the value of a specific field or set of fields, ordered by the values of field as specified in index. The index needs to be created only once for collection. To create an index you need to use ensure Index() method of MongoDB.

Basic Syntax:

db. COLLECTION_NAME.ensureIndex({key : 1}).

Here key is the name of field on which you want to create index & 1 is or ascending order.
To create index in descending order you need to use -1. e.g.

db.mycol.ensureIndex({ “Title” })

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

db.mycol.ensureIndex({ “title” : 1 , “description” : 1})

Aggregation:

Aggregation operations process data records and return completed the results.Aggregation operations group values from multiple documents together, and can perform a variety of operations on grouped data to return a single digit. In SQL count(*) and with group by is an equivalent of MongoDB aggregation.MongoDB provides a number of aggregation tools that go beyond basic query Functionality.

1. These ranges from simply counting the number of documents in a collection to using. Mapreduce to do complex data analysis.

2. For the aggregation in MongoDB you should use aggregate() method.

Basic Syntax:
db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)

Different types of the aggregation operation are given below. i.e.

1. **$sum:**
   Sums up the defined value from all documents in the collection.

   e.g. db.mycol.aggregate([{ $group : { num tutorial : { $sum : “$likes” }}}])

2. **$avg:**
   Calculates the average of all given values from all documents in the collection. e.g. db.mycol.aggregate([{ $group : { new tutorial : { $arg : “$likes” }}}])

3. **$min:**
   Get the minimum of the corresponding values from all documents in the collection. e.g. db.mycol.aggregate([{ $group : { num_tutorial : { $min : “$likes” }}}])

4. **$max:**
   Gets the maximum of the corresponding values from all documents in the collection. e.g. db.aggregate([{ $group : { num_tutorial : { $max : “$likes”}}}])

5. **$first:**
   Gets the first document from the source documents according to the grouping. Typically this makes only sense together with somey applied “$sort” – stage. e.g. db.mycol.aggregate([{ $group : { -id : “$byuser”, First_url : { $first : “$url” }}}])

6. **$last:**
   Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort” – stage. e.g. db.mycol.aggregate([{ $group : {_id : $by user_last_url : {$cost : “$url”}}}])

   Conclusion:

   We have learnt the aggregation and indexing in MongoDB using suitable examples.