**Assignment No:7**

**Title:** Design and develop MongoDB queries using aggregation and indexing, with a suitable example.

**Problem Statement:** Execute aggregation operations and indexing efficiently within MongoDB, illustrated through relevant examples to showcase their practical application and benefits.

**Objective:** To gain knowledge of NoSQL databases for processing unstructured data.

**Outcome:** Use NoSQL databases for processing unstructured data.

**Tools Required:** Ubuntu OS, MongoDB.

**Theory:**

**1. 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 MongoDB 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 the index.

**createIndex() Method:** To create an index, you need to use createIndex() method of MongoDB.

|  |
| --- |
| **Syntax:**  db.Collection\_Name.createIndex({Key:1})  **Example:**  db.mycol.createIndex({"title":1}) |

**dropIndex() method:** You can drop a particular index using the dropIndex() method of MongoDB.

|  |
| --- |
| **Syntax:**  db.Collection\_Name.dropIndex({Key:1})  **Example:**  db.mycol. dropIndex ({"title":1}) |

**2. 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.

**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 : {$sum : "$likes"}}}])  db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$avg : "$likes"}}}]) |

Following is a list of 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"}}}]) |

**Conclusion:**

We have successfully implemented MongoDB aggregation and indexing operations.