Assignment No B2

Title: MongoDB – Aggregation and Indexing:
Design and Develop MongoDB Queries using aggregation and indexing with suitable example using MongoDB.

Objective: Understand aggregation and indexing operations in MongoDB

Theory:

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.
For aggregation in mongodb use aggregate() method.

Syntax:

>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)

aggregate() method

Expression	Description
$sum	Sums up the defined value from all documents in the collection.
$avg	Calculates the average of all given values from all documents in the collection.
$min	Gets the minimum of the corresponding values from all documents in the collection.
$max	Gets the maximum of the corresponding values from all documents in the collection.
$first	Gets the first document from the source documents according to the grouping.
$last	Gets the last document from the source documents according to the grouping.

Possible stages in aggregation

- $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 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.

Indexing: Indexes support the efficient execution of queries in MongoDB

Indexing Types

- **Single field index** only includes data from a single field of the Single Field Indexes documents in a collection.
- **Compound index** includes more than one field of the documents in Compound Indexes a collection.
- **Multikey index** is an index on an array field, adding an index key for Multikey each value in the array.
- **Geospatial indexes** support location-based searches. Geospatial Indexes and Queries Text Indexes
- **Text indexes** support search of string content in documents.
- **Hashed Index** -Hashed indexes maintain entries with hashes of the values of the indexed field and are used with sharded clusters to support hashed shard keys.

Index Properties:

Index Properties -The properties you can specify when building indexes.

1. TTL Indexes: The TTL index is used for TTL collections, which expire data after a period of time

2. Unique Indexes: A unique index causes MongoDB to reject all documents that contain a duplicate value for the indexed field.

3. Sparse Indexes: A sparse index does not index documents that do not have the indexed field.

Index Creation:

Syntax:

db.CollectionName.createIndex({ KeyName: 1 or -1})

- 1 for Ascending Sorting
- -1 for Descending Sorting

Index Creation Example:

- Single: db.stud.createIndex({ zipcode: 1})
- Compound: db.stud.createIndex({ dob: 1, zipcode: -1 })
- Unique: db.stud.createIndex({ rollno: 1 }, { unique: true })
- Sparse: db.stud.createIndex({ age: 1 }, { sparse: true })

Index Display

db.collection.getIndexes()

Returns an array that holds a list of documents that identify and describe the existing indexes on the collection.

Index Drop

Syntax:

- db.collection.dropIndex()
- db.collection.dropIndex(index)

Example:

- db.stud.dropIndex()
- db.stud.dropIndex({ “name" : 1 })

Indexing and Querying

- create an ascending index on the field name for a collection records:

  db.records.createIndex({ name: 1 })

- This index can support an ascending sort on name :

  db.records.find().sort({ name: 1 })

- The index can also support descending sort

  db.records.find().sort({ a: -1 })

- db.stud.findOne({rno:2}), using g index {rno:1}

Indexing with Unique:

- db.collectionname.ensureIndex ({x:1}, {unique:true})
- Don’t allow {_id:10,x:2} and {_id:11,x:2}
- Don’t allow {_id:12} and {_id:13} (both match {x:null}

Conclusion: Here we understood the concept of aggregation and indexing operations in MongoDB.