Name : Tejas V. Redkar
PRN : 10322210937
Roll No : PC-44 Panel-C
BDT- 2 BDT Roll No : 22

BDT Lab Assignment - 3

* Problem Statement :

To understand the aggregation pipeline & indexing in Mongo DB

* Objectives :

1) To learn concepts of aggregation pipeline
2) To learn how to create indexes in Mongo DB.

* Theory

Aggregation pipeline stages in MongoDB with pipeline diagram:

The aggregation pipeline in MongoDB is a framework for data transformation & analysis. It allows you to process & reshape data within a collection to generate more meaningful & useful results. The aggregation pipeline consists of several stages that are applied sequentially to the documents in a collection.

The stages in MongoDB aggregation pipeline are typically applied in the following order:

1) $match
2) $project
3) $group
4) $sort
5) $skip
6) $limit
7) $unwind
8) $lookup
9) $replaceRoot
10) $out

Diagram:

Collection
↓
db.orders.aggregate([
$match stage → { $match: {status: "A"}},
$group stage → { $group: {_id: "cust_id", total:
{ $sum: "$amount"
}}}

```
{
  cust_id : "A123",
  amount : 500
  status : "A"
}

{
  cust_id : "A123"
  amount: 250
  status: "A"
}

{
  cust_id : "B212"
  amount: 200
  status : "A"
}

{
  cust_id : "A123"
  amount : 300
  status : "D"
}
```

orders

$match

```
{
  cust_id : "A123"
  amount : 500,
  status : "A"
}

{
  cust_id : "A123"
  amount : 250,
  status : "A"
}

{
  cust_id : "B212",
  amount : 200,
  status : "A"
}
```

$group

Results

```
{
  _id : "A123"
  total : 750
}

{
  _id : "B212"
  total : 200
}
```

Use of indexing in MongoDB

Indexing is a crucial feature in MongoDB, as it helps improve the performance of database queries. It does so by allowing MongoDB to efficiently locate & retrieve specific documents within a collection.

Here are some key benefits of indexing in MongoDB:

1) Faster Query Performance
2) Sorting
3) Unique Constraints
4) Covered Queries
5) Text Search
6) Geospatial queries
7) Compound Indexes
8) Sparse Indexes
9) TTL Indexes
10) Partial Indexes
11) Hashed Indexes
12) Hinting
13) Aggregation Framework optimization

- State syntax of importing JSON into MongoDB as a collection

To import a ~~Fe~~ JSON file into MongoDB as a collection, you can use the 'mongoimport' command-line tool, which comes with MongoDB.

```
mongoimport --db your_database_name
            --collection your_collection_name
            --file your_file_name
```

Example: `mongoimport --db mydb --collection --file data.json`

* Platform: 64-bit open source Linux/Windows

* Conclusion: Hence, I learned to import JSON into MongoDB, create indexes & execute aggregation pipeline operation

* FAQs

Q1) Explain with syntax the concept of various indexes in MongoDB

Ans In MongoDB, you can create various types of indexes to optimize query performance, enforce uniqueness constraints, & support specific data access patterns. Here are some common index types, along with their syntax & use case:

1) Single Field Index:

Syntax:

```
db.collection.createIndex({field_name:1})
```

2) Compound Index:

Syntax:

```
db.collection.createIndex({field:1, field:-1})
```

3) Unique Index:

Syntax:

```
db.collection.createIndex({unique-field:1},
                          {unique:true})
```

4) Text Index:

Syntax:

```
db.collection.createIndex({text_field:"text"})
```

5) Geospatial Index

Syntax:

```
db.collection.createIndex({location:"2d"})
```

6) Sparse Index:

Syntax:

```
db.collection.createIndex({optional_field:1},
                          {sparse:true})
```

7) TTL (Time-To-Live) Index:

Syntax:

```
db.collection.createIndex({createdAt:1},{expire
                          AfterSeconds:3600})
```

8) Hashed Index

Syntax:

```
db.collection.createIndex({sharding_field:"hashed"})
```

9) Wildcard Index

Syntax:

```
db.collection.createIndex({"$**":1})
```

Q2) Explain all possible ways, along with their syntax, to import JSON/CSV files in MongoDB

Ans MongoDB provides various ways to import JSON & CSV into a database. Here are some common methods along with their syntax:

Importing JSON files:

1) Using 'mongoimport'

Syntax:

```
mongoimport --db your_database_name
            --collection your_collection_name
            --file your_file_name
```

2) Using 'insertMany' method (MongoShell)

Syntax:

```
db.collection_name.insertMany([json_object1,
                               json_object2,...])
```

Importing CSV files:

1) Using 'mongoimport' (Command Line)

Syntax:

```
mongoimport --db your_database_name
            --collection your_collection_name
            --type csv --headerline --file your_file.csv
```

Q3) Explain aggregation expressions with examples of their syntax.

Ans Aggregation expressions in MongoDB are operators used in the aggregation framework to perform various data transformations during the aggregation process. These expressions allow you to shape & manipulate data in meaningful ways. Here are some common aggregation expressions along with their examples of their syntax:

1) $match: Filters the documents in the pipeline to include only those that meet specified criteria.

Syntax:

```
{ $match: { field_name: criteria } }
```

2) $project: Reshape documents by including, excluding or renaming fields and can compute new fields based on expressions.

Syntax:

```
{ $project: { new_field_name: expression, _id: 0 } }
```

3) $group: Groups documents by a specified key & computes aggregate values using accumulator expressions.

Syntax:

{ $group : { _id : "$group-field", total : { $sum : "$value" } } }

4) $sort : Sorts the documents in the pipeline based on specified fields & sorting orders

Syntax:

{ $sort : { field_name : 1 } }

5) $limit : Limits the number of documents in the output. often used for pagination

Syntax:

{ $limit : 10 }

6) $skip : Skips a specified number of documents from the beginning of the result set. Also used for pagination.

Syntax:

{ $ skip : 5 }

7) $unwind : Deconstructs an array field, creating seperate documents for each array element

Syntax:

{ $unwind : "$Array_field" }

8) $lookup : Performs a left outer join between documents in the current collection & documents from another collection

Syntax:

```
{
  $lookup : {
    from : "other_collection",
    localfield : "local_field",
    foreign field : "foreign_field",
    as : "new_field".
  }
}
```

9) $addFields : Add new fields to documents

Syntax:

{$addFields : { new_field_name : expression}}

31/10/23