Group: - 1 from T4 Batch
0077 – Aakash Joshi
2039 – Akanksha Lokhande

Title-: MongoDB - Aggregation and Indexing:

Date of Completion-:

Objectives-:

- To develop Database programming skills.
- To develop basic Database administration skills.
- To develop skills to handle NoSQL database.
- To learn, understand and execute process of software application development.

Outcomes-:

- Design schema in appropriate normal form considering actual requirements.
- Implement SQL queries for given requirements , using different SQL concepts..
- Implement NoSQL queries using MongoDB.

Problem Statement-:

Design and Develop MongoDB Queries using aggregation and indexing with suitable example using MongoDB.

Software and Hardware requirement-:

- 64-bit Open source Linux or its derivative.
- Mango-DB

Theory-:

Aggregation Operations

Aggregation operations process multiple documents and return computed results. You can use aggregation operations to:

- Group values from multiple documents together.
- Perform operations on the grouped data to return a single result.
- Analyze data changes over time.

To perform aggregation operations, you can use:

- Aggregation pipelines, which are the preferred method for performing aggregations.
- Single purpose aggregation methods, which are simple but lack the capabilities of an aggregation pipeline.

Aggregation Pipelines

An aggregation pipeline consists of one or more stages that process documents:

- Each stage performs an operation on the input documents. For example, a stage can filter documents, group documents, and calculate values.

- The documents that are output from a stage are passed to the next stage.
- An aggregation pipeline can return results for groups of documents. For example, return the total, average, maximum, and minimum values.

Starting in MongoDB 4.2, you can update documents with an aggregation pipeline if you use the stages shown in Updates with Aggregation Pipeline.

Aggregation pipelines run with the `db.collection.aggregate()` method do not modify documents in a collection, unless the pipeline contains a `$merge` or `$out` stage.

Aggregation Pipeline Example

The following aggregation pipeline example contains two stages and returns the total order quantity of medium size pizzas grouped by pizza name:

```
db.orders.aggregate( [

// Stage 1: Filter pizza order documents by pizza
size
{


$match: { size: "medium" }
},

// Stage 2: Group remaining documents by pizza
name and calculate total quantity
{
$group: { _id: "$name", totalQuantity: { $sum:
"$quantity" } }
}

] )
```

The `$match` stage:

- Filters the pizza order documents to pizzas with a `size` of `medium`.
- Passes the remaining documents to the `$group` stage.

The `$group` stage:

- Groups the remaining documents by pizza `name`.
- Uses `$sum` to calculate the total order for each pizza name. The total is

quantity

stored in the totalQuantity field returned by the aggregation pipeline.

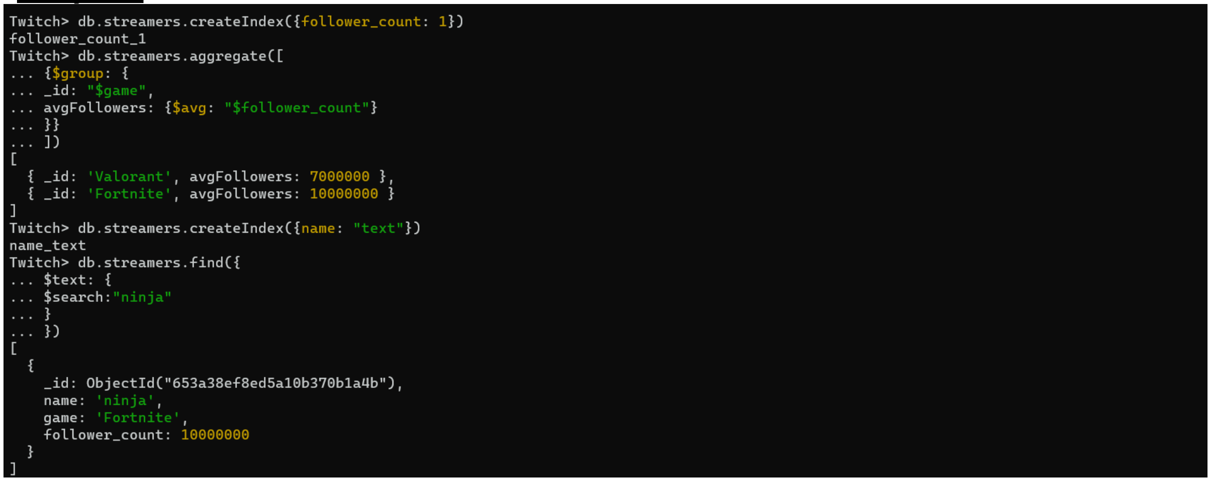
For runnable examples containing sample input documents, see Complete Aggregation Pipeline Examples.

Single Purpose Aggregation Methods

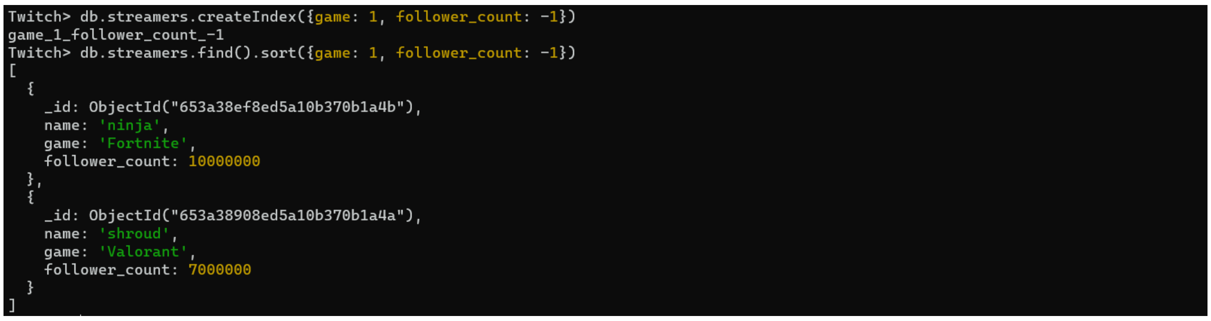
The single purpose aggregation methods aggregate documents from a single collection. The methods are simple but lack the capabilities of an aggregation pipeline.

Method	Description
db.collection.estimatedDocumentCount()	Returns an approximate count of the documents in a collection or a view.
db.collection.count()	Returns a count of the number of documents in a collection or a view.
db.collection.distinct()	Returns an array of documents that have distinct values for the specified field.

Output-:



```
Twitch> db.streamers.createIndex({follower_count: 1})
follower_count_1
Twitch> db.streamers.aggregate([
... {$group: {
... _id: "$game",
... avgFollowers: {$avg: "$follower_count"}
... }}
... ])
[
  { _id: 'Valorant', avgFollowers: 7000000 },
  { _id: 'Fortnite', avgFollowers: 10000000 }
]
Twitch> db.streamers.createIndex({name: "text"})
name_text
Twitch> db.streamers.find({
... $text: {
... $search:"ninja"
... }
... })
[
  {
    _id: ObjectId("653a38ef8ed5a10b370b1a4b"),
    name: 'ninja',
    game: 'Fortnite',
    follower_count: 10000000
  }
]
```



```
Twitch> db.streamers.createIndex({game: 1, follower_count: -1})
game_1_follower_count_-1
Twitch> db.streamers.find().sort({game: 1, follower_count: -1})
[
  {
    _id: ObjectId("653a38ef8ed5a10b370b1a4b"),
    name: 'ninja',
    game: 'Fortnite',
    follower_count: 10000000
  },
  {
    _id: ObjectId("653a38908ed5a10b370b1a4a"),
    name: 'shroud',
    game: 'Valorant',
    follower_count: 7000000
  }
]
```

Conclusion-:

After Completion of this Assignment I am able to implement the Mangodb aggression and indexing concepts.