Group B

Assignment No 11

Title of the Assignment: MongoDB – Aggregation and Indexing:

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

Objective of the Assignment: To understand the difference aggregation and indexing in MongoDB.

Outcome: Students will be able to learn and understand MongoDB Queries using aggregation and indexing

Theory:

MongoDB - Aggregation

Aggregations operations process data records and return computed results. Aggregation operations group valuesfrom 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.

Aggregation is a way of processing a large number of documents in a collection by means of passing them throughdifferent stages. The stages make up what is known as a pipeline. The stages in a pipeline can filter, sort, group, reshape and modify documents that pass through the pipeline.

One of the most common use cases of Aggregation is to calculate aggregate values for groups of documents. This issimilar to the basic aggregation available in SQL with the GROUP BY clause and COUNT, SUM and AVG functions. MongoDB Aggregation goes further though and can also perform relational-like joins, reshape documents, create new and update existing collections, and so on.

While there are other methods of obtaining aggregate data in MongoDB, the aggregation framework is therecommended approach for most work.

There are what are called single purpose methods like estimatedDocumentCount(), count(), and distinct() which areappended to a find() query making them quick to use but limited in scope.

Following is a list of available aggregation expression

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"}}}])

- **Program for aggregation:**

```
test> use customer switched to db customer
customer> db.createCollection('Customer');
{ ok: 1 }
customer> db.Customer.insertOne({'CustID':'A123', 'Amount':500,
'Status':'completed'})
{
acknowledged: true,
insertedId: ObjectId("635a9a5d29e81711afa59d0b")
}
customer> db.Customer.insertOne({'CustID':'A124', 'Amount':200,
'Status':'completed'})
{
acknowledged: true,
insertedId: ObjectId("635a9a7c29e81711afa59d0c")
}
customer> db.Customer.insertOne({'CustID':'A150', 'Amount':250,
'Status':'pending'})
{
acknowledged: true,
insertedId: ObjectId("635a9a9a29e81711afa59d0d")
}
customer> db.Customer.insertOne({'CustID':'A150', 'Amount':300,
'Status':'pending'})
{
acknowledged: true,
insertedId: ObjectId("635a9aa629e81711afa59d0e")
}
customer> db.Customer.insertOne({'CustID':'A201', 'Amount':750,
'Status':'pending'})
{
acknowledged: true,
insertedId: ObjectId("635a9aef29e81711afa59d0f")
}

customer> db.Customer.find().pretty()
[
{
_id: ObjectId("635a9a5d29e81711afa59d0b"),CustID: 'A123',
Amount: 500, Status: 'completed'
},
{
_id: ObjectId("635a9a7c29e81711afa59d0c"),CustID: 'A124',
Amount: 200, Status: 'completed'
},
{
_id: ObjectId("635a9a9a29e81711afa59d0d"),

CustID: 'A150',
```

```
Amount: 250, Status: 'pending'
},
{
_id: ObjectId("635a9aa629e81711afa59d0e"),CustID: 'A150',
Amount: 300, Status: 'pending'
},
{
_id: ObjectId("635a9aef29e81711afa59d0f"),CustID: 'A201',
Amount: 750, Status: 'pending'
}
]
```

customer>

1. Find the customer whose status is "completed"

```
customer> db.Customer.aggregate([{$match:{'Status':'completed'}}])
[
{
_id: ObjectId("635a9a5d29e81711afa59d0b"),CustID: 'A123',
Amount: 500, Status: 'completed'
},
{
_id: ObjectId("635a9a7c29e81711afa59d0c"),CustID: 'A124',
Amount: 200, Status: 'completed'
}
]
```

2. Find the customer whose status is "completed" and whose amount is more than or equal to 250

```
customer>
db.Customer.aggregate([{$match:{'Status':'complete
d','Amount':{$gte:250}}}])
[
{
_id: ObjectId("635a9a5d29e81711afa59d0b"),
CustID: 'A123',
Amount: 500,
Status: 'completed'
}
```

3. In above query use match operator twice to acheive same result

```
customer>
db.Customer.aggregate([{$match:{'Status':'completed'}},{$match:{'Amount':
{$gte:250}}}])
```

```
[
{
_id: ObjectId("635a9a5d29e81711afa59d0b"),CustID: 'A123',
Amount: 500,
Status: 'completed'
}
]
```

4. **Find sum of amount of customer under status "completed"**

```
customer>
db.Customer.aggregate([{$match:{'Status':'completed'}},{$group:{_
id:'$Cus tID','totalAmount':{$sum:'$Amount'}}}])
[
{ _id: 'A123', totalAmount: 500 },
{ _id: 'A124', totalAmount: 200 }
]
```

Find the minimum amount of customer under status "pending"

```
customer>
db.Customer.aggregate([{$match:{'Status':'pending'}},{$group:{_id
:'$CustI D','minAmount':{$max:'$Amount'}}}])
[ { _id: 'A150', minAmount: 300 }, { _id: 'A201', minAmount: 750
} ]
```

Find the maximum amount of customer under status "pending"

```
customer>
db.Customer.aggregate([{$match:{'Status':'pending'}},{$group:{_id
:'$CustI D','maxAmount':{$max:'$Amount'}}}]).pretty()
[ { _id: 'A150', maxAmount: 300 }, { _id: 'A201', maxAmount: 750
} ]
```

5. Find the average amount of customer under status "pending"

```
customer>
db.Customer.aggregate([{$match:{'Status':'pending'}},{$group:{_id
:'$CustI D','avgAmount':{$avg:'$Amount'}}}]).pretty()
[ { _id: 'A150', avgAmount: 275 }, { _id: 'A201', avgAmount: 750 }
]
```

Sort the customer according to CustID in ascending order

```
customer> db.Customer.aggregate([{$sort:{'CustID':1}}]).pretty()
 [
  { _id: ObjectId("635a9a5d29e81711afa59d0b"), CustID: 'A123',
  Amount: 500,          Status: 'completed'
},
{
_id: ObjectId("635a9a7c29e81711afa59d0c"),
CustID: 'A124',

Amount: 200,
Status: 'completed'
},
{
_id: ObjectId("635a9a9a29e81711afa59d0d"),
CustID: 'A150',
Amount: 250,
Status: 'pending'
},
{
_id: ObjectId("635a9aa629e81711afa59d0e"),
CustID: 'A150',
Amount: 300,
Status: 'pending'
},
{
_id: ObjectId("635a9aef29e81711afa59d0f"), CustID: 'A201',
Amount: 750,
 Status: 'pending'
}
]
```

6. Sort the customer according to CustID in descending order

```
customer> db.Customer.aggregate([{$sort:{'CustID':-1}}]).pretty()
[
{
_id: ObjectId("635a9aef29e81711afa59d0f"), CustID: 'A201',
Amount: 750, Status: 'pending'
},
{
```

```
_id: ObjectId("635a9a9a29e81711afa59d0d"), CustID: 'A150',
Amount: 250, Status: 'pending'
},
{
_id: ObjectId("635a9aa629e81711afa59d0e"),
CustID: 'A150',
Amount: 300, Status: 'pending'
},
{
_id: ObjectId("635a9a7c29e81711afa59d0c"), CustID: 'A124',
Amount: 200, Status: 'completed'
},
{
_id: ObjectId("635a9a5d29e81711afa59d0b"), CustID: 'A123',
Amount: 500,
Status: 'completed'
}
]
```

7. using skip operator find total amount of customer

```
customer>
db.Customer.aggregate([{$skip:1},{$group:{_id:'$CustID',
'totalAmount':{$sum:'$Amount'}}}])
[
{ _id: 'A124', totalAmount: 200 },
{ _id: 'A150', totalAmount: 550 },
{ _id: 'A201', totalAmount: 750 }
]
```

8. using limit operator find total amount of customer

```
customer>
db.Customer.aggregate([{$limit:2},{$group:{_id:'$CustID',
'totalAmount':{$sum:'$Amount'}}}])
[
{ _id: 'A123', totalAmount: 500 },
{ _id: 'A124', totalAmount: 200 }
]
```

9. using limit operator find first amount of customer

```
customer>
db.Customer.aggregate([{$limit:4},{$group:{_id:'$CustID',
'totalAmount':{$first:'$Amount'}}}])
[
{ _id: 'A123', totalAmount: 500 },
{ _id: 'A124', totalAmount: 200 },
{ _id: 'A150', totalAmount: 250 }
]
```

10. using limit operator find last amount of customer

```
customer>
db.Customer.aggregate([{$limit:4},{$group:{_id:'$CustID',
'totalAmount':{$last:'$Amount'}}}])
[
{ _id: 'A123', totalAmount: 500 },
{ _id: 'A124', totalAmount: 200 },
{ _id: 'A150', totalAmount: 300 },
]
```

❖ MongoDB – Indexing

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of acollection 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 indexstores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.

1. Ensure Index

```
customer>
db.Customer.ensureIndex({'CustID':1}
)[ 'CustID_1' ]
```

2.Find Indexes

```
customer>
db.Customer.getIndexes(
)[
```

```
{ v: 2, key: { _id: 1 }, name: '_id_' },
{ v: 2, key: { CustID: 1 }, name: 'CustID_1' }
]
```

3. Create Index

```
customer> db.Customer.createIndex({'Amount':-1})
Amount_-1
customer>
db.Customer.getIndexes(
)[
{ v: 2, key: { _id: 1 }, name: '_id_' },
{ v: 2, key: { CustID: 1 }, name: 'CustID_1' },
{ v: 2, key: { Amount: -1 }, name: 'Amount_-1' }
]
```

4. Drop Index

```
customer> db.Customer.dropIndex({'Amount':-1})
{ nIndexesWas: 3, ok: 1 } customer> db.Customer.getIndexes() [
{ v: 2, key: { _id: 1 }, name: '_id_' },
{ v: 2, key: { CustID: 1 }, name: 'CustID_1' }
]
```

5. sort indexes

a. Descending

```
customer>
db.Customer.find().sort({'CustID':-
1})[
{
_id: ObjectId("635a9aef29e81711afa59d0f"),
CustID: 'A201',
Amount: 750,
Status: 'pending'
},
{
_id: ObjectId("635a9aa629e81711afa59d0e"),
CustID: 'A150',
Amount: 300,
Status: 'pending'
```

```
},
{
_id: ObjectId("635a9a9a29e81711afa59d0d"),
CustID: 'A150',
Amount: 250,
Status: 'pending'
},
{
_id: ObjectId("635a9a7c29e81711afa59d0c"),

CustID: 'A124',
Amount: 200,
Status: 'completed'
},
{
_id: ObjectId("635a9a5d29e81711afa59d0b"),
CustID: 'A123',
Amount: 500,
Status: 'completed'
}
]
```

b. Ascending

```
customer>
db.Customer.find().sort({'CustID':1}
)[

{
_id: ObjectId("635a9a5d29e81711afa59d0b"),
CustID: 'A123',
Amount: 500,
Status: 'completed'
},
{
_id: ObjectId("635a9a7c29e81711afa59d0c"),
CustID: 'A124',
Amount: 200,
Status: 'completed'
},
{
_id: ObjectId("635a9a9a29e81711afa59d0d"),
```

```
CustID: 'A150',
Amount: 250,
Status: 'pending'
},
{
_id: ObjectId("635a9aa629e81711afa59d0e"),
CustID: 'A150',
Amount: 300,
Status: 'pending'
},
{
_id: ObjectId("635a9aef29e81711afa59d0f"),
CustID: 'A201',
Amount: 750,
Status: 'pending'
}
]
```

Note:
createIndex() used to create indexes on collections whereas ensureIndex() creates an index on the specified field ifthe index does not already exist.
Moreover, when we execute createIndex() twice the second execution will just fail whereas with ensureIndex() you can invoke it multiple times and it will not fail

Conclusion: Performed Aggregation and indexing operations in MongoDB.

Viva Question:

- What is Aggregation in MongoDB?
- What is indexing in MongoDB?
- List out the aggregation expression?
- What are the features of MongoDB?
- Write difference between RDBMS and MongoDB?

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator:	