Name :- Rushikesh Kazbhazi Palve
Roll No. 31258

Date :	Page No.
/ /20	

Assignment No. 10 [B2]

DOP : 17-11-2021 DOS : 30-11-2021

Title :- MongoDB : Aggregation and Indexing :

Problem Definition :-

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

Objectives :-

① Understand indexing and aggregation concept on MongoDB records.

Learning Outcomes :-

After completion of assignment, students will be able to
ⓘ Understand the concepts of aggregation and indexing on MongoDB records.
ⓘⓘ Design and develop MongoDB queries using aggregation and indexing.

Theory related to concepts :-

Scanned by TapScanner

MongoDB is an open-source document database and leading NoSQL database. MongoDB is written in C++. This tutorial will give great understanding on MongoDB concepts needed to create and deploy a highly scalable and performance-oriented database.

Aggregation :-

Aggregation 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.

Basic syntax of aggregate() method is as follows:

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

Example :-

In the collection you have the following data -

```
{
   _id : ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by_user : 'tutorials point',
```

Scanned by TapScanner

```
    url: 'http://www.tutorialspoint.com',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 100
},
{
    _id: ObjectId(7df78ad8902d)
    title: 'NoSQL Overview',
    description: 'No sql & database is very fast',
    by_user: 'tutorials point',
    url: 'http://www.tutorialspoint.com',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 10
},
{
    _id: ObjectId(7df78ad8902e)
    title: 'Neo4j Overview',
    description: 'Neo4j is no sql database',
    by_user: 'Neo4j',
    url: 'http://www.neo4j.com',
    tags: ['neo4j', 'database', 'NoSQL'],
    likes: 750
},
```

Now from the above collection, if you want to display a list stating how many tutorials are written by each user, then you will use the following aggregate() method.

```
> db.mycol.aggregate([{$group:{_id:"$by_user",
  num_tutorial:{$sum:1}}}])

{
   "result":[
```

Scanned by TapScanner

```
{
    "_id": 'tutorials point", "num_tutorial": 2
},
{
    "_id": "Neo4j", "num_tutorial": 1
}],
"ok": 1
}>
```

Sql equivalent query for the above use case will be selected by_user, count(*) from mycol group by by_user.

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"}}}])

Scanned by TapScanner

Expression	Description	Example
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group:{_id:"$by_user", num_tutorial:{$max:"$likes"}}}])
$push	Insert the value to an array in the resulting document.	db.mycol.aggregate([{$group:{_id:"$by_user", url:{$push:"$url"}}}])
$addToSet	Insect the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$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"}}}])

Scanned by TapScanner

Pipeline Concept :-

In UNIX command, shell pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on. MongoDB also supports same concept in aggregation framework. there is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents (or the final resulting JSON document at the end of the pipeline). This can then in turn be used for the next stage and so on.

Following are the possible stages in aggregation framework :

- **$project :** Used to select some specific fields from a collection.
- **$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.

Scanned by TapScanner

- $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 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 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.

The createIndex() method :-

MongoDB provides a createIndex() method to create one or more indexes on collections.

Scanned by TapScanner

Syntax :-

```
db.collection_name.createIndex(
        keys : { Field_name : 1/-1 },
        options : <document>,
        commitQuorum : <string or integer>
)
```

parameters :-

- The first parameter is a document that contains the field and value pairs when the field is the index key and the value describes the type of index for that field. For an ascending index on a field, specify the value 1 and for descending index, specify the value -1.
- Others are optional -

Optional parameters :-

- Options - It is a set of options that controls the creation of the index. The type of this parameter is document.
- CommitQuorum - It is the minimum number of data-bearing voting replica set members.

CONCLUSION :-

Thus, We have studied ~~and~~ use and implementation of aggregation function and indexing function.

Scanned by TapScanner

OUTPUT :-

1.) Create Database for Assignment No. 10.

```
> use assignment_no_10
switched to db assignment_no_10
```

2.) Create Student Collection. (Use of createCollection())

```
> db.createCollection("Student")
{ "ok" : 1 }
```

3.) Insert 10 records in 'Student' Collection. (Use of insert())

```
> db.Student.insert({_id:1, rollNo:101, regNo:100001, name:"Vidyut", dept:"Computer",
marks:[95, 90, 92, 91, 93]})
WriteResult({ "nInserted" : 1 })
> db.Student.insert({_id:2, rollNo:102, regNo:100002, name:"Pratap", dept:"IT",
marks:[92, 91, 92, 91, 90]})
WriteResult({ "nInserted" : 1 })
> db.Student.insert({_id:3, rollNo:103, regNo:100003, name:"Kailash", dept:"E&TC",
marks:[90, 98, 97, 96, 99]})
WriteResult({ "nInserted" : 1 })
> db.Student.insert({_id:4, rollNo:104, regNo:100004, name:"Mukund",
dept:"Mechanical", marks:[95, 94, 93, 90, 90]})
WriteResult({ "nInserted" : 1 })
> db.Student.insert({_id:5, rollNo:105, regNo:100005, name:"Girish", dept:"Civil",
marks:[92, 98, 94, 96, 93]})
WriteResult({ "nInserted" : 1 })
> db.Student.insert({_id:6, rollNo:106, regNo:100006, name:"Neeraj",
dept:"Electrical", marks:[98, 96, 94, 93, 92]})
WriteResult({ "nInserted" : 1 })
> db.Student.insert({_id:7, rollNo:107, regNo:100007, name:"Prashant", dept:"E&TC",
marks:[98, 99, 97, 93, 92]})
WriteResult({ "nInserted" : 1 })
> db.Student.insert({_id:8, rollNo:108, regNo:100008, name:"Raj", dept:"Computer",
marks:[90, 90, 90, 96, 92]})
WriteResult({ "nInserted" : 1 })
> db.Student.insert({_id:9, rollNo:109, regNo:100009, name:"Hari", dept:"IT",
marks:[91, 92, 93, 94, 95]})
WriteResult({ "nInserted" : 1 })
> db.Student.insert({_id:10, rollNo:110, regNo:100010,name:"Aditya",
dept:"Mechanical", marks:[99, 98, 97, 96, 95]})
WriteResult({ "nInserted" : 1 })
```

4.) Display All Records from 'Student' Collection. (Use of find())

```
> db.Student.find()

{ "_id" : 1, "rollNo" : 101, "regNo" : 100001, "name" : "Vidyut", "dept" : "Computer",
"marks" : [ 95, 90, 92, 91, 93 ] }
{ "_id" : 2, "rollNo" : 102, "regNo" : 100002, "name" : "Pratap", "dept" : "IT",
"marks" : [ 92, 91, 92, 91, 90 ] }
```

Scanned by TapScanner

```
{ "_id" : 3, "rollNo" : 103, "regNo" : 100003, "name" : "Kailash", "dept" : "E&TC",
"marks" : [ 90, 98, 97, 96, 99 ] }
{ "_id" : 4, "rollNo" : 104, "regNo" : 100004, "name" : "Mukund", "dept" :
"Mechanical", "marks" : [ 95, 94, 93, 90, 90 ] }
{ "_id" : 5, "rollNo" : 105, "regNo" : 100005, "name" : "Girish", "dept" : "Civil",
"marks" : [ 92, 98, 94, 96, 93 ] }
{ "_id" : 6, "rollNo" : 106, "regNo" : 100006, "name" : "Neeraj", "dept" :
"Electrical", "marks" : [ 98, 96, 94, 93, 92 ] }
{ "_id" : 7, "rollNo" : 107, "regNo" : 100007, "name" : "Prashant", "dept" : "E&TC",
"marks" : [ 98, 99, 97, 93, 92 ] }
{ "_id" : 8, "rollNo" : 108, "regNo" : 100008, "name" : "Raj", "dept" : "Computer",
"marks" : [ 90, 90, 90, 96, 92 ] }
{ "_id" : 9, "rollNo" : 109, "regNo" : 100009, "name" : "Hari", "dept" : "IT", "marks"
: [ 91, 92, 93, 94, 95 ] }
{ "_id" : 10, "rollNo" : 110, "regNo" : 100010, "name" : "Aditya", "dept" :
"Mechanical", "marks" : [ 99, 98, 97, 96, 95 ] }


A.] Indexing :-
================================================================================
=====================================

5.) Display 'executionStats' before creating Index on 'rollNo' field. (Use of
explain())

{
        "explainVersion" : "1",
        "queryPlanner" : {
                "namespace" : "assignment_no_10.Student",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "rollNo" : {
                                "$eq" : 9
                        }
                },
                "maxIndexedOrSolutionsReached" : false,
                "maxIndexedAndSolutionsReached" : false,
                "maxScansToExplodeReached" : false,
                "winningPlan" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                                "rollNo" : {
                                        "$eq" : 9
                                }
                        },
                        "direction" : "forward"
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 0,
                "executionTimeMillis" : 1,
                "totalKeysExamined" : 0,
                "totalDocsExamined" : 10,
                "executionStages" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
```

Scanned by TapScanner

```
                "rollNo" : {
                        "$eq" : 9
                }
        },
        "nReturned" : 0,
        "executionTimeMillisEstimate" : 0,
        "works" : 12,
        "advanced" : 0,
        "needTime" : 11,
        "needYield" : 0,
        "saveState" : 0,
        "restoreState" : 0,
        "isEOF" : 1,
        "direction" : "forward",
                "docsExamined" : 10
        }
},
"command" : {
        "find" : "Student",
        "filter" : {
                "rollNo" : 9
        },
        "$db" : "assignment_no_10"
},
"serverInfo" : {
        "host" : "RUSHI-BHAGU",
        "port" : 27017,
        "version" : "5.0.3",
        "gitVersion" : "657fea5a61a74d7a79df7aff8e4bcf0bc742b748"
        },
        "serverParameters" : {
                "internalQueryFacetBufferSizeBytes" : 104857600,
                "internalQueryFacetMaxOutputDocSizeBytes" : 104857600,
                "internalLookupStageIntermediateDocumentMaxSizeBytes" : 104857600,
                "internalDocumentSourceGroupMaxMemoryBytes" : 104857600,
                "internalQueryMaxBlockingSortMemoryUsageBytes" : 104857600,
                "internalQueryProhibitBlockingMergeOnMongoS" : 0,
                "internalQueryMaxAddToSetBytes" : 104857600,
                "internalDocumentSourceSetWindowFieldsMaxMemoryBytes" : 104857600
        },
        "ok" : 1
}
```


=== Single Field Index :- ===

6.) Create Single Field Index on 'rollNo' field. (Use of createIndex())

```
> db.Student.createIndex({rollNo:1})

{
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "createdCollectionAutomatically" : false,
        "ok" : 1
}
```

Scanned by TapScanner

7.) Display 'executionStats' after creating Index on 'rollNo' field. (Use of explain() and find())

```
> db.Student.explain("executionStats").find({rollNo:9})

{
        "explainVersion" : "1",
        "queryPlanner" : {
                "namespace" : "assignment_no_10.Student",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "rollNo" : {
                                "$eq" : 9
                        }
                },
                "maxIndexedOrSolutionsReached" : false,
                "maxIndexedAndSolutionsReached" : false,
                "maxScansToExplodeReached" : false,
                "winningPlan" : {
                        "stage" : "FETCH",
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "keyPattern" : {
                                        "rollNo" : 1
                                },
                                "indexName" : "rollNo_1",
                                "isMultiKey" : false,
                                "multiKeyPaths" : {
                                        "rollNo" : [ ]
                                },
                                "isUnique" : false,
                                "isSparse" : false,
                                "isPartial" : false,
                                "indexVersion" : 2,
                                "direction" : "forward",
                                "indexBounds" : {
                                        "rollNo" : [
                                                "[9.0, 9.0]"
                                        ]
                                }
                        }
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 0,
                "executionTimeMillis" : 0,
                "totalKeysExamined" : 0,
                "totalDocsExamined" : 0,
                "executionStages" : {
                        "stage" : "FETCH",
                        "nReturned" : 0,
                        "executionTimeMillisEstimate" : 0,
                        "works" : 1,
                        "advanced" : 0,
                        "needTime" : 0,
                        "needYield" : 0,
```

Scanned by TapScanner

```
"saveState" : 0,
"restoreState" : 0,
"isEOF" : 1,
"docsExamined" : 0,
"alreadyHasObj" : 0,
"inputStage" : {
        "stage" : "IXSCAN",
        "nReturned" : 0,
        "executionTimeMillisEstimate" : 0,
        "works" : 1,
        "advanced" : 0,
        "needTime" : 0,
        "needYield" : 0,
        "saveState" : 0,
        "restoreState" : 0,
        "isEOF" : 1,
        "keyPattern" : {
                "rollNo" : 1
        },
        "indexName" : "rollNo_1",
        "isMultiKey" : false,
        "multiKeyPaths" : {
                "rollNo" : [ ]
        },
        "isUnique" : false,
        "isSparse" : false,
        "isPartial" : false,
        "indexVersion" : 2,
        "direction" : "forward",
                "indexBounds" : {
                        "rollNo" : [
                                "[9.0, 9.0]"
                        ]
                },
                "keysExamined" : 0,
                "seeks" : 1,
                "dupsTested" : 0,
                "dupsDropped" : 0
        }
    }
},
"command" : {
        "find" : "Student",
        "filter" : {
                "rollNo" : 9
        },
        "$db" : "assignment_no_10"
},
"serverInfo" : {
        "host" : "RUSHI-BHAGU",
        "port" : 27017,
        "version" : "5.0.3",
        "gitVersion" : "657fea5a61a74d7a79df7aff8e4bcf0bc742b748"
},
"serverParameters" : {
        "internalQueryFacetBufferSizeBytes" : 104857600,
        "internalQueryFacetMaxOutputDocSizeBytes" : 104857600,
        "internalLookupStageIntermediateDocumentMaxSizeBytes" : 104857600,
```


Scanned by TapScanner

```
                "internalDocumentSourceGroupMaxMemoryBytes" : 104857600,
                "internalQueryMaxBlockingSortMemoryUsageBytes" : 104857600,
                "internalQueryProhibitBlockingMergeOnMongoS" : 0,
                "internalQueryMaxAddToSetBytes" : 104857600,
                "internalDocumentSourceSetWindowFieldsMaxMemoryBytes" : 104857600
        },
        "ok" : 1
}


=== Index Administration :-

8.) Use getIndexes() on 'Student' Collection.

> db.Student.getIndexes()

[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_"
        },
        {
                "v" : 2,
                "key" : {
                        "rollNo" : 1
                },
                "name" : "rollNo_1"
        }
]


=== Compound Index :- ===

9.) Create compound index on fields rollNo and name. (Use of createIndex())

> db.Student.createIndex({"rollNo":1, "name":1})

{
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "createdCollectionAutomatically" : false,
        "ok" : 1
}


=== Index Administration :-

10.) Use getIndexes() on 'Student' Collection.

> db.Student.getIndexes()

[
        {
                "v" : 2,
                "key" : {
```

Scanned by TapScanner

```
                "_id" : 1
        },
        "name" : "_id_"
},
{
        "v" : 2,
        "key" : {
                "rollNo" : 1
        },
        "name" : "rollNo_1"
},
{
        "v" : 2,
        "key" : {
                "rollNo" : 1,
                "name" : 1
        },
        "name" : "rollNo_1_name_1"
}
]
```


11.) Display 'executionStats' before creating Index on 'regNo' field. (Use of explain())

```
> db.Student.explain("executionStats").find({regNo:100007})

        "explainVersion" : "1",
        "queryPlanner" : {
        "namespace" : "assignment_no_10.Student",
        "indexFilterSet" : false,
        "parsedQuery" : {
                "regNo" : {
                        "$eq" : 100007
                }
        },
        "maxIndexedOrSolutionsReached" : false,
        "maxIndexedAndSolutionsReached" : false,
        "maxScansToExplodeReached" : false,
        "winningPlan" : {
                "stage" : "COLLSCAN",
                "filter" : {
                        "regNo" : {
                                "$eq" : 100007
                        }
                        },
                        "direction" : "forward"
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 1,
                "executionTimeMillis" : 0,
                "totalKeysExamined" : 0,
                "totalDocsExamined" : 10,
                "executionStages" : {
                        "stage" : "COLLSCAN",
```

Scanned by TapScanner

```
"filter" : {
        "regNo" : {
                "$eq" : 100007
        }
},
"nReturned" : 1,
"executionTimeMillisEstimate" : 0,
"works" : 12,
"advanced" : 1,
"needTime" : 10,
"needYield" : 0,
"saveState" : 0,
"restoreState" : 0,
"isEOF" : 1,
                "direction" : "forward",
                "docsExamined" : 10
        }
},
"command" : {
        "find" : "Student",
        "filter" : {
                "regNo" : 100007
        },
        "$db" : "assignment_no_10"
},
"serverInfo" : {
        "host" : "RUSHI-BHAGU",
        "port" : 27017,
        "version" : "5.0.3",
                "gitVersion" : "657fea5a61a74d7a79df7aff8e4bcf0bc742b748"
        },
        "serverParameters" : {
                "internalQueryFacetBufferSizeBytes" : 104857600,
                "internalQueryFacetMaxOutputDocSizeBytes" : 104857600,
                "internalLookupStageIntermediateDocumentMaxSizeBytes" : 104857600,
                "internalDocumentSourceGroupMaxMemoryBytes" : 104857600,
                "internalQueryMaxBlockingSortMemoryUsageBytes" : 104857600,
                "internalQueryProhibitBlockingMergeOnMongoS" : 0,
                "internalQueryMaxAddToSetBytes" : 104857600,
                "internalDocumentSourceSetWindowFieldsMaxMemoryBytes" : 104857600
        },
        "ok" : 1
}
```


```
=== Unique Index :- ===

12.) Create unique index on 'regNo' field. (Use of createIndex() and unique)

> db.Student.createIndex({"regNo":1}, {unique:true})
{
        "numIndexesBefore" : 3,
        "numIndexesAfter" : 4,
        "createdCollectionAutomatically" : false,
        "ok" : 1
}
```

Scanned by TapScanner

13.) Display 'executionStats' after creating Index on 'regNo' field. (Use of explain())

```
> db.Student.explain("executionStats").find({regNo:100007})

{
        "explainVersion" : "1",
        "queryPlanner" : {
                "namespace" : "assignment_no_10.Student",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "regNo" : {
                                "$eq" : 100007
                        }
                },
                "maxIndexedOrSolutionsReached" : false,
                "maxIndexedAndSolutionsReached" : false,
                "maxScansToExplodeReached" : false,
                "winningPlan" : {
                        "stage" : "FETCH",
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "keyPattern" : {
                                        "regNo" : 1
                                },
                                "indexName" : "regNo_1",
                                "isMultiKey" : false,
                                "multiKeyPaths" : {
                                        "regNo" : [ ]
                                },
                                "isUnique" : true,
                                "isSparse" : false,
                                "isPartial" : false,
                                "indexVersion" : 2,
                                "direction" : "forward",
                                "indexBounds" : {
                                        "regNo" : [
                                                "[100007.0, 100007.0]"
                                        ]
                                }
                        }
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 1,
                "executionTimeMillis" : 3,
                "totalKeysExamined" : 1,
                "totalDocsExamined" : 1,
                "executionStages" : {
                        "stage" : "FETCH",
                        "nReturned" : 1,
                        "executionTimeMillisEstimate" : 0,
                        "works" : 2,
                        "advanced" : 1,
                        "needTime" : 0,
```

Scanned by TapScanner

```
"needYield" : 0,
"saveState" : 0,
"restoreState" : 0,
"isEOF" : 1,
"docsExamined" : 1,
"alreadyHasObj" : 0,
"inputStage" : {
        "stage" : "IXSCAN",
        "nReturned" : 1,
        "executionTimeMillisEstimate" : 0,
        "works" : 2,
        "advanced" : 1,
        "needTime" : 0,
        "needYield" : 0,
        "saveState" : 0,
        "restoreState" : 0,
        "isEOF" : 1,
        "keyPattern" : {
                "regNo" : 1
        },
        "indexName" : "regNo_1",
        "isMultiKey" : false,
        "multiKeyPaths" : {
                "regNo" : [ ]
        },
        "isUnique" : true,
        "isSparse" : false,
        "isPartial" : false,
        "indexVersion" : 2,
                        "direction" : "forward",
                        "indexBounds" : {
                                "regNo" : [
                                        "[100007.0, 100007.0]"
                                ]
                        },
                        "keysExamined" : 1,
                        "seeks" : 1,
                        "dupsTested" : 0,
                        "dupsDropped" : 0
                }
        }
},
"command" : {
        "find" : "Student",
        "filter" : {
                "regNo" : 100007
        },
        "$db" : "assignment_no_10"
},
"serverInfo" : {
        "host" : "RUSHI-BHAGU",
        "port" : 27017,
        "version" : "5.0.3",
        "gitVersion" : "657fea5a61a74d7a79df7aff8e4bcf0bc742b748"
},
"serverParameters" : {
        "internalQueryFacetBufferSizeBytes" : 104857600,
        "internalQueryFacetMaxOutputDocSizeBytes" : 104857600,
```


Scanned by TapScanner

```
                "internalLookupStageIntermediateDocumentMaxSizeBytes" : 104857600,
                "internalDocumentSourceGroupMaxMemoryBytes" : 104857600,
                "internalQueryMaxBlockingSortMemoryUsageBytes" : 104857600,
                "internalQueryProhibitBlockingMergeOnMongoS" : 0,
                "internalQueryMaxAddToSetBytes" : 104857600,
                "internalDocumentSourceSetWindowFieldsMaxMemoryBytes" : 104857600
        },
        "ok" : 1
}



=== Index Administration :-

14.) Use getIndexes() on 'Student' Collection.

> db.Student.getIndexes()

[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_"
        },
        {
                "v" : 2,
                "key" : {
                        "rollNo" : 1
                },
                "name" : "rollNo_1"
        },
        {
                "v" : 2,
                "key" : {
                        "rollNo" : 1,
                        "name" : 1
                },
                "name" : "rollNo_1_name_1"
        },
        {
                "v" : 2,
                "key" : {
                        "regNo" : 1
                },
                "name" : "regNo_1",
                "unique" : true
        }
]


=== Aggregation Commands :- ===

15.) Count the total number of documents. (Use of count())

> db.Student.count()
10
```

Scanned by TapScanner

16.) Find all distinct roll numbers in 'Student' Collection.(Use of distinct())

```
> db.Student.distinct("rollNo")
[ 101, 102, 103, 104, 105, 106, 107, 108, 109, 110 ]
```

B.] Aggregation :-

==========

17.) Display the total marks of all students.

```
> db.Student.aggregate([{$unwind:"$marks"}, {$group:{"_id":"$rollNo",
"total_Score":{$sum:"$marks"}}}])

{ "_id" : 106, "total_Score" : 473 }
{ "_id" : 107, "total_Score" : 479 }
{ "_id" : 101, "total_Score" : 461 }
{ "_id" : 103, "total_Score" : 480 }
{ "_id" : 108, "total_Score" : 458 }
{ "_id" : 109, "total_Score" : 465 }
{ "_id" : 102, "total_Score" : 456 }
{ "_id" : 105, "total_Score" : 473 }
{ "_id" : 110, "total_Score" : 485 }
{ "_id" : 104, "total_Score" : 462 }
```

18.) Display the total marks of all students in decreasing order.

```
> db.Student.aggregate([{$unwind:"$marks"}, {$group:{"_id":"$rollNo",
"total_Score":{$sum:"$marks"}}}, {$sort:{"total_Score":-1}}])

{ "_id" : 110, "total_Score" : 485 }
{ "_id" : 103, "total_Score" : 480 }
{ "_id" : 107, "total_Score" : 479 }
{ "_id" : 106, "total_Score" : 473 }
{ "_id" : 105, "total_Score" : 473 }
{ "_id" : 109, "total_Score" : 465 }
{ "_id" : 104, "total_Score" : 462 }
{ "_id" : 101, "total_Score" : 461 }
{ "_id" : 108, "total_Score" : 458 }
{ "_id" : 102, "total_Score" : 456 }
```

19.) Display the total marks of 'Computer' department students in decreasing order.

```
> db.Student.aggregate([{$unwind:"$marks"}, {$match:{dept:"Computer"}},
{$group:{"_id":"$rollNo", "total_Score":{$sum:"$marks"}}}, {$sort:{"total_Score":-
1}}])

{ "_id" : 101, "total_Score" : 461 }
{ "_id" : 108, "total_Score" : 458 }
```

20.) Find the highest scorer in all departments.

Scanned by TapScanner

```
> db.Student.aggregate([{$unwind:"$marks"}, {$group:{"_id":"$rollNo",
"total_Score":{$sum:"$marks"}}}, {$sort:{"total_Score":-1}}, {$limit:1}])
{ "_id" : 110, "total_Score" : 485 }
```

21.) Find the highest scorer in 'Computer' department.

```
> db.Student.aggregate([{$unwind:"$marks"}, {$match:{dept:"Computer"}},
{$group:{"_id":"$rollNo", "total_Score":{$sum:"$marks"}}}, {$sort:{"total_Score":-1}},
{$limit:1}])
{ "_id" : 101, "total_Score" : 461 }
```

22.) Find the highest scorer in 'IT' department.

```
> db.Student.aggregate([{$unwind:"$marks"}, {$match:{dept:"IT"}},
{$group:{"_id":"$rollNo", "total_Score":{$sum:"$marks"}}}, {$sort:{"total_Score":-1}},
{$limit:1}])
{ "_id" : 109, "total_Score" : 465 }
```

23.) Find the highest scorer in 'IT' department.

```
> db.Student.aggregate([{$unwind:"$marks"}, {$match:{dept:"E&TC"}},
{$group:{"_id":"$rollNo", "total_Score":{$sum:"$marks"}}}, {$sort:{"total_Score":-1}},
{$limit:1}])
{ "_id" : 103, "total_Score" : 480 }
```

24.) Find the highest scorer in 'Civil' department.

```
> db.Student.aggregate([{$unwind:"$marks"}, {$match:{dept:"Civil"}},
{$group:{"_id":"$rollNo", "total_Score":{$sum:"$marks"}}}, {$sort:{"total_Score":-1}},
{$limit:1}])
{ "_id" : 105, "total_Score" : 473 }
```

25.) Find the highest scorer in 'Mechanical' department.

```
> db.Student.aggregate([{$unwind:"$marks"}, {$match:{dept:"Mechanical"}},
{$group:{"_id":"$rollNo", "total_Score":{$sum:"$marks"}}}, {$sort:{"total_Score":-1}},
{$limit:1}])
{ "_id" : 110, "total_Score" : 485 }
```

Scanned by TapScanner