Problem Stataement: Study of Open Source NOSQL Database: MongoDB (Installation, BasicCRUD operations, Execution)

---Creating and Inserting a Data in new Database

> **show dbs**

EventDB 0.014GB

StudentLoginDB 0.000GB

admin 0.000GB

blogDB 0.000GB

config 0.000GB

fruitsDB 0.000GB

local 0.000GB

shopDB 0.000GB

todolistDB 0.000GB

userDB 0.000GB

> **use FruitDB**

switched to db FruitDB

> **db**

FruitDB

>

> **db.createCollection("Product")**

{ "ok" : 1 }

> **show collections**

Product

> **db.Product.drop()**

true

> **show collections**

> **db.createCollection("Product")**

{ "ok" : 1 }

> **db.Product.insert**({_id:1, name:"Apple"})

WriteResult({ "nInserted" : 1 })

> db.Product.insert({_id:2, name:"Mango"})

WriteResult({ "nInserted" : 1 })

> db.Product**.find()**

{ "_id" : 1, "name" : "Apple" }

{ "_id" : 2, "name" : "Mango" }

> db.Product.**update**({_id:1}, {$set:{name:"Leamon"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> **db.Product.find()**

{ "_id" : 1, "name" : "Leamon" }

{ "_id" : 2, "name" : "Mango" }

> db.Product.**remove**({_id:1})

WriteResult({ "nRemoved" : 1 })

> db.Product.**find**()

{ "_id" : 2, "name" : "Mango" }

> ^C

Bye

Problem Statement: Design and Develop MongoDB Queries using CRUD operations. (Use CRUDoperations, SAVE method, logical operators).

--Create Collection

use StudentLoginDB

switched to db StudentLoginDB

> db.createCollection("StudentInfo")

{ "ok" : 1 }

> show collections

StudentInfo

--Insert

> db.StudentInfo.insert({_id:1, name:"Aishu "})

WriteResult({ "nInserted" : 1 })

> db.StudentInfo.insert({_id:2, name:"Priya "})

WriteResult({ "nInserted" : 1 })

> db.StudentInfo.insert({_id:3, name:"Snehal"})

WriteResult({ "nInserted" : 1 })

> db.StudentInfo.insert({_id:4, name:"Pritam"})

WriteResult({ "nInserted" : 1 })

--Find

> db.StudentInfo.find()

{ "_id" : 1, "name" : "Aishu " }

{ "_id" : 2, "name" : "Priya " }

{ "_id" : 3, "name" : "Snehal" }

{ "_id" : 4, "name" : "Pritam" }

--Update

> db.StudentInfo.update({_id:3}, {$set:{name:"Sita "}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.StudentInfo.find()

{ "_id" : 1, "name" : "Aishu " }

{ "_id" : 2, "name" : "Priya " }

{ "_id" : 3, "name" : "Sita " }

{ "_id" : 4, "name" : "Pritam" }

--Save

> db.StudentInfo.save({ "_id" : 4, "name" : "Gita" })

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.StudentInfo.find()

{ "_id" : 1, "name" : "Aishu " }

{ "_id" : 2, "name" : "Priya " }

{ "_id" : 3, "name" : "Sita " }

{ "_id" : 4, "name" : "Gita" }

--Delete

> db.StudentInfo.remove({ "_id" : 4 })

WriteResult({ "nRemoved" : 0 })

> db.StudentInfo.find()

{ "_id" : 1, "name" : "Aishu " }

{ "_id" : 2, "name" : "Priya " }

{ "_id" : 3, "name" : "Sita " }

LOGICAL OPERATORS:

AND:

> db.StudentInfo.find({ "_id" : 1, "name" : "Aishu " }).pretty()

{ "_id" : 1, "name" : "Aishu " }

OR

> db. StudentInfo.find({$or:[{ "_id" : 1},{"name" : "Aishu "}]}).pretty()

{ "_id" : 1, "name" : "Aishu " }

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

```
> use StudentInfoDB
switched to db StudentInfoDB

> db.Student.find()
{ "_id" : 1, "name" : "Aishu ", "Marks" : 90 }
{ "_id" : 3, "name" : "Snehal", "Marks" : 93 }
{ "_id" : 4, "name" : "Pritam", "Marks" : 80 }
{ "_id" : 2, "name" : "Priya ", "Marks" : 68 }
```

Matching Documents ($match):

```
> db.Student.aggregate([
...     {
...         $match: { Marks: { $gte: 90 } }
...     }
... ])
{ "_id" : 1, "name" : "Aishu ", "Marks" : 90 }
{ "_id" : 3, "name" : "Snehal", "Marks" : 93 }
```

Grouping Documents ($group):

```
> db.Student.aggregate([
...     {
...         $group: {
...             _id: null,
...             totalMarks: { $sum: "$Marks" },
...             avgMarks: { $avg: "$Marks" },
...             minMarks: { $min: "$Marks" },
...             maxMarks: { $max: "$Marks" }
...         }
...     }
... ])
{ "_id" : null, "totalMarks" : 331, "avgMarks" : 82.75, "minMarks" : 68, "maxMarks" : 93 }
```

Counting the Number of Documents in a Group:

```
> db.Student.aggregate([
...     {
...         $group: {
```

```
...             _id: "$name",
...             count: { $sum: 1 }
...         }
...     }
... ])
{ "_id" : "Pritam", "count" : 1 }
{ "_id" : "Priya ", "count" : 1 }
{ "_id" : "Snehal", "count" : 1 }
{ "_id" : "Aishu ", "count" : 1 }
```

Sorting the Results ($sort):

```
> db.Student.aggregate([
...     {
...         $sort: { Marks: -1 }
...     }
... ])
{ "_id" : 3, "name" : "Snehal", "Marks" : 93 }
{ "_id" : 1, "name" : "Aishu ", "Marks" : 90 }
{ "_id" : 4, "name" : "Pritam", "Marks" : 80 }
{ "_id" : 2, "name" : "Priya ", "Marks" : 68 }
```

Limiting the Number of Results ($limit):

```
> db.Student.aggregate([
...     {
...         $limit: 3
...     }
... ])
{ "_id" : 1, "name" : "Aishu ", "Marks" : 90 }
{ "_id" : 3, "name" : "Snehal", "Marks" : 93 }
{ "_id" : 4, "name" : "Pritam", "Marks" : 80 }
```

$first and $last Operators:

```
> db.Student.aggregate([
...     {
...         $sort: { Marks: 1 } // Sort by marks in ascending order
...     },
...     {
...         $group: {
```

```
...             _id: null,
...             highestMarksStudent: { $last: "$name" },
...             highestMarks: { $last: "$Marks" },
...             lowestMarksStudent: { $first: "$name" },
...             lowestMarks: { $first: "$Marks" }
...         }
...     }
... ])
{ "_id" : null, "highestMarksStudent" : "Snehal", "highestMarks" : 93, "lowestMarksStudent" : "Priya ", "lowestMarks" : 68 }
```

$skip:

```
> db.Student.aggregate([
...     {
...         $skip: 2
...     }
... ])
{ "_id" : 4, "name" : "Pritam", "Marks" : 80 }
{ "_id" : 2, "name" : "Priya ", "Marks" : 68 }
```

Problem Statement: MongoDB – Map-reduces operations: Implement Map reduces operation with suitable example using MongoDB.

Calculate the Average Marks Using Map-Reduce:

```
> var mapFunction = function() {
...     emit('average', this.Marks);
... };
>
> var reduceFunction = function(key, values) {
...     return Array.sum(values) / values.length;
... };
>
> db.Student.mapReduce(
...     mapFunction,
...     reduceFunction,
...     { out: { inline: 1 } }
... )
{
        "results" : [
                {
                        "_id" : "average",
                        "value" : 82.75
                }
        ],
        "ok" : 1
}
```

Calculate the Total Marks Using Map-Reduce:

```
> var mapFunction = function() {
...     emit('total', this.Marks);
... };
>
> var reduceFunction = function(key, values) {
...     return Array.sum(values);
... };
>
```

```
> db.Student.mapReduce(
...     mapFunction,
...     reduceFunction,
...     { out: { inline: 1 } }
... )
{ "results" : [ { "_id" : "total", "value" : 331 } ], "ok" : 1 }
```

Calculating Average Marks per Student Name Using Map-Reduce:

```
> var mapFunction = function() {
...     emit(this.name, this.Marks);
... };
>
> var reduceFunction = function(key, values) {
...     return Array.sum(values) / values.length;
... };
>
> db.Student.mapReduce(
...     mapFunction,
...     reduceFunction,
...     { out: { inline: 1 } }
... )
{
        "results" : [
                {
                        "_id" : "Priya ",
                        "value" : 68
                },
                {
                        "_id" : "Pritam",
                        "value" : 80
                },
                {
                        "_id" : "Aishu ",
                        "value" : 90
                },
```

```
        {
            "_id" : "Snehal",
            "value" : 93
        }
    ],
    "ok" : 1
}
```