M1. Design and Develop MongoDB Queries using CRUD operations:

Create Employee collection by considering following Fields:

i. Name: Embedded Doc (FName, LName)

ii. Company Name: String

iii. Salary: Number

iv. Designation: String

v. Age: Number

vi. Expertise: Array

vii. DOB: String or Date

viii. Email id: String

ix. Contact: String

x. Address: Array of Embedded Doc (PAddr, LAddr)

Insert at least 5 documents in collection by considering above

attribute and execute following queries:

1. Select all documents where the Designation field has the value

"Programmer" and the value of the salary field is greater than

30000.

2. Creates a new document if no document in the employee collection

contains

{Designation: "Tester", Company\_name: "TCS", Age: 25}

3. Increase salary of each Employee working with “Infosys" 10000.

4. Finds all employees working with "TCS" and reduce their salary

by 5000.

5. Return documents where Designation is not equal to "Tester".

6. Find all employee with Exact Match on an Array having Expertise:

['Mongodb','Mysql','Cassandra']

db.Employee.insertMany([

{

Name: { FName: "John", LName: "Doe" },

CompanyName: "TCS",

Salary: 35000,

Designation: "Programmer",

Age: 28,

Expertise: ["Mongodb", "Node.js", "React"],

DOB: new Date("1995-04-10"),

Email: "john.doe@example.com",

Contact: "1234567890",

Address: [{ PAddr: "123 Main St", LAddr: "456 Elm St" }]

},

{

Name: { FName: "Jane", LName: "Smith" },

CompanyName: "Infosys",

Salary: 45000,

Designation: "Programmer",

Age: 32,

Expertise: ["Java", "C++", "Mongodb"],

DOB: new Date("1991-07-20"),

Email: "jane.smith@example.com",

Contact: "0987654321",

Address: [{ PAddr: "789 Oak St", LAddr: "101 Pine St" }]

},

{

Name: { FName: "Bob", LName: "Brown" },

CompanyName: "TCS",

Salary: 28000,

Designation: "Tester",

Age: 25,

Expertise: ["Testing", "Automation", "Selenium"],

DOB: new Date("1998-03-15"),

Email: "bob.brown@example.com",

Contact: "1231231234",

Address: [{ PAddr: "111 Cedar St", LAddr: "222 Maple St" }]

},

{

Name: { FName: "Alice", LName: "White" },

CompanyName: "Infosys",

Salary: 52000,

Designation: "Developer",

Age: 30,

Expertise: ["Python", "Django", "Mongodb"],

DOB: new Date("1994-02-25"),

Email: "alice.white@example.com",

Contact: "3213214321",

Address: [{ PAddr: "333 Birch St", LAddr: "444 Fir St" }]

},

{

Name: { FName: "Eve", LName: "Green" },

CompanyName: "Wipro",

Salary: 47000,

Designation: "Programmer",

Age: 29,

Expertise: ["Mongodb", "Mysql", "Cassandra"],

DOB: new Date("1995-10-12"),

Email: "eve.green@example.com",

Contact: "4321432143",

Address: [{ PAddr: "555 Palm St", LAddr: "666 Willow St" }]

}

]);

Select all documents where Designation is "Programmer" and Salary is greater than 30000:

db.Employee.find({

Designation: "Programmer",

Salary: { $gt: 30000 }

});

Create a new document if no document exists with {Designation: "Tester", CompanyName: "TCS", Age: 25}:

db.Employee.updateOne(

{ Designation: "Tester", CompanyName: "TCS", Age: 25 },

{

$setOnInsert: {

Name: { FName: "New", LName: "Tester" },

CompanyName: "TCS",

Salary: 30000,

Designation: "Tester",

Age: 25,

Expertise: ["Testing"],

DOB: new Date("1998-05-15"),

Email: "new.tester@tcs.com",

Contact: "5675675678",

Address: [{ PAddr: "777 Spruce St", LAddr: "888 Maple St" }]

}

},

{ upsert: true }

);

Increase the salary of each Employee working with "Infosys" by 10000:

db.Employee.updateMany(

{ CompanyName: "Infosys" },

{ $inc: { Salary: 10000 } }

);

Find all employees working with "TCS" and reduce their salary by

db.Employee.updateMany(

{ CompanyName: "TCS" },

{ $inc: { Salary: -5000 } }

);

Return documents where Designation is not "Tester":

db.Employee.find({

Designation: { $ne: "Tester" }

});

Find all employees with an exact match on an array with Expertise: ['Mongodb', 'Mysql', 'Cassandra']:

db.Employee.find({

Expertise: ["Mongodb", "Mysql", "Cassandra"]

});

M2.Design and Develop MongoDB Queries using CRUD operations:

Create Employee collection by considering following Fields:

i. Name: Embedded Doc (FName, LName)

ii. Company Name: String

iii. Salary: Number

iv. Designation: String

v. Age: Number

vi. Expertise: Array

vii. DOB: String or Date

viii. Email id: String

ix. Contact: String

x. Address: Array of Embedded Doc (PAddr, LAddr)

Insert at least 5 documents in collection by considering above

attribute and execute following queries:

1. Final name of Employee where age is less than 30 and salary more

than 50000.

2. Creates a new document if no document in the employee collection

contains

{Designation: "Tester", Company\_name: "TCS", Age: 25}

3. Selects all documents in the collection where the field age has

a value less than 30 or the value of the salary field is greater

than 40000.

4. Find documents where Designation is not equal to "Developer".

5. Find \_id, Designation, Address and Name from all documents where

Company\_name is "Infosys".

6. Display only FName and LName of all Employees

1. Final name of Employee where age is less than 30 and salary more than 50000.

db.Employee.find(

{ Age: { $lt: 30 }, Salary: { $gt: 50000 } },

{ "Name.FName": 1, "Name.LName": 1, \_id: 0 }

);

2. Create a new document if no document in the employee collection contains { Designation: "Tester", Company\_name: "TCS", Age: 25 }.

db.Employee.updateOne(

{ Designation: "Tester", CompanyName: "TCS", Age: 25 },

{

$setOnInsert: {

Name: { FName: "New", LName: "Tester" },

CompanyName: "TCS",

Salary: 30000,

Designation: "Tester",

Age: 25,

Expertise: ["Testing"],

DOB: new Date("1998-05-15"),

Email: "new.tester@tcs.com",

Contact: "5675675678",

Address: [{ PAddr: "777 Spruce St", LAddr: "888 Maple St" }]

}

},

{ upsert: true }

);

3. Selects all documents in the collection where the field Age is less than 30 or the field Salary is greater than 40000.

db.Employee.find({

$or: [

{ Age: { $lt: 30 } },

{ Salary: { $gt: 40000 } }

]

});

4. Find documents where Designation is not equal to "Developer".

db.Employee.find({

Designation: { $ne: "Developer" }

});

5. Find \_id, Designation, Address, and Name from all documents where CompanyName is "Infosys".

db.Employee.find(

{ CompanyName: "Infosys" },

{ \_id: 1, Designation: 1, Address: 1, Name: 1 }

);

6. Display only FName and LName of all employees.

db.Employee.find(

{},

{ "Name.FName": 1, "Name.LName": 1, \_id: 0 }

);

M3.Design and Develop MongoDB Queries using CRUD operations:

Create Employee collection by considering following Fields:

i. Emp\_id : Number

ii. Name: Embedded Doc (FName, LName)

iii. Company Name: String

iv. Salary: Number

v. Designation: String

vi. Age: Number

vii. Expertise: Array

viii. DOB: String or Date

ix. Email id: String

x. Contact: String

xi. Address: Array of Embedded Doc (PAddr, LAddr)

Insert at least 5 documents in collection by considering above

attribute and execute following queries:

1. Creates a new document if no document in the employee collection

contains

{Designation: "Tester", Company\_name: "TCS", Age: 25}

2. Finds all employees working with Company\_name: "TCS" and

increase their salary by 2000.

3. Matches all documents where the value of the field Address is an

embedded document that contains only the field city with the

value "Pune" and the field Pin\_code with the value "411001".

4. Find employee details who are working as "Developer" or

"Tester".

5. Drop Single documents where designation="Developer".

6. Count number of documents in employee collection.

1. **Create a new document if no document in the employee collection contains {Designation: "Tester", Company\_name: "TCS", Age: 25}**

db.Employee.updateOne(

{ Designation: "Tester", CompanyName: "TCS", Age: 25 },

{

$setOnInsert: {

Emp\_id: 1001,

Name: { FName: "New", LName: "Tester" },

CompanyName: "TCS",

Salary: 30000,

Designation: "Tester",

Age: 25,

Expertise: ["Testing"],

DOB: new Date("1998-05-15"),

Email: "new.tester@tcs.com",

Contact: "5675675678",

Address: [{ PAddr: "123 Test St", LAddr: "456 Test Ave" }]

}

},

{ upsert: true }

);

2. **Find all employees working with CompanyName: "TCS" and increase their salary by 2000**

db.Employee.updateMany(

{ CompanyName: "TCS" },

{ $inc: { Salary: 2000 } }

);

3. **Match all documents where the value of the field Address is an embedded document containing only the field city with the value "Pune" and the field Pin\_code with the value "411001"**

db.Employee.find(

{ Address: { $elemMatch: { city: "Pune", Pin\_code: "411001" } } }

);

4. **Find employee details who are working as "Developer" or "Tester"**

db.Employee.find(

{ Designation: { $in: ["Developer", "Tester"] } }

);

5. **Drop single documents where Designation: "Developer"**

db.Employee.deleteOne(

{ Designation: "Developer" }

);

6. **Count the number of documents in the employee collection**

db.Employee.countDocuments();

M4. Design and Develop MongoDB Queries using Aggregation operations:

Create Employee collection by considering following Fields:

i. Emp\_id : Number

ii. Name: Embedded Doc (FName, LName)

iii. Company Name: String

iv. Salary: Number

v. Designation: String

vi. Age: Number

vii. Expertise: Array

viii. DOB: String or Date

ix. Email id: String

x. Contact: String

xi. Address: Array of Embedded Doc (PAddr, LAddr)

Insert at least 5 documents in collection by considering above

attribute and execute following:

1. Using aggregation Return Designation with Total Salary is Above

200000.

2. Using Aggregate method returns names and \_id in upper case and

in alphabetical order.

3. Using aggregation method find Employee with Total Salary for

Each City with Designation="DBA".

4. Create Single Field Indexes on Designation field of employee

collection

5. To Create Multikey Indexes on Expertise field of employee

collection.

6. Create an Index on Emp\_id field, compare the time require to

search Emp\_id before and after creating an index. (Hint Add at

least 10000 Documents)

7. Return a List of Indexes on created on employee Collection.

1. Return Designation with Total Salary Above 200000

db.Employee.aggregate([

{

$group: {

\_id: "$Designation",

totalSalary: { $sum: "$Salary" }

}

},

{

$match: {

totalSalary: { $gt: 200000 }

}

}

]);

2. Return Names and \_id in Upper Case and Alphabetical Order

db.Employee.aggregate([

{

$project: {

\_id: 1,

"Name.FName": { $toUpper: "$Name.FName" },

"Name.LName": { $toUpper: "$Name.LName" }

}

},

{

$sort: { "Name.FName": 1 }

}

]);

3. Find Total Salary for Each City for Employees with Designation="DBA"

db.Employee.aggregate([

{

$match: { Designation: "DBA" }

},

{

$unwind: "$Address"

},

{

$group: {

\_id: "$Address.PAddr",

totalSalary: { $sum: "$Salary" }

}

}

]);

4. Create Single Field Index on Designation

db.Employee.createIndex({ Designation: 1 });

5. Create Multikey Index on Expertise

db.Employee.createIndex({ Expertise: 1 });

6. Create an Index on Emp\_id and Compare Query Time

// Define the document structure you want to insert

const employeeDoc = {

Emp\_id: 1,

Name: { FName: "John", LName: "Doe" },

CompanyName: "TechCorp",

Salary: 60000,

Designation: "Engineer",

Age: 30,

Expertise: ["MongoDB", "JavaScript"],

DOB: new Date("1993-04-15"),

Email: "john.doe@techcorp.com",

Contact: "1234567890",

Address: [{ PAddr: "123 Main St", LAddr: "456 Secondary St" }]

};

// Loop to insert the same document 10,000 times

for (let i = 0; i < 10000; i++) {

db.Employee.insertOne(employeeDoc);

}

**6. Create an Index on Emp\_id and Compare Query Time**

First, perform a search without the index:

var startTime = new Date();

db.Employee.find({ Emp\_id: 5000 }); // Sample query

var endTime = new Date();

print("Time taken without index: ", endTime - startTime);

Create an index on the Emp\_id field:

db.Employee.createIndex({ Emp\_id: 1 });

Now, search again and check the time taken:

var startTime = new Date();

db.Employee.find({ Emp\_id: 5000 });

var endTime = new Date();

print("Time taken with index: ", endTime - startTime);

7. Return a List of Indexes on the Employee Collection

db.Employee.getIndexes();

M5. Design and Develop MongoDB Queries using Aggregation operations:

Create Employee collection by considering following Fields:

i. Emp\_id : Number

ii. Name: Embedded Doc (FName, LName)

iii. Company Name: String

iv. Salary: Number

v. Designation: String

vi. Age: Number

vii. Expertise: Array

viii. DOB: String or Date

ix. Email id: String

x. Contact: String

xi. Address: Array of Embedded Doc (PAddr, LAddr)

Insert at least 5 documents in collection by considering above

attribute and execute following:

1. Using aggregation Return separates value in the Expertise array

and return sum of each element of array.

2. Using Aggregate method return Max and Min Salary for each

company.

3. Using Aggregate method find Employee with Total Salary for Each

City with Designation="DBA".

4. Using aggregation method Return separates value in the Expertise

array for employee name where Swapnil Jadhav

5. To Create Compound Indexes on Name: 1, Age: -1

6. Create an Index on Emp\_id field, compare the time require to

search Emp\_id before and after creating an index. (Hint Add at

least 10000 Documents)

7. Return a List of Indexes on created on employee Collection.

1. Using Aggregation to Return Separate Values in the Expertise Array and Sum Each Element’s Occurrences

db.Employee.aggregate([

{ $unwind: "$Expertise" },

{ $group: { \_id: "$Expertise", count: { $sum: 1 } } }

]);

2. Using Aggregation to Return Max and Min Salary for Each Company

db.Employee.aggregate([

{

$group: {

\_id: "$CompanyName",

maxSalary: { $max: "$Salary" },

minSalary: { $min: "$Salary" }

}

}

]);

**3. Using Aggregation to Find Total Salary for Each City with Designation "DBA"**

db.Employee.aggregate([

{ $match: { Designation: "DBA" } },

{ $unwind: "$Address" },

{

$group: {

\_id: "$Address.PAddr",

totalSalary: { $sum: "$Salary" }

}

}

]);

4. Using Aggregation to Return Separate Values in the Expertise Array for Employee Named "Swapnil Jadhav"

db.Employee.aggregate([

{ $match: { "Name.FName": "Swapnil", "Name.LName": "Jadhav" } },

{ $unwind: "$Expertise" },

{

$group: {

\_id: "$Name",

expertiseList: { $addToSet: "$Expertise" }

}

}

]);

5. Creating a Compound Index on Name and Age

db.Employee.createIndex({ "Name.FName": 1, Age: -1 });

6. Creating an Index on Emp\_id Field and Comparing Search Times Before and After Indexing

**Check Search Time Without Index**:

db.Employee.find({ Emp\_id: 1 }).explain("executionStats");

**Create Index on Emp\_id**:

db.Employee.createIndex({ Emp\_id: 1 });

**Check Search Time With Index**:

db.Employee.find({ Emp\_id: 1 }).explain("executionStats");

7. Returning a List of Indexes on the Employee Collection

db.Employee.getIndexes();

M6. Design MongoDB database and perform following Map reduce operation:

Create Employee collection by considering following Fields:

i. Name: Embedded Doc (FName, LName)

ii. Company Name: String

iii. Salary: Number

iv. Designation: String

v. Age: Number

vi. Expertise: Array

vii. DOB: String or Date

viii. Email id: String

ix. Contact: String

x. Address: Array of Embedded Doc (PAddr, LAddr)

Execute the following query:

1. Display the total salary of per company

2. Display the total salary of company Name:"TCS"

3. Return the average salary of company whose address is “Pune".

4. Display total count for “City=Pune”

5. Return count for city pune and age greater than 40.

1. **Create Employee Collection with Sample Data**

db.Employee.insertMany([

{

Name: { FName: "John", LName: "Doe" },

CompanyName: "TCS",

Salary: 60000,

Designation: "Manager",

Age: 45,

Expertise: ["Java", "MongoDB"],

DOB: "1978-10-15",

Email: "john.doe@tcs.com",

Contact: "1234567890",

Address: [{ PAddr: "Pune", LAddr: "Sector 12" }]

},

{

Name: { FName: "Jane", LName: "Smith" },

CompanyName: "TCS",

Salary: 70000,

Designation: "Developer",

Age: 35,

Expertise: ["JavaScript", "MongoDB"],

DOB: "1988-03-25",

Email: "jane.smith@tcs.com",

Contact: "1234567891",

Address: [{ PAddr: "Pune", LAddr: "Sector 10" }]

},

{

Name: { FName: "Mark", LName: "Taylor" },

CompanyName: "Infosys",

Salary: 50000,

Designation: "Tester",

Age: 38,

Expertise: ["QA", "Automation"],

DOB: "1985-06-12",

Email: "mark.taylor@infosys.com",

Contact: "1234567892",

Address: [{ PAddr: "Pune", LAddr: "Sector 8" }]

},

{

Name: { FName: "Sara", LName: "Connor" },

CompanyName: "Wipro",

Salary: 55000,

Designation: "Developer",

Age: 42,

Expertise: ["Python", "Django"],

DOB: "1981-05-01",

Email: "sara.connor@wipro.com",

Contact: "1234567893",

Address: [{ PAddr: "Mumbai", LAddr: "Sector 5" }]

},

{

Name: { FName: "Sam", LName: "Johnson" },

CompanyName: "TCS",

Salary: 65000,

Designation: "Developer",

Age: 29,

Expertise: ["Node.js", "MongoDB"],

DOB: "1994-07-21",

Email: "sam.johnson@tcs.com",

Contact: "1234567894",

Address: [{ PAddr: "Pune", LAddr: "Sector 6" }]

}

]);

2. **MapReduce Queries**

**1. Display the total salary per company**

var mapFunction = function() {

emit(this.CompanyName, this.Salary);

};

var reduceFunction = function(key, values) {

return Array.sum(values);

};

db.Employee.mapReduce(mapFunction, reduceFunction, { out: { inline: 1 } });

2. **Display the total salary of company "TCS"**

var mapFunction = function() {

if (this.CompanyName == "TCS") {

emit(this.CompanyName, this.Salary);

}

};

var reduceFunction = function(key, values) {

return Array.sum(values);

};

db.Employee.mapReduce(mapFunction, reduceFunction, { out: { inline: 1 } });

3. **Return the average salary of company whose address is "Pune"**

var mapFunction = function() {

for (var i = 0; i < this.Address.length; i++) {

if (this.Address[i].PAddr == "Pune") {

emit(this.CompanyName, { salary: this.Salary, count: 1 });

}

}

};

var reduceFunction = function(key, values) {

var result = { salary: 0, count: 0 };

values.forEach(function(value) {

result.salary += value.salary;

result.count += value.count;

});

return result;

};

var finalizeFunction = function(key, reducedValue) {

reducedValue.averageSalary = reducedValue.salary / reducedValue.count;

return reducedValue;

};

db.Employee.mapReduce(mapFunction, reduceFunction, { out: { inline: 1 }, finalize: finalizeFunction });

4. **Display total count for "City=Pune"**

var mapFunction = function() {

for (var i = 0; i < this.Address.length; i++) {

if (this.Address[i].PAddr == "Pune") {

emit("Pune", 1);

}

}

};

var reduceFunction = function(key, values) {

return Array.sum(values);

};

db.Employee.mapReduce(mapFunction, reduceFunction, { out: { inline: 1 } });

5. **Return count for city "Pune" and age greater than 40**

var mapFunction = function() {

for (var i = 0; i < this.Address.length; i++) {

if (this.Address[i].PAddr == "Pune" && this.Age > 40) {

emit("Pune\_Age\_Above\_40", 1);

}

}

};

var reduceFunction = function(key, values) {

return Array.sum(values);

};

db.Employee.mapReduce(mapFunction, reduceFunction, { out: { inline: 1 } });