**5 Create a collection sites(url,dateofaccess). Write a MapReduce function to find the no. of times a site was accessed in a month.**

// Step 1: Insert sample data into the 'sites' collection

db.sites.insertMany([

{ url: "example.com", dateofaccess: new Date("2025-05-10") },

{ url: "example.com", dateofaccess: new Date("2025-05-15") },

{ url: "example.com", dateofaccess: new Date("2025-06-01") },

{ url: "google.com", dateofaccess: new Date("2025-05-20") },

{ url: "google.com", dateofaccess: new Date("2025-05-22") }

]);

// Step 2: Define the Map function

// This function emits a key for each site and month-year, with a count of 1

var mapFunction = function() {

var date = new Date(this.dateofaccess);

var month = date.getMonth() + 1; // getMonth() returns 0-11, so add 1

var year = date.getFullYear();

// Format key as "url\_yyyy-mm", with leading zero for month if needed

var key = this.url + "\_" + year + "-" + (month < 10 ? "0" + month : month);

emit(key, 1); // Emit key with value 1 for each access

};

// Step 3: Define the Reduce function

// This function sums all the counts for the same key

var reduceFunction = function(key, values) {

return Array.sum(values);

};

// Step 4: Run the MapReduce operation

db.sites.mapReduce(

mapFunction,

reduceFunction,

{

out: "site\_access\_per\_month" // Output collection name

}

);

// Step 5: View the results

db.site\_access\_per\_month.find().forEach(printjson);

**12 Implement Indexing and querying with MongoDB using following example.**

**Students(stud\_id, stud\_name,stud\_addr,stud\_marks)**

// Insert sample data with Indian names

db.Students.insertMany([

{ stud\_id: 1, stud\_name: "Amit Sharma", stud\_addr: "Delhi", stud\_marks: 82 },

{ stud\_id: 2, stud\_name: "Sneha Patel", stud\_addr: "Ahmedabad", stud\_marks: 91 },

{ stud\_id: 3, stud\_name: "Rajesh Kumar", stud\_addr: "Mumbai", stud\_marks: 77 },

{ stud\_id: 4, stud\_name: "Priya Singh", stud\_addr: "Bangalore", stud\_marks: 89 },

{ stud\_id: 5, stud\_name: "Anjali Rao", stud\_addr: "Hyderabad", stud\_marks: 85 }

]);

// Create indexes

db.Students.createIndex({ stud\_id: 1 });

db.Students.createIndex({ stud\_name: 1 });

db.Students.createIndex({ stud\_addr: 1, stud\_marks: -1 });

// Queries

db.Students.find({ stud\_id: 3 });

db.Students.find({ stud\_name: "Priya Singh" });

db.Students.find({ stud\_addr: "Delhi", stud\_marks: { $gt: 80 } }).sort({ stud\_marks: -1 });

**16 Implement Map reduce operation with following example using MongoDB**

**Students(stud\_id, stud\_name,stud\_addr,stud\_marks)**

**AND**

**Write a PL/SQL code to calculate total and percentage of marks of the students in four subjects.**

**PART A:**

// Sample data insertion (if needed)

db.Students.insertMany([

{ stud\_id: 1, stud\_name: "Amit Sharma", stud\_addr: "Delhi", stud\_marks: 320 },

{ stud\_id: 2, stud\_name: "Sneha Patel", stud\_addr: "Ahmedabad", stud\_marks: 350 },

{ stud\_id: 3, stud\_name: "Rajesh Kumar", stud\_addr: "Delhi", stud\_marks: 300 },

{ stud\_id: 4, stud\_name: "Priya Singh", stud\_addr: "Bangalore", stud\_marks: 310 },

{ stud\_id: 5, stud\_name: "Anjali Rao", stud\_addr: "Ahmedabad", stud\_marks: 340 }

]);

// Map function: emit stud\_addr as key and stud\_marks as value

var mapFunction = function() {

emit(this.stud\_addr, this.stud\_marks);

};

// Reduce function: sum all marks for each stud\_addr

var reduceFunction = function(key, values) {

return Array.sum(values);

};

// Run MapReduce

db.Students.mapReduce(

mapFunction,

reduceFunction,

{ out: "total\_marks\_by\_city" }

);

// View results

db.total\_marks\_by\_city.find();

**PART B:**

-- Step 1: Create the students table with marks for 4 subjects

CREATE TABLE students (

stud\_id NUMBER PRIMARY KEY,

stud\_name VARCHAR2(50),

marks\_sub1 NUMBER(3),

marks\_sub2 NUMBER(3),

marks\_sub3 NUMBER(3),

marks\_sub4 NUMBER(3)

);

-- Step 2: Insert sample student data

INSERT INTO students VALUES (1, 'Amit Sharma', 85, 78, 92, 88);

INSERT INTO students VALUES (2, 'Sneha Patel', 90, 85, 88, 91);

INSERT INTO students VALUES (3, 'Rajesh Kumar', 75, 80, 70, 68);

INSERT INTO students VALUES (4, 'Priya Singh', 88, 92, 95, 90);

INSERT INTO students VALUES (5, 'Anjali Rao', 80, 82, 79, 85);

COMMIT;

-- Step 3: PL/SQL block to calculate total and percentage for each student

DECLARE

v\_total NUMBER(3); -- Variable to store total marks

v\_percentage NUMBER(5,2); -- Variable to store percentage with 2 decimal places

BEGIN

-- Loop through each student record

FOR rec IN (

SELECT stud\_id, stud\_name, marks\_sub1, marks\_sub2, marks\_sub3, marks\_sub4 FROM students

)

LOOP

-- Calculate total marks by summing marks of 4 subjects

v\_total := rec.marks\_sub1 + rec.marks\_sub2 + rec.marks\_sub3 + rec.marks\_sub4;

-- Calculate percentage assuming each subject is out of 100 marks

v\_percentage := (v\_total / 400) \* 100;

-- Display student details, total marks and percentage

DBMS\_OUTPUT.PUT\_LINE('Student: ' || rec.stud\_name);

DBMS\_OUTPUT.PUT\_LINE('Total Marks: ' || v\_total);

DBMS\_OUTPUT.PUT\_LINE('Percentage: ' || ROUND(v\_percentage, 2) || '%');

DBMS\_OUTPUT.PUT\_LINE('-----------------------------');

END LOOP;

END;

/

**17 Create following collection and using MongoDB implement all CRUD operations.**

**Orders( cust\_id, amount, status)**

// Create and insert orders (Create)

db.Orders.insertOne({ cust\_id: 101, amount: 500, status: "pending" });

db.Orders.insertMany([

{ cust\_id: 102, amount: 1200, status: "completed" },

{ cust\_id: 103, amount: 700, status: "pending" },

{ cust\_id: 104, amount: 1500, status: "shipped" }

]);

// Read operations (Retrieve)

db.Orders.find(); // Find all orders

db.Orders.find({ status: "pending" }); // Find orders with status "pending"

db.Orders.find({ amount: { $gt: 1000 } }); // Find orders with amount > 1000

// Update operations

db.Orders.updateOne(

{ cust\_id: 101 },

{ $set: { status: "completed" } }

);

db.Orders.updateMany(

{ status: "pending" },

{ $inc: { amount: 100 } } // Increase amount by 100

);

// Delete operations

db.Orders.deleteOne({ cust\_id: 104 }); // Delete order with cust\_id 104

db.Orders.deleteMany({ status: "completed" }); // Delete all completed orders

**23 Implement all Aggregation operations and types of indexing with following collection using MongoDB.**

**Employee(emp\_id, emp\_name,emp\_dept,salary)**

// Step 1: Insert sample data

db.Employee.insertMany([

{ emp\_id: 101, emp\_name: "Amit Sharma", emp\_dept: "HR", salary: 40000 },

{ emp\_id: 102, emp\_name: "Sneha Patel", emp\_dept: "IT", salary: 60000 },

{ emp\_id: 103, emp\_name: "Rajesh Kumar", emp\_dept: "Finance", salary: 50000 },

{ emp\_id: 104, emp\_name: "Priya Singh", emp\_dept: "IT", salary: 75000 },

{ emp\_id: 105, emp\_name: "Anjali Rao", emp\_dept: "HR", salary: 42000 },

{ emp\_id: 106, emp\_name: "Manoj Verma", emp\_dept: "Finance", salary: 52000 }

]);

// Step 2: Aggregation Operations

// 1. $match – filter employees from IT department

db.Employee.aggregate([

{ $match: { emp\_dept: "IT" } }

]);

// 2. $group – group by department and calculate average salary

db.Employee.aggregate([

{ $group: { \_id: "$emp\_dept", avg\_salary: { $avg: "$salary" } } }

]);

// 3. $sort – sort employees by salary descending

db.Employee.aggregate([

{ $sort: { salary: -1 } }

]);

// 4. $project – include only emp\_name and salary

db.Employee.aggregate([

{ $project: { emp\_name: 1, salary: 1, \_id: 0 } }

]);

// 5. $count – count total number of employees

db.Employee.aggregate([

{ $count: "total\_employees" }

]);

// 6. $limit – show only top 3 highest paid employees

db.Employee.aggregate([

{ $sort: { salary: -1 } },

{ $limit: 3 }

]);

// Step 3: Indexing

// 1. Single Field Index – on emp\_id

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

// 2. Compound Index – on emp\_dept and salary

db.Employee.createIndex({ emp\_dept: 1, salary: -1 });

// 3. Unique Index – to ensure emp\_id is unique

db.Employee.createIndex({ emp\_id: 1 }, { unique: true });

// 4. Text Index – on emp\_name for search

db.Employee.createIndex({ emp\_name: "text" });

// Optional: Check all created indexes

db.Employee.getIndexes();

**25 Design and Implement any 5 query using MongoDB**

1. **Create a collection called ‘games’.**
2. **Add 5 games to the database. Give each document the following properties: name, gametype, score (out of 100), achievements**
3. **Write a query that returns all the games**
4. **Write a query that returns the 3 highest scored games.**
5. **Write a query that returns all the games that have both the ‘Game Maser’ and**

**the ‘Speed Demon’ achievements.**

// Step 1: Create and insert 5 games into the 'games' collection

db.games.insertMany([

{

name: "Racing Thunder",

gametype: "Racing",

score: 88,

achievements: ["Speed Demon", "Track Master"]

},

{

name: "Battle Arena",

gametype: "Action",

score: 92,

achievements: ["Game Master", "Sharp Shooter"]

},

{

name: "Mystery Quest",

gametype: "Adventure",

score: 75,

achievements: ["Explorer", "Puzzle Solver"]

},

{

name: "Speed Legend",

gametype: "Racing",

score: 95,

achievements: ["Speed Demon", "Game Master"]

},

{

name: "Kingdom Clash",

gametype: "Strategy",

score: 89,

achievements: ["Game Master", "Tactician"]

}

]);

// Step 2: Query 1 – Return all games

db.games.find();

// Step 3: Query 2 – Return top 3 highest scored games

db.games.find().sort({ score: -1 }).limit(3);

// Step 4: Query 3 – Return games that have both 'Game Master' and 'Speed Demon' achievements

db.games.find({

achievements: { $all: ["Game Master", "Speed Demon"] }

});

**31.Create database :Citydetails(\_id,name,area,population(total,Adults,seniorcitizens,sexratio), geography(avgtemp, avgrainfall, longitude, latitude))**

**1. Find the total population in pune.**

**2. returns all city with total population greater than 10 million**

**3. returns the average populations for each city.**

**4. returns the minimum and maximum cities by population for each city.**

// Step 1: Insert sample city data

db.Citydetails.insertMany([

{

\_id: 1,

name: "Pune",

area: 731,

population: {

total: 7600000,

Adults: 5400000,

seniorcitizens: 800000,

sexratio: 948

},

geography: {

avgtemp: 24,

avgrainfall: 722,

longitude: 73.8567,

latitude: 18.5204

}

},

{

\_id: 2,

name: "Mumbai",

area: 603,

population: {

total: 12400000,

Adults: 9000000,

seniorcitizens: 1200000,

sexratio: 852

},

geography: {

avgtemp: 27,

avgrainfall: 2422,

longitude: 72.8777,

latitude: 19.0760

}

},

{

\_id: 3,

name: "Nagpur",

area: 227,

population: {

total: 2400000,

Adults: 1700000,

seniorcitizens: 250000,

sexratio: 961

},

geography: {

avgtemp: 26,

avgrainfall: 1200,

longitude: 79.0882,

latitude: 21.1458

}

},

{

\_id: 4,

name: "Delhi",

area: 1484,

population: {

total: 19000000,

Adults: 14000000,

seniorcitizens: 2200000,

sexratio: 850

},

geography: {

avgtemp: 25,

avgrainfall: 800,

longitude: 77.1025,

latitude: 28.7041

}

}

]);

// Step 2: Query – Find total population in Pune

db.Citydetails.find(

{ name: "Pune" },

{ \_id: 0, name: 1, "population.total": 1 }

);

// Step 3: Query – Cities with total population greater than 10 million

db.Citydetails.find(

{ "population.total": { $gt: 10000000 } },

{ \_id: 0, name: 1, "population.total": 1 }

);

// Step 4: Query – Average population across all cities

db.Citydetails.aggregate([

{

$group: {

\_id: null,

avg\_population: { $avg: "$population.total" }

}

}

]);

// Step 5: Query – Minimum and maximum population among all cities

db.Citydetails.aggregate([

{

$group: {

\_id: null,

min\_population: { $min: "$population.total" },

max\_population: { $max: "$population.total" }

}

}

]);

**32.Create database :Citydetails(\_id,name,area,population(total,Adults,seniorcitizens,sexratio), geography (avgtemp, avgrainfall, longitude, latitude))**

1. **Find area wise total population and sort them in increasing order.**
2. **Retrieve name and area where average rain fall is greater than 60**
3. **Create index on city and area find the max population in Mumbai**
4. **Create index on name.**

// Step 1: Insert sample data into Citydetails collection

db.Citydetails.insertMany([

{

\_id: 1,

name: "Pune",

area: 731,

population: {

total: 7600000,

Adults: 5400000,

seniorcitizens: 800000,

sexratio: 948

},

geography: {

avgtemp: 24,

avgrainfall: 722,

longitude: 73.8567,

latitude: 18.5204

}

},

{

\_id: 2,

name: "Mumbai",

area: 603,

population: {

total: 12400000,

Adults: 9000000,

seniorcitizens: 1200000,

sexratio: 852

},

geography: {

avgtemp: 27,

avgrainfall: 2422,

longitude: 72.8777,

latitude: 19.0760

}

},

{

\_id: 3,

name: "Nagpur",

area: 227,

population: {

total: 2400000,

Adults: 1700000,

seniorcitizens: 250000,

sexratio: 961

},

geography: {

avgtemp: 26,

avgrainfall: 1200,

longitude: 79.0882,

latitude: 21.1458

}

},

{

\_id: 4,

name: "Delhi",

area: 1484,

population: {

total: 19000000,

Adults: 14000000,

seniorcitizens: 2200000,

sexratio: 850

},

geography: {

avgtemp: 25,

avgrainfall: 800,

longitude: 77.1025,

latitude: 28.7041

}

}

]);

// Step 2: Query – Find area-wise total population and sort in increasing order

db.Citydetails.aggregate([

{

$project: {

name: 1,

area: 1,

total\_population: "$population.total"

}

},

{

$sort: { area: 1 }

}

]);

// Step 3: Query – Retrieve name and area where average rainfall > 60

db.Citydetails.find(

{ "geography.avgrainfall": { $gt: 60 } },

{ \_id: 0, name: 1, area: 1 }

);

// Step 4: Create index on city (name) and area, then find max population in Mumbai

db.Citydetails.createIndex({ name: 1, area: 1 });

// Find max population in Mumbai

db.Citydetails.find(

{ name: "Mumbai" },

{ \_id: 0, name: 1, "population.total": 1 }

).sort({ "population.total": -1 }).limit(1);

// Step 5: Create index on name only

db.Citydetails.createIndex({ name: 1 });