# MONGODB

3 Design Mongo DB Schema for Restaurant having RestaurantId, Name, Cuisine, Score, Address i) Write a MongoDB query to display all the documents in the collection restaurants. ii) Write a MongoDB query to display all the documents in the collection restaurants with fields Name, Cuisine, Score and exclude Id and Address iii) Write a MongoDB query to find the restaurants who achieved a score more than 80. iv) Write a MongoDB query to find the restaurant having highest score. Write a MongoDB query to find the count of restaurants for each cuisine

use myDatabase; db.createCollection("restaurants"); db.restaurants.insertMany([

{

"RestaurantId": "R001",

"Name": "The Gourmet Kitchen", "Cuisine": "Italian",

"Score": 85,

"Address": {

"Street": "123 Pasta Lane", "City": "Rome",

"Zipcode": "10001"

}

},

{

"RestaurantId": "R002", "Name": "Sushi Haven", "Cuisine": "Japanese", "Score": 92,

"Address": {

"Street": "456 Sashimi Ave", "City": "Tokyo",

"Zipcode": "20002"

}

},

{

"RestaurantId": "R003", "Name": "Burger Palace", "Cuisine": "American", "Score": 78,

"Address": {

"Street": "789 Burger Blvd", "City": "New York", "Zipcode": "30003"

}

},

{

"RestaurantId": "R004", "Name": "Spicy Tadka", "Cuisine": "Indian", "Score": 88,

"Address": {

"Street": "321 Curry Road", "City": "Delhi",

"Zipcode": "40004"

}

},

{

"RestaurantId": "R005", "Name": "Dragon Wok", "Cuisine": "Chinese", "Score": 82,

"Address": {

"Street": "654 Noodle Street", "City": "Beijing",

"Zipcode": "50005"

}

}

]);

db.restaurants.find();

db.restaurants.find({}, { Address: 0, \_id: 0 }); db.restaurants.find({ Score: { $gt: 80 } }); db.restaurants.find().sort({ Score: -1 }).limit(1); db.restaurants.aggregate([

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

]);

.9 Design and Develop MongoDB Queries & use aggregation and indexing Create ‘zipcode’ collection with city,state & population Insert atleast 10 records with different variations. Execute following queries. i)Display records from collection. ii)Display total population statewise. iii) Display total population statewise where population > 20000. iv)Create index on state. v) Display all index for collection.

// Create 'zipcode' collection and insert data db.zipcode.insertMany([

{ city: "Mumbai", state: "Maharashtra", population: 12478447 },

{ city: "Delhi", state: "Delhi", population: 11007835 },

{ city: "Bangalore", state: "Karnataka", population: 8436675 },

{ city: "Hyderabad", state: "Telangana", population: 6809970 },

{ city: "Ahmedabad", state: "Gujarat", population: 5577940 },

{ city: "Chennai", state: "Tamil Nadu", population: 7090000 },

{ city: "Kolkata", state: "West Bengal", population: 4486679 },

{ city: "Pune", state: "Maharashtra", population: 3124458 },

{ city: "Surat", state: "Gujarat", population: 4467797 },

{ city: "Jaipur", state: "Rajasthan", population: 3073350 }

]);

db.zipcode.find(); db.zipcode.aggregate([

{ $group: { \_id: "$state", totalPopulation: { $sum: "$population" } } }

]);

db.zipcode.aggregate([

{ $group: { \_id: "$state", totalPopulation: { $sum: "$population" } } },

{ $match: { totalPopulation: { $gt: 20000 } } }

]);

db.zipcode.createIndex({ state: 1 }); db.zipcode.getIndexes();

15. Design and Develop MongoDB Queries & use aggregation and indexing Create ‘zipcode’ collection with city,state & population Insert atleast 10 records with different variations. Execute following queries. i)Display records from collection. ii)Display total population statewise. iii) Display total population statewise where population > 20000. iv)Display average populations for cities in each state. v) Display smallest and largest population for each state.cities by

db.zipcode.insertMany([

{ city: "City1", state: "State1", population: 5000 },

{ city: "City2", state: "State1", population: 10000 },

{ city: "City3", state: "State2", population: 15000 },

{ city: "City4", state: "State2", population: 25000 },

{ city: "City5", state: "State3", population: 3000 },

{ city: "City6", state: "State3", population: 7000 },

{ city: "City7", state: "State4", population: 22000 },

{ city: "City8", state: "State4", population: 32000 },

{ city: "City9", state: "State5", population: 10000 },

{ city: "City10", state: "State5", population: 18000 }

]);

db.zipcode.find({}); db.zipcode.aggregate([

{ $group: { \_id: "$state", total\_population: { $sum: "$population" } } }

]);

db.zipcode.aggregate([

{ $match: { population: { $gt: 20000 } } },

{ $group: { \_id: "$state", total\_population: { $sum: "$population" } } }

]);

db.zipcode.aggregate([

{ $group: { \_id: "$state", average\_population: { $avg: "$population" } } }

]);

db.zipcode.aggregate([

{ $group: {

\_id: "$state",

min\_population: { $min: "$population" }, max\_population: { $max: "$population" }

} }

]);

13 Design and Develop MongoDB Queries Create ‘users’ collection with name,age and status. Insert atleast 10 records with different variations. Execute following queries. i)Display first five records from collection. ii)Update status as “rejected” if age is less than 18. iii)Delete the record of user whose name is ‘Akshata’l iv)Delete records having age greater than 50 v)Display users having age less than 40 and status = ‘paid’

db.createCollection('users'); db.users.insertMany([

{ name: "John", age: 25, status: "paid" },

{ name: "Akshata", age: 17, status: "pending" },

{ name: "Raj", age: 30, status: "paid" },

{ name: "Ravi", age: 45, status: "unpaid" },

{ name: "Priya", age: 20, status: "paid" },

{ name: "Sara", age: 35, status: "pending" },

{ name: "Nina", age: 15, status: "unpaid" },

{ name: "Mike", age: 28, status: "paid" },

{ name: "Amit", age: 40, status: "rejected" },

{ name: "Riya", age: 60, status: "unpaid" }

]);

db.users.find().limit(5);

db.users.updateMany({ age: { $lt: 18 } }, { $set: { status: "rejected" } }); db.users.deleteOne({ name: "Akshata" });

db.users.deleteMany({ age: { $gt: 50 } });

db.users.find({ age: { $lt: 40 }, status: "paid" });

# TRIGGER

6 Write a database trigger on library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library\_Audit Table.

CREATE TABLE Library (

book\_id INT PRIMARY KEY AUTO\_INCREMENT, title VARCHAR(255),

author VARCHAR(255)

);

CREATE TABLE Library\_Audit (

audit\_id INT PRIMARY KEY AUTO\_INCREMENT,

operation VARCHAR(10), book\_id INT,

title VARCHAR(255),

author VARCHAR(255),

timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

INSERT INTO Library (title, author) VALUES

('Book 1', 'ABC'),

('Book 2', 'John'),

('Book 3', 'Alice'),

('Book 4', 'Bob'); DELIMITER $$

CREATE TRIGGER track\_library\_changes AFTER UPDATE OR DELETE ON Library FOR EACH ROW

BEGIN

DECLARE operation\_type VARCHAR(10);

-- Determine the type of operation (UPDATE or DELETE) IF (OLD.book\_id IS NOT NULL) THEN

IF (NEW.book\_id IS NOT NULL) THEN

SET operation\_type = 'UPDATE'; ELSE

SET operation\_type = 'DELETE'; END IF;

-- Insert the old record into the audit table for DELETE or UPDATE INSERT INTO Library\_Audit (operation, book\_id, title, author) VALUES (operation\_type, OLD.book\_id, OLD.title, OLD.author);

END IF;

END $$ DELIMITER ;

UPDATE Library

SET title = 'Updated Book 1', author = 'Updated ABC' WHERE book\_id = 1;

DELETE FROM Library WHERE book\_id = 2;

SELECT \* FROM Library\_Audit;

8.Create Customer(Cid, CustName, City), Product(Pid, ProductName, Qty) and Order(Oid, Cid, Pid, Qty) tables. 1) Insert data into tables. 2) Write a trigger to update count of product in Product table when customer successfully paced order for particular product. Hint: Order will be placed when its entry will be inserted into order table.

-- Creating Customer table CREATE TABLE Customer (

Cid INT PRIMARY KEY, CustName VARCHAR(100), City VARCHAR(100)

);

-- Inserting data into Customer table

INSERT INTO Customer (Cid, CustName, City) VALUES

(1, 'John Doe', 'New York'),

(2, 'Jane Smith', 'Los Angeles'),

(3, 'Alice Brown', 'Chicago');

-- Creating Product table CREATE TABLE Product (

Pid INT PRIMARY KEY,

ProductName VARCHAR(100), Qty INT

);

-- Inserting data into Product table

INSERT INTO Product (Pid, ProductName, Qty) VALUES

(1, 'Laptop', 50),

(2, 'Phone', 100),

(3, 'Headphones', 200);

-- Creating Order table CREATE TABLE Order (

Oid INT PRIMARY KEY, Cid INT,

Pid INT, Qty INT,

FOREIGN KEY (Cid) REFERENCES Customer(Cid), FOREIGN KEY (Pid) REFERENCES Product(Pid)

);

-- Inserting data into Order table INSERT INTO Order (Oid, Cid, Pid, Qty) VALUES

(1, 1, 1, 2),

(2, 2, 2, 3),

(3, 3, 3, 1); DELIMITER $$

CREATE TRIGGER update\_product\_qty AFTER INSERT ON Order

FOR EACH ROW BEGIN

-- Update the product quantity in Product table when an order is placed UPDATE Product

SET Qty = Qty - NEW.Qty WHERE Pid = NEW.Pid;

END $$ DELIMITER ;

-- Inserting a new order into the Order table INSERT INTO Order (Oid, Cid, Pid, Qty)

VALUES (4, 1, 2, 5); -- Customer with Cid 1 orders 5 units of Product with Pid 2 (Phone)

-- Checking the updated Product table SELECT \* FROM Product;

# PK AND FK

1. Identify primary keys and foreign keys for following database. Create tables and execute queries for given statements. employee(eid,ename,salary) assignment(projectid,eid) project(projectid,project\_name,manager) manager(eid,ename) i) Alter table to add address in employee table. ii) Display employee name and projects on which they are working/ iii)Display projectid, projectname and their managers. iv) Create view of employees working on 'Bank Management' project.

v) Print names of employees whose salary is greater than 40000 vi) Update salary of each employee with increase of Rs.2000

CREATE TABLE employee ( eid INT PRIMARY KEY, ename VARCHAR(50),

salary INT

);

CREATE TABLE manager ( eid INT PRIMARY KEY,

ename VARCHAR(50)

);

CREATE TABLE project ( projectid INT PRIMARY KEY, project\_name VARCHAR(100), manager INT,

FOREIGN KEY (manager) REFERENCES manager(eid)

);

CREATE TABLE assignment ( projectid INT,

eid INT,

PRIMARY KEY (projectid, eid),

FOREIGN KEY (projectid) REFERENCES project(projectid), FOREIGN KEY (eid) REFERENCES employee(eid)

);

1.

ALTER TABLE employee ADD address VARCHAR(255);

INSERT INTO employee (eid, ename, salary, address) VALUES (1, 'Alice', 45000, 'New York'),

(2, 'Bob', 38000, 'Los Angeles'),

(3, 'Charlie', 42000, 'Chicago'),

(4, 'David', 47000, 'Houston');

INSERT INTO manager (eid, ename) VALUES (1, 'Alice'),

(4, 'David');

INSERT INTO project (projectid, project\_name, manager) VALUES (101, 'Bank Management', 1),

(102, 'E-Commerce Platform', 4),

(103, 'Healthcare System', 1);

INSERT INTO assignment (projectid, eid) VALUES (101, 1),

(101, 2),

(102, 3),

(103, 4);

2.

SELECT e.ename, p.project\_name FROM employee e

JOIN assignment a ON e.eid = a.eid

JOIN project p ON a.projectid = p.projectid;

3.

SELECT p.projectid, p.project\_name, m.ename AS manager\_name FROM project p

JOIN manager m ON p.manager = m.eid;

4.

CREATE VIEW BankManagementEmployees AS SELECT e.eid, e.ename, e.salary, e.address FROM employee e

JOIN assignment a ON e.eid = a.eid

JOIN project p ON a.projectid = p.projectid WHERE p.project\_name = 'Bank Management';

-- Querying the View

SELECT \* FROM BankManagementEmployees;

5.

SELECT ename FROM employee

WHERE salary > 40000;

6.

UPDATE employee

SET salary = salary + 2000; SELECT \* FROM employee;

1. Identify primary keys and foreign keys for following database. Create tables and execute queries for given statements. employee(eid, ename, salary) assignment(projectid,eid) project(projectid,project\_name,manager) manager(eid,ename) i)Modify eid to use auto\_increment ii) Display Employees working in both projects 'Bank Management' and 'Content Management'. iii) Display average salary of organization. iv) Display employees who do not work on 'Bank Management' Project.

v) Delete employee whose id is 5. vi) Display employee having highest salary in oraganization.

CREATE TABLE employee (

eid INT AUTO\_INCREMENT PRIMARY KEY, ename VARCHAR(50),

salary INT

);

CREATE TABLE manager ( eid INT PRIMARY KEY,

ename VARCHAR(50)

);

CREATE TABLE project ( projectid INT PRIMARY KEY,

project\_name VARCHAR(100), manager INT,

FOREIGN KEY (manager) REFERENCES manager(eid)

);

CREATE TABLE assignment ( projectid INT,

eid INT,

PRIMARY KEY (projectid, eid),

FOREIGN KEY (projectid) REFERENCES project(projectid), FOREIGN KEY (eid) REFERENCES employee(eid)

);

INSERT INTO employee (ename, salary) VALUES ('Alice', 45000),

('Bob', 38000),

('Charlie', 42000),

('David', 47000),

('Eve', 39000);

INSERT INTO manager (eid, ename) VALUES (1, 'Alice'),

(4, 'David');

mysql> INSERT INTO project (projectid, project\_name, manager) VALUES (101, 'Bank Management', 1),

(102, 'Content Management', 4),

(103, 'E-Commerce', 1);

INSERT INTO assignment (projectid, eid) VALUES (101, 1),

(101, 2),

(102, 1),

(102, 3),

(103, 4),

(103, 5);

2.

SELECT e.ename FROM employee e

JOIN assignment a1 ON e.eid = a1.eid

JOIN project p1 ON a1.projectid = p1.projectid JOIN assignment a2 ON e.eid = a2.eid

JOIN project p2 ON a2.projectid = p2.projectid

WHERE p1.project\_name = 'Bank Management' AND p2.project\_name = 'Content Management';

3.

SELECT AVG(salary) AS average\_salary FROM employee;

4.

SELECT e.ename FROM employee e

WHERE e.eid NOT IN (

SELECT a.eid FROM assignment a

JOIN project p ON a.projectid = p.projectid WHERE p.project\_name = 'Bank Management'

);

6.

SELECT ename, salary FROM employee

WHERE salary = (SELECT MAX(salary) FROM employee);

5.

ALTER TABLE assignment

DROP FOREIGN KEY assignment\_ibfk\_2; ALTER TABLE assignment

ADD CONSTRAINT assignment\_ibfk\_2 FOREIGN KEY (eid)

REFERENCES employee (eid) ON DELETE CASCADE;

12 Identify primary keys and foreign keys for following database. Create tables and execute queries for given statements. supplier(supplierid,sname,saddress) parts(part\_id,part\_name,color); catalog(supplierid,part\_id,cost); 1)Find name of supplier who supply ‘green’ parts. 2)find name of suppliers who supply both blue and green parts. 3)Find supplier who supply all parts. 4)Fid total cost of red parts. 5) Find supplier who supply green parts with minimum cost. 6)Update color of part having part\_id = 4 and supplier\_id = 2

CREATE TABLE supplier ( supplierid INT PRIMARY KEY, sname VARCHAR(50),

saddress VARCHAR(100)

);

CREATE TABLE parts (

part\_id INT PRIMARY KEY, part\_name VARCHAR(50), color VARCHAR(20)

);

CREATE TABLE catalog ( supplierid INT,

part\_id INT,

cost DECIMAL(10,2),

PRIMARY KEY (supplierid, part\_id),

FOREIGN KEY (supplierid) REFERENCES supplier(supplierid), FOREIGN KEY (part\_id) REFERENCES parts(part\_id)

);

-- Insert sample data (you can replace with your own data)

INSERT INTO supplier VALUES (1, 'Supplier A', 'Address A'),

(2, 'Supplier B', 'Address B'),

(3, 'Supplier C', 'Address C');

INSERT INTO parts VALUES (1, 'Part 1', 'Red'),

(2, 'Part 2', 'Green'),

(3, 'Part 3', 'Blue');

INSERT INTO catalog VALUES (1, 1, 10.00),

(1, 2, 15.00),

(2, 2, 12.00),

(2, 3, 20.00),

(3, 1, 8.00),

(3, 2, 18.00);

1.

SELECT s.sname FROM supplier s

JOIN catalog c ON s.supplierid = c.supplierid JOIN parts p ON c.part\_id = p.part\_id WHERE p.color = 'green';

2.

SELECT s.sname FROM supplier s

JOIN catalog c ON s.supplierid = c.supplierid JOIN parts p ON c.part\_id = p.part\_id GROUP BY s.sname

HAVING SUM(p.color = 'blue') > 0 AND SUM(p.color = 'green') > 0;

3.

SELECT s.sname FROM supplier s

WHERE (SELECT COUNT(DISTINCT p.part\_id) FROM catalog c JOIN parts p ON c.part\_id = p.part\_id WHERE c.supplierid = s.supplierid) = (SELECT COUNT(\*) FROM parts);

4.

SELECT SUM(c.cost) AS total\_cost FROM catalog c

JOIN parts p ON c.part\_id = p.part\_id WHERE p.color = 'red';

5.

SELECT s.sname FROM supplier s

JOIN catalog c ON s.supplierid = c.supplierid JOIN parts p ON c.part\_id = p.part\_id WHERE p.color = 'green'

GROUP BY s.sname

HAVING MIN(c.cost) = (SELECT MIN(c.cost) FROM catalog c JOIN parts p ON c.part\_id = p.part\_id WHERE p.color = 'green');

6.

UPDATE parts SET color = 'yellow'

WHERE part\_id = 4 AND EXISTS (SELECT 1 FROM catalog c WHERE c.part\_id = 4 AND c.supplierid

= 2);

**Cursor**

1.Write a Stored Procedure namely proc\_Grade for the categorization of student. If marks scored by

students in examination is &lt;=1500 and marks&gt;=990 then student will be placed in distinction

category if marks scored are between 989 and900 category is first class, if marks 899 and 825

category is Higher Second Class Write a PL/SQL block for using procedure created with above

requirement. Stud\_Marks(Rollno,name, total\_marks) Result(Roll,Name, Class)

CREATE DATABASE student\_Result;

USE student\_Result;

CREATE TABLE Stud\_Marks (

Rollno INT PRIMARY KEY,

Name VARCHAR(255),

total\_marks INT

);

INSERT INTO Stud\_Marks VALUES

(101, "PRATIK", 1499),

(102, "PRIYA", 989),

(103, "MANOJ", 921);

CREATE TABLE Result (

roll INT PRIMARY KEY,

Name VARCHAR(255),

class VARCHAR(255)

);

DELIMITER //

CREATE PROCEDURE proc\_Grade()

BEGIN

DECLARE v\_Roll INT;

DECLARE v\_name VARCHAR(255);

DECLARE v\_marks INT;

DECLARE done INT DEFAULT 0;

DECLARE c1 CURSOR FOR SELECT Rollno, name, total\_marks FROM Stud\_Marks;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN c1;

read\_grade: LOOP

FETCH c1 INTO v\_Roll, v\_name, v\_marks;

IF done = 1 THEN

LEAVE read\_grade;

END IF;

IF v\_marks BETWEEN 990 AND 1500 THEN

INSERT INTO Result VALUES(v\_Roll, v\_name, "Destination");

ELSEIF v\_marks BETWEEN 900 AND 989 THEN

INSERT INTO Result VALUES(v\_Roll, v\_name, "First Class");

ELSEIF v\_marks BETWEEN 825 AND 899 THEN

INSERT INTO Result VALUES(v\_Roll, v\_name, "Higher Second Class");

END IF;

END LOOP read\_grade;

CLOSE c1;

END;

//

DELIMITER ;

CALL proc\_Grade();

SELECT \* FROM Result;

2.Write a Procedure code using cursor that will create the list of customer’s name from Customer

table.

CREATE DATABASE CustomerDBInfo;

USE CustomerDBInfo;

CREATE TABLE customerinfo (

c\_id INT PRIMARY KEY,

c\_name VARCHAR(255)

);

INSERT INTO customerinfo VALUES

(1, "PRATIK"),

(2, "RAHUL");

SELECT \* FROM customerinfo;

CREATE TABLE cust\_list (

c\_name VARCHAR(255)

);

DELIMITER //

CREATE PROCEDURE proc\_details()

BEGIN

DECLARE v\_name VARCHAR(255);

DECLARE done INT DEFAULT 0;

DECLARE c1 CURSOR FOR SELECT c\_name FROM customerinfo;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN c1;

read\_name: LOOP

FETCH c1 INTO v\_name;

IF done = 1 THEN

LEAVE read\_name;

END IF;

INSERT INTO cust\_list VALUES (v\_name);

END LOOP read\_name;

CLOSE c1;

END;

//

DELIMITER ;

CALL proc\_details();

SELECT \* FROM cust\_list;

3.Write a Procedure code using cursor that will merge the data available in newly created table

N\_RollCall with the data available in the O\_RollCall. If the data in the first table already exists in

the second table then that data should be skipped.  
  
CREATE TABLE O\_RollCall (

rollNo INT PRIMARY KEY,

Name VARCHAR(255)

);

INSERT INTO O\_RollCall VALUES

(1, "PRATIK"),

(2, "RAHUL");

SELECT \* FROM O\_RollCall;

CREATE TABLE N\_RollCall (

rollNo INT PRIMARY KEY,

Name VARCHAR(255)

);

-- Insert some data into N\_RollCall

INSERT INTO N\_RollCall VALUES

(2, "RAHUL"), -- Duplicate

(3, "SNEHA"), -- New entry

(4, "OMKAR"); -- New entry

SELECT \* FROM N\_RollCall;

DELIMITER //

CREATE PROCEDURE proc\_Skip()

BEGIN

DECLARE v\_roll INT;

DECLARE v\_name VARCHAR(255);

DECLARE done INT DEFAULT 0;

DECLARE c1 CURSOR FOR SELECT rollNo, Name FROM N\_RollCall;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN c1;

read\_details: LOOP

FETCH c1 INTO v\_roll, v\_name;

IF done = 1 THEN

LEAVE read\_details;

END IF;

-- Check for duplicate before inserting

IF NOT EXISTS (SELECT 1 FROM O\_RollCall WHERE rollNo = v\_roll) THEN

INSERT INTO O\_RollCall VALUES (v\_roll, v\_name);

END IF;

END LOOP read\_details;

CLOSE c1;

END;

//

DELIMITER ;

CALL proc\_Skip();