DBMS Question Bank Answers

* **Program 1:** Consider the relation employee (emp\_id, e\_name,salary ,Date of Joining,Dapt\_no,Designation) perform basic SQL operations.

1. Create the table employee

CREATE TABLE employee (

emp\_id INT PRIMARY KEY,

e\_name VARCHAR(100),

salary DECIMAL(10, 2),

Date\_of\_Joining DATE,

Dept\_no INT,

Designation VARCHAR(50)

);

1. Insert 10 records into the table

INSERT INTO employee (emp\_id, e\_name, salary, Date\_of\_Joining, Dept\_no, Designation) VALUES

(1, 'Alice Smith', 50000.00, '2020-01-15', 101, 'Manager'),

(2, 'Bob Johnson', 45000.00, '2019-03-22', 102, 'Developer'),

(3, 'Charlie Brown', 60000.00, '2021-06-12', 101, 'Analyst'),

(4, 'David Wilson', 55000.00, '2018-11-30', 103, 'Designer'),

(5, 'Eva Green', 47000.00, '2022-02-20', 102, 'Developer'),

(6, 'Frank Miller', 52000.00, '2020-07-08', 103, 'Manager'),

(7, 'Grace Lee', 48000.00, '2019-10-25', 101, 'Analyst'),

(8, 'Henry Adams', 53000.00, '2021-04-15', 102, 'Developer'),

(9, 'Ivy Taylor', 49000.00, '2018-09-14', 103, 'Designer'),

(10, 'Jack White', 51000.00, '2022-03-03', 101, 'Manager');

1. Create a view emp\_vl of the employee table

CREATE VIEW emp\_vl AS

SELECT emp\_id, e\_name, Dept\_no

FROM employee;

1. Create another view of the table (optional; if you want to create a different view, specify the attributes)

CREATE VIEW emp\_salary\_designation AS

SELECT emp\_id, e\_name, salary, Designation

FROM employee;

1. Update the department of any employee in the view and check if it gets updated

-- Update the department of employee with emp\_id = 2

UPDATE employee

SET Dept\_no = 104

WHERE emp\_id = 2;

-- Verify the update by selecting from the employee table and the view

SELECT \* FROM employee;

SELECT \* FROM emp\_vl;

1. **Create emp\_id as the primary key and show indices on the employee table**

The primary key was already defined when creating the table. To show indices:

SHOW INDEX FROM employee;

1. Show indices on the table

SHOW INDEX FROM employee;

1. Create a user-defined index on any column

CREATE INDEX idx\_employee\_name ON employee(e\_name);

-----------------------------------------------------------------------------------------------------

**Program 2:** Consider the relation employee (emp\_id,e\_name,salary ,Date of Joining,Dapt\_no,Designation) perform basic SQL operations.

1. Display employees whose name contains letter ‘e’:

SELECT e\_name

FROM employee

WHERE e\_name LIKE '%e%';

1. Display different types of designation:

SELECT DISTINCT Designation FROM Employee;

1. Display name and salary of employee whose location is Mumbai:

SELECT e\_name, salary

FROM employee

WHERE city = 'Mumbai';

1. Display name and department of employee working in Manager or Marketing department:

SELECT e\_name, Dapt\_no

FROM employee

WHERE Designation = 'Manager' OR Dapt\_no = 'Marketing';

1. Display the department name whose employees are more than one:

SELECT Dapt\_no

FROM employee

GROUP BY Dapt\_no

HAVING COUNT(\*) > 1;

1. Rename employee table as emp1:

RENAME TABLE employee TO emp1;

1. Add a new column city in the employee table:

ALTER TABLE emp1 ADD city VARCHAR(50);

----------------------------------------------------------------------------------------------------------------

Program 3: Consider the relation employee(emp\_id,e\_name,salary ,Date of Joining,Dapt\_no,Designation) perform basic SQL operations.

-- Create the employee table

CREATE TABLE employee (

emp\_id INT PRIMARY KEY,

e\_name VARCHAR(50),

salary DECIMAL(10, 2),

Date\_of\_Joining DATE,

Dapt\_no INT,

Designation VARCHAR(50)

);

-- Insert sample records into the employee table

INSERT INTO employee (emp\_id, e\_name, salary, Date\_of\_Joining, Dapt\_no, Designation) VALUES

(1, 'Alice', 50000, '2005-06-15', 1, 'Manager'),

(2, 'Bob', 45000, '2006-08-20', 1, 'Developer'),

(3, 'Charlie', 60000, '2004-12-01', 2, 'Market Analyst'),

(4, 'David', 55000, '2007-01-11', 3, 'Developer'),

(5, 'Eve', 70000, '2006-09-18', 2, 'Manager'),

(6, 'Frank', 40000, '2008-02-28', 4, 'Analyst'),

(7, 'Grace', 75000, '2009-11-13', 1, 'Manager'),

(8, 'Henry', 42000, '2006-10-05', 3, 'Developer'),

(9, 'Ivy', 47000, '2005-08-23', 1, 'Market Analyst'),

(10, 'John', 53000, '2004-04-09', 4, 'Intern');

-- 1. Find department in which maximum employees work.

SELECT Dapt\_no, COUNT(\*) AS employee\_count

FROM employee

GROUP BY Dapt\_no

ORDER BY employee\_count DESC

LIMIT 1;

-- 2. Display name, designation, and department no of employees whose name starts with either ‘A’ or ‘P’.

SELECT e\_name, Designation, Dapt\_no

FROM employee

WHERE e\_name LIKE 'A%' OR e\_name LIKE 'P%';

-- 3. Display max. salary from department 2 and min. salary from department 4.

SELECT

(SELECT MAX(salary) FROM employee WHERE Dapt\_no = 2) AS max\_salary\_dept\_2,

(SELECT MIN(salary) FROM employee WHERE Dapt\_no = 4) AS min\_salary\_dept\_4;

-- 4. Display employee data where salary is less than average salary from department 3.

SELECT \*

FROM employee

WHERE salary < (SELECT AVG(salary) FROM employee WHERE Dapt\_no = 3);

-- 5. Display employees who were hired earliest or latest.

SELECT \*

FROM employee

WHERE Date\_of\_Joining = (SELECT MIN(Date\_of\_Joining) FROM employee)

OR Date\_of\_Joining = (SELECT MAX(Date\_of\_Joining) FROM employee);

-- 6. Display name and department no of employees who are managers or market analysts (using predicates).

SELECT e\_name, Dapt\_no

FROM employee

WHERE Designation IN ('Manager', 'Market Analyst');

-- 7. List employees hired in August.

SELECT \*

FROM employee

WHERE MONTH(Date\_of\_Joining) = 8;

-- 8. List employees who are hired after 31/12/2006.

SELECT \*

FROM employee

WHERE Date\_of\_Joining > '2006-12-31';

-- 9. Find average annual salary per department.

SELECT Dapt\_no, AVG(salary \* 12) AS avg\_annual\_salary

FROM employee

GROUP BY Dapt\_no;

----------------------------------------------------------------------------------------------------------------

* Program 4: Consider two tables Customer(c\_id, c\_name , email , city , pincode)Order(order\_id , date , amount , cust\_id.

-- Step 1: Create the Customer and Order tables with primary key and foreign key constraints

CREATE TABLE Customer (

c\_id INT PRIMARY KEY,

c\_name VARCHAR(50),

email VARCHAR(100),

city VARCHAR(50),

pincode VARCHAR(10)

);

CREATE TABLE `Order` (

order\_id INT PRIMARY KEY,

date DATE,

amount DECIMAL(10, 2),

cust\_id INT,

FOREIGN KEY (cust\_id) REFERENCES Customer(c\_id)

);

-- Step 2: Insert 10 records into each table

INSERT INTO Customer (c\_id, c\_name, email, city, pincode) VALUES

(1, 'Alice', 'alice@example.com', 'New York', '10001'),

(2, 'Bob', 'bob@example.com', 'Los Angeles', '90001'),

(3, 'Charlie', 'charlie@example.com', 'Chicago', '60001'),

(4, 'David', 'david@example.com', 'Houston', '77001'),

(5, 'Eva', 'eva@example.com', 'Phoenix', '85001'),

(6, 'Frank', 'frank@example.com', 'Philadelphia', '19101'),

(7, 'Grace', 'grace@example.com', 'San Antonio', '78201'),

(8, 'Henry', 'henry@example.com', 'San Diego', '92101'),

(9, 'Ivy', 'ivy@example.com', 'Dallas', '75201'),

(10, 'Jack', 'jack@example.com', 'San Jose', '95101');

INSERT INTO `Order` (order\_id, date, amount, cust\_id) VALUES

(101, '2024-01-15', 250.00, 1),

(102, '2024-02-20', 500.00, 2),

(103, '2024-03-25', 750.00, 3),

(104, '2024-04-30', 1000.00, 2),

(105, '2024-05-10', 125.00, 4),

(106, '2024-06-15', 200.00, 5),

(107, '2024-07-20', 300.00, 6),

(108, '2024-08-25', 400.00, 7),

(109, '2024-09-30', 600.00, 8),

(110, '2024-10-05', 800.00, 1);

-- Step 3: Find all orders placed by customers with cust\_id = 2

SELECT \* FROM `Order`

WHERE cust\_id = 2;

-- Step 4: Find list of customers who placed their order and details of order

SELECT Customer.c\_id, Customer.c\_name, Customer.email, Customer.city, Customer.pincode,

`Order`.order\_id, `Order`.date, `Order`.amount

FROM Customer

JOIN `Order` ON Customer.c\_id = `Order`.cust\_id;

-- Step 5: List of customers who haven’t placed an order

SELECT \* FROM Customer

WHERE c\_id NOT IN (SELECT cust\_id FROM `Order`);

-- Step 6: List all orders and append to customer table

SELECT c\_id AS ID, c\_name AS Name, email AS Email, city AS City, pincode AS Pincode, NULL AS order\_id, NULL AS order\_date, NULL AS order\_amount

FROM Customer

UNION

SELECT cust\_id AS ID, NULL AS Name, NULL AS Email, NULL AS City, NULL AS Pincode, order\_id, date AS order\_date, amount AS order\_amount

FROM `Order`;

-- Step 7: Display all records from both tables separately

SELECT \* FROM Customer;

SELECT \* FROM `Order`;

-- Step 8: Display customers that are from the same city

SELECT A.c\_name AS Customer1, B.c\_name AS Customer2, A.city

FROM Customer A

JOIN Customer B ON A.city = B.city AND A.c\_id < B.c\_id;

Program 5: Consider tables Borrower (RollNo, Name, DateofIssue, NameofBook, Status) and

Fine (Roll\_no,Date,Amt). Status is either Issued or Returned.

Step 1: Create the Tables

CREATE TABLE Borrower (

RollNo INT PRIMARY KEY,

Name VARCHAR(50),

DateofIssue DATE,

NameofBook VARCHAR(100),

Status VARCHAR(10) CHECK (Status IN ('Issued', 'Returned'))

);

CREATE TABLE Fine (

Roll\_no INT,

Date DATE,

Amt DECIMAL(10, 2),

PRIMARY KEY (Roll\_no, Date),

FOREIGN KEY (Roll\_no) REFERENCES Borrower(RollNo)

);

Step 2: Insert 10 Records into Each Table

-- Inserting into Borrower table

INSERT INTO Borrower (RollNo, Name, DateofIssue, NameofBook, Status) VALUES

(1, 'Alice', '2024-01-10', 'Database Systems', 'Issued'),

(2, 'Bob', '2024-01-12', 'Operating Systems', 'Returned'),

(3, 'Charlie', '2024-01-15', 'Data Structures', 'Issued'),

(4, 'David', '2024-01-20', 'Algorithms', 'Returned'),

(5, 'Eve', '2024-01-25', 'Machine Learning', 'Issued'),

(6, 'Frank', '2024-01-28', 'Computer Networks', 'Issued'),

(7, 'Grace', '2024-02-01', 'Artificial Intelligence', 'Returned'),

(8, 'Hannah', '2024-02-05', 'Cybersecurity', 'Issued'),

(9, 'Isaac', '2024-02-10', 'Big Data Analytics', 'Returned'),

(10, 'John', '2024-02-15', 'Cloud Computing', 'Issued');

-- Inserting into Fine table

INSERT INTO Fine (Roll\_no, Date, Amt) VALUES

(1, '2024-01-20', 50.00),

(2, '2024-01-25', 30.00),

(3, '2024-02-01', 40.00),

(4, '2024-02-05', 25.00),

(5, '2024-02-10', 60.00),

(6, '2024-02-15', 35.00),

(7, '2024-02-18', 20.00),

(8, '2024-02-20', 55.00),

(9, '2024-02-22', 45.00),

(10, '2024-02-25', 50.00);

Step 3: Find the Count of Books with Issued Status

SELECT COUNT(\*) AS IssuedBooksCount

FROM Borrower

WHERE Status = 'Issued';

Step 4: Display All Records

-- Display all records from Borrower table

SELECT \* FROM Borrower;

-- Display all records from Fine table

SELECT \* FROM Fine;

Step 5: Display RollNo Whose Date of Issue Is the Same

SELECT RollNo, DateofIssue

FROM Borrower

GROUP BY DateofIssue

HAVING COUNT(DateofIssue) > 1;

* Program 6: Consider student (roll\_no, name, marks, class) table. Column roll\_no is primary key. Perform any 3 DLL and any 3 DML operations on the table.

**DDL Operations (Data Definition Language):**

1. **Create the student table:**

SQL

CREATE TABLE student (

roll\_no INT PRIMARY KEY,

name VARCHAR(50),

marks INT,

class INT

);

1. **Add a new column email to the student table:**

SQL

ALTER TABLE student

ADD COLUMN email VARCHAR(100);

1. **Drop the email column from the student table:**

SQL

ALTER TABLE student

DROP COLUMN email;

**DML Operations (Data Manipulation Language):**

1. **Insert a new record into the student table:**

SQL

INSERT INTO student (roll\_no, name, marks, class)

VALUES (101, 'Alice', 95, 10);

1. **Update the marks of a student with roll\_no 101:**

SQL

UPDATE student

SET marks = 98

WHERE roll\_no = 101;

1. **Delete a student record with roll\_no 102:**

SQL

DELETE FROM student

WHERE roll\_no = 102;

* Program 7

-- Step 1: Create the jobs table

CREATE TABLE jobs (

job\_id INT PRIMARY KEY,

job\_title VARCHAR(50),

min\_sal DECIMAL(10, 2),

max\_sal DECIMAL(10, 2)

);

-- Step 2: Create the job\_history table with constraints

CREATE TABLE job\_history (

employee\_id INT UNIQUE, -- Unique constraint on employee\_id

start\_date DATE,

end\_date DATE,

job\_id INT,

department\_id INT,

FOREIGN KEY (job\_id) REFERENCES jobs(job\_id) -- Foreign key reference to jobs table

);

Program 8: For the given relation schema: employee(employee-name, street, city)

works (employee-name, company-name, salary)

company (company-name, city)

manages (employee-name, manager-name)

Give an expression in SQL for each of the following queries:

CREATE DATABASE employee\_db;

USE employee\_db;

CREATE TABLE employee (

employee\_name VARCHAR(50) PRIMARY KEY,

street VARCHAR(100),

city VARCHAR(50)

);

CREATE TABLE works (

employee\_name VARCHAR(50),

company\_name VARCHAR(50),

salary DECIMAL(10,2),

PRIMARY KEY (employee\_name, company\_name),

FOREIGN KEY (employee\_name) REFERENCES employee(employee\_name),

FOREIGN KEY (company\_name) REFERENCES company(company\_name)

);

CREATE TABLE company (

company\_name VARCHAR(50) PRIMARY KEY,

city VARCHAR(50)

);

CREATE TABLE manages (

employee\_name VARCHAR(50),

manager\_name VARCHAR(50),

PRIMARY KEY (employee\_name, manager\_name),

FOREIGN KEY (employee\_name) REFERENCES employee(employee\_name),

FOREIGN KEY (manager\_name) REFERENCES employee(employee\_name)

);

-- Inserting sample data

INSERT INTO employee VALUES

('Alice', '123 Main St', 'New York'),

('Bob', '456 Elm St', 'Los Angeles'),

('Charlie', '789 Oak St', 'Chicago'),

('David', '101 Pine St', 'New York'),

('Eve', '202 Cedar St', 'Los Angeles');

INSERT INTO works VALUES

('Alice', 'CompanyA', 50000),

('Bob', 'CompanyA', 60000),

('Charlie', 'CompanyB', 40000),

('David', 'CompanyC', 70000),

('Eve', 'CompanyA', 80000);

INSERT INTO company VALUES

('CompanyA', 'New York'),

('CompanyB', 'Chicago'),

('CompanyC', 'Los Angeles');

INSERT INTO manages VALUES

('Bob', 'Alice'),

('Charlie', 'Alice'),

('David', 'Bob'),

('Eve', 'Bob');

-- Query 1: Find employees working for the same company and earning more than $10,000

SELECT e1.employee\_name, e1.street, e1.city

FROM employee e1, employee e2, works w1, works w2

WHERE e1.employee\_name = w1.employee\_name

AND e2.employee\_name = w2.employee\_name

AND w1.company\_name = w2.company\_name

AND w1.salary > 10000

AND w2.salary > 10000

AND e1.employee\_name <> e2.employee\_name;

-- Query 2: Find employees living in the same city as their company

SELECT e.employee\_name

FROM employee e, works w, company c

WHERE e.employee\_name = w.employee\_name

AND w.company\_name = c.company\_name

AND e.city = c.city;

-- Query 3: Find employees earning more than the average salary of their company

SELECT e.employee\_name

FROM employee e, works w

WHERE e.employee\_name = w.employee\_name

AND w.salary > (

SELECT AVG(salary)

FROM works w2

WHERE w2.company\_name = w.company\_name

);

Program 9: For the given relation schema: employee(employee-name, street, city)

works (employee-name, company-name, salary)

company (company-name, city) , manages (employee-name, manager-name)

-- Step 1: Create Tables

-- Create the employee table

CREATE TABLE employee (

employee\_name VARCHAR(100) PRIMARY KEY,

street VARCHAR(100),

city VARCHAR(100)

);

-- Create the works table

CREATE TABLE works (

employee\_name VARCHAR(100),

company\_name VARCHAR(100),

salary DECIMAL(10, 2),

FOREIGN KEY (employee\_name) REFERENCES employee(employee\_name)

);

-- Create the company table

CREATE TABLE company (

company\_name VARCHAR(100) PRIMARY KEY,

city VARCHAR(100)

);

-- Create the manages table

CREATE TABLE manages (

employee\_name VARCHAR(100),

manager\_name VARCHAR(100),

FOREIGN KEY (employee\_name) REFERENCES employee(employee\_name),

FOREIGN KEY (manager\_name) REFERENCES employee(employee\_name)

);

-- Step 2: Insert Sample Data

-- Insert data into employee table

INSERT INTO employee (employee\_name, street, city) VALUES

('Alice', '123 Maple St', 'Springfield'),

('Bob', '456 Oak St', 'Springfield'),

('Charlie', '789 Pine St', 'Shelbyville'),

('David', '123 Maple St', 'Springfield'),

('Eve', '456 Oak St', 'Shelbyville');

-- Insert data into works table

INSERT INTO works (employee\_name, company\_name, salary) VALUES

('Alice', 'TechCorp', 60000),

('Bob', 'TechCorp', 55000),

('Charlie', 'BizInc', 50000),

('David', 'BizInc', 62000),

('Eve', 'WebSolutions', 70000);

-- Insert data into company table

INSERT INTO company (company\_name, city) VALUES

('TechCorp', 'Springfield'),

('BizInc', 'Shelbyville'),

('WebSolutions', 'Springfield');

-- Insert data into manages table

INSERT INTO manages (employee\_name, manager\_name) VALUES

('Alice', 'Bob'),

('Bob', 'David'),

('Charlie', 'David'),

('David', 'Alice'),

('Eve', 'Charlie');

-- Step 3: Queries

-- Query a: Find the name of the company that has the smallest payroll

SELECT company\_name

FROM works

GROUP BY company\_name

ORDER BY SUM(salary) ASC

LIMIT 1;

-- Query b: Find the names of all employees who live in the same cities and on the same streets as their managers

SELECT e1.employee\_name

FROM employee e1

JOIN manages m ON e1.employee\_name = m.employee\_name

JOIN employee e2 ON m.manager\_name = e2.employee\_name

WHERE e1.street = e2.street AND e1.city = e2.city;

Program 10: Implement CRUD operations. SAVE method. Use following Collection. Perform Map Reduce to count quantity of each item. Item: Item ID, Item quantity, price, brand.

// Step 1: Create the Item collection

db.createCollection("Item");

// Step 2: Insert initial data into the Item collection

db.Item.insertMany([

{ Item\_ID: 1, Item\_quantity: 10, price: 150, brand: "BrandA" },

{ Item\_ID: 2, Item\_quantity: 5, price: 100, brand: "BrandB" },

{ Item\_ID: 3, Item\_quantity: 20, price: 250, brand: "BrandA" },

{ Item\_ID: 4, Item\_quantity: 15, price: 300, brand: "BrandC" },

{ Item\_ID: 5, Item\_quantity: 8, price: 200, brand: "BrandB" }

]);

// Step 3: CRUD Operations

// Create

db.Item.insertOne({

Item\_ID: 6,

Item\_quantity: 12,

price: 120,

brand: "BrandD"

});

db.Item.insertMany([

{ Item\_ID: 7, Item\_quantity: 7, price: 80, brand: "BrandE" },

{ Item\_ID: 8, Item\_quantity: 14, price: 220, brand: "BrandF" }

]);

// Read

db.Item.find().pretty();

db.Item.find({ Item\_ID: 1 }).pretty();

// Update

db.Item.updateOne(

{ Item\_ID: 1 },

{ $set: { Item\_quantity: 15 } }

);

db.Item.updateMany(

{ brand: "BrandB" },

{ $set: { price: 110 } }

);

// Delete

db.Item.deleteOne({ Item\_ID: 6 });

db.Item.deleteMany({ brand: "BrandE" });

END HERE FOR PROGRAM 11…………

// Step 4: MapReduce to count quantity of each item

var mapFunction = function() {

emit(this.brand, this.Item\_quantity);

};

var reduceFunction = function(key, values) {

return Array.sum(values);

};

db.Item.mapReduce(

mapFunction,

reduceFunction,

{

out: "item\_quantity\_count" // Output collection

}

);

// Step 5: View the results of MapReduce

db.item\_quantity\_count.find().pretty();

Program 12: Implement CRUD operations. SAVE method. Use following Collection.

Item: Item ID, Item quantity, price, brand, discount

// Step 1: Insert sample data into the Item collection

db.Item.insertMany([

{ Item\_ID: 1, Item\_quantity: 10, price: 150, brand: "BrandA", discount: 10 },

{ Item\_ID: 2, Item\_quantity: 5, price: 100, brand: "BrandB", discount: 5 },

{ Item\_ID: 3, Item\_quantity: 20, price: 250, brand: "BrandA", discount: 15 },

{ Item\_ID: 4, Item\_quantity: 15, price: 300, brand: "BrandC", discount: 20 },

{ Item\_ID: 5, Item\_quantity: 8, price: 200, brand: "BrandB", discount: 10 },

{ Item\_ID: 6, Item\_quantity: 12, price: 120, brand: "BrandD", discount: 5 }

]);

// Step 2: CRUD Operations

// Create

db.Item.insertOne({

Item\_ID: 7,

Item\_quantity: 6,

price: 110,

brand: "BrandA",

discount: 7

});

// Read

db.Item.find().pretty();

db.Item.find({ Item\_ID: 1 }).pretty();

// Update

db.Item.updateOne(

{ Item\_ID: 2 },

{ $set: { Item\_quantity: 10, price: 105 } }

);

// Delete

db.Item.deleteOne({ Item\_ID: 7 });

db.Item.deleteMany({ brand: "BrandC" });

// Step 3: Specific Queries

// 1. Display the count of items by brand

db.Item.aggregate([

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

]);

// 2. Display item with minimum price

db.Item.find().sort({ price: 1 }).limit(1);

// 3. Display maximum discount given for any item

db.Item

Program 13: Implement Map reduces operation for counting the marks of students.

Use: student (roll\_no, name marks, class)

Expected output: student name or roll no and total marks.

// Step 1: Insert sample data into the student collection

db.student.insertMany([

{ roll\_no: 1, name: "Alice", marks: 85, class: "10A" },

{ roll\_no: 2, name: "Bob", marks: 75, class: "10A" },

{ roll\_no: 1, name: "Alice", marks: 90, class: "10A" },

{ roll\_no: 3, name: "Charlie", marks: 82, class: "10B" },

{ roll\_no: 2, name: "Bob", marks: 88, class: "10A" },

{ roll\_no: 4, name: "David", marks: 95, class: "10B" },

{ roll\_no: 1, name: "Alice", marks: 78, class: "10A" },

{ roll\_no: 3, name: "Charlie", marks: 91, class: "10B" }

]);

// Step 2: Define MapReduce functions

// Map function

var mapFunction = function() {

emit(this.roll\_no, this.marks);

};

// Reduce function

var reduceFunction = function(key, values) {

return Array.sum(values);

};

// Step 3: Execute MapReduce to calculate total marks for each student

db.student.mapReduce(

mapFunction,

reduceFunction,

{ out: "total\_marks" }

);

// Step 4: Display results

db.total\_marks.find().pretty();

Program 14: Implement Map reduces operation for displaying persons with same profession.

Use: person (person\_id, name, addr, profession)

// Step 1: Insert sample data into the person collection

db.person.insertMany([

{ person\_id: 1, name: "Alice", addr: "123 Maple St", profession: "Engineer" },

{ person\_id: 2, name: "Bob", addr: "456 Oak St", profession: "Doctor" },

{ person\_id: 3, name: "Charlie", addr: "789 Pine St", profession: "Engineer" },

{ person\_id: 4, name: "David", addr: "101 Birch St", profession: "Teacher" },

{ person\_id: 5, name: "Eve", addr: "202 Cedar St", profession: "Doctor" },

{ person\_id: 6, name: "Frank", addr: "303 Elm St", profession: "Engineer" }

]);

// Step 2: Define MapReduce functions

// Map function

var mapFunction = function() {

emit(this.profession, { names: [this.name], addresses: [this.addr] });

};

// Reduce function

var reduceFunction = function(key, values) {

var result = { names: [], addresses: [] };

values.forEach(function(value) {

result.names = result.names.concat(value.names);

result.addresses = result.addresses.concat(value.addresses);

});

return result;

};

// Step 3: Execute MapReduce to group persons by profession

db.person.mapReduce(

mapFunction,

reduceFunction,

{ out: "profession\_groups" }

);

// Step 4: Display results

db.profession\_groups.find().pretty();

Program 15: Perform CRUD operation in mongo db –

Use : person( person\_id, name, addr, profession )

// Connect to MongoDB

use myDatabase

// Create a Collection

db.createCollection("persons")

// Insert Data

db.persons.insertOne({

person\_id: 1,

name: "Alice",

addr: "123 Main St",

profession: "Engineer"

})

db.persons.insertMany([

{ person\_id: 2, name: "Bob", addr: "456 Elm St", profession: "Doctor" },

{ person\_id: 3, name: "Charlie", addr: "789 Oak St", profession: "Teacher" }

])

// Read Data

// Find all documents

db.persons.find()

// Find a specific document

db.persons.findOne({ person\_id: 2 })

// Find documents with a specific profession

db.persons.find({ profession: "Engineer" })

// Update a document using the `updateOne` method

db.persons.updateOne({ person\_id: 2 }, { $set: { addr: "987 Pine St" } })

// Update multiple documents using the `updateMany` method

db.persons.updateMany({ profession: "Teacher" }, { $set: { profession: "Professor" } })

// Delete a specific document

db.persons.deleteOne({ person\_id: 3 })

// Delete multiple documents

db.persons.deleteMany({ profession: "Professor" })

Program 16: Perform CRUD operation and Aggregation in mongo db

employee(emp\_id,e\_name,salary ,Date of Joining,Dapt\_no,Designation)

1. Display the count of employee department wise.

2. Dsiplay the average salary of employee in sales department.

3. Dsiplay minimum salary to employees joins in June 2016

4. Display maximum salary given to employee in production department.

5. Display record of first and last employee department wise.

use myDatabase

// Create a Collection

db.createCollection("employees")

// Insert Data

db.employees.insertMany([

{ emp\_id: 1, e\_name: "Alice", salary: 50000, DOJ: ISODate("2016-06-15"), dept\_no: 10, designation: "Manager" },

{ emp\_id: 2, e\_name: "Bob", salary: 30000, DOJ: ISODate("2017-01-10"), dept\_no: 20, designation: "Engineer" },

{ emp\_id: 3, name: "Charlie", salary: 45000, DOJ: ISODate("2016-07-12"), dept\_no: 10, designation: "Analyst" },

{ emp\_id: 4, name: "David", salary: 60000, DOJ: ISODate("2015-12-20"), dept\_no: 20, designation: "Senior Engineer" },

{ emp\_id: 5, name: "Eve", salary: 35000, DOJ: ISODate("2016-06-25"), dept\_no: 10, designation: "Assistant Manager" }

])

// Count of employees department-wise

db.employees.aggregate([

{ $group: { \_id: "$dept\_no", count: { $sum: 1 } } }])

// Average salary of employees in the sales department

db.employees.aggregate([

{ $match: { dept\_no: 20 } },

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

])

// Minimum salary of employees joining in June 2016

db.employees.aggregate([

{ $match: { DOJ: { $gte: ISODate("2016-06-01"), $lt: ISODate("2016-07-01") } } } },

{ $group: { \_id: null, min\_salary: { $min: "$salary" } } }

])

// Maximum salary given to employees in the production department

db.employees.aggregate([

{ $match: { dept\_no: 10 } },

{ $group: { \_id: null, max\_salary: { $max: "$salary" } } }

])

// First and last employee department-wise

db.employees.aggregate([

{ $sort: { dept\_no: 1, emp\_id: 1 } },

{ $group: { \_id: "$dept\_no", first: { $first: "$$ROOT" }, last: { $last: "$$ROOT" } } }

])

**Program 18:** Implement Stored Procedure namely proc\_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=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(name, total\_marks) Result(Roll,Name, Class).

**create database Score;**

**use Score;**

**create table stud\_marks(name varchar(20),total\_marks int(5));**

**create table Result(roll\_no int(3) primary key,name varchar(20),class varchar(20));**

**insert into stud\_marks values('Suresh',995);**

**insert into stud\_marks values('Harish',865);**

**insert into stud\_marks values('Samart',920);**

**insert into stud\_marks values('Mohan',1000);**

**insert into stud\_marks values('Soham',745);**

**select \* from stud\_marks;**

**insert into Result(roll\_no,Name) values(1,'Suresh');**

**insert into Result(roll\_no,Name) values(2,'Harish');**

**insert into Result(roll\_no,Name) values(3,'Samart');**

**insert into Result(roll\_no,Name) values(4,'Mohan');**

**insert into Result(roll\_no,Name) values(5,'Soham');**

**select \* from Result;**

**delimiter //**

**create procedure proc\_Grade(in r int(2),out grade char(25))**

**begin**

**declare m int(4);**

**select total\_marks into m from stud\_marks where name= (select name from Result where roll\_no=r);**

**if m>=990 and m<=1500 then**

**select 'Distinction' into grade;**

**update Result set Class='Distinction' where Roll\_no=r;**

**elseif m>=900 and m<=989 then**

**select 'FirstClass' into grade;**

**update Result set Class='FirstClass' where Roll\_no=r;**

**elseif m>=825 and m<=899 then**

**select 'SecondClass' into grade;**

**update Result set Class='SecondClass' where Roll\_no=r;**

**else**

**select '--' into grade;**

**update Result set Class='--' where Roll\_no=r;**

**end if;**

**end //**

**delimiter //**

**create function func\_Grade(r int(2))**

**returns varchar(25)**

**deterministic**

**begin**

**declare grade varchar(25);**

**call proc\_Grade(r,grade);**

**return grade;**

**end //**

**select func\_Grade(1); //**

**select func\_Grade(2); //**

**select func\_Grade(3); //**

**select func\_Grade(4); //**

**select func\_Grade(5); //**

**select \* from Result; //**

* **Program 19:** Write a database trigger on customer( cust\_id, c\_name, addr) 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 cust\_Audit table.

-- Step 1: Create the customer table

CREATE TABLE customer (

cust\_id INT PRIMARY KEY,

c\_name VARCHAR(255),

addr VARCHAR(255)

);

-- Step 2: Create the cust\_Audit table to store audit records

CREATE TABLE cust\_Audit (

audit\_id INT PRIMARY KEY AUTO\_INCREMENT,

cust\_id INT,

c\_name VARCHAR(255),

addr VARCHAR(255),

action\_type VARCHAR(10), -- "UPDATE" or "DELETE"

action\_timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP);

-- Step 3: Create a trigger to track updates on the customer table

DELIMITER //

CREATE TRIGGER customer\_update\_audit

BEFORE UPDATE ON customer

FOR EACH ROW

BEGIN

INSERT INTO cust\_Audit (cust\_id, c\_name, addr, action\_type, action\_timestamp)

VALUES (OLD.cust\_id, OLD.c\_name, OLD.addr, 'UPDATE', NOW());

END;

//

-- Step 4: Create a trigger to track deletions on the customer table

CREATE TRIGGER customer\_delete\_audit

BEFORE DELETE ON customer

FOR EACH ROW

BEGIN

INSERT INTO cust\_Audit (cust\_id, c\_name, addr, action\_type, action\_timestamp)

VALUES (OLD.cust\_id, OLD.c\_name, OLD.addr, 'DELETE', NOW());

END;

//

DELIMITER ;

INSERT INTO customer (cust\_id, c\_name, addr) VALUES (1, 'John Doe', '123 Elm Street');

INSERT INTO customer (cust\_id, c\_name, addr) VALUES (2, 'Jane Smith', '456 Oak Avenue');

UPDATE customer SET addr = '789 Maple Street' WHERE cust\_id = 1;

DELETE FROM customer WHERE cust\_id = 2;

SELECT \* FROM cust\_Audit;

--------------------------------------------------------------------------------------------------------------

* Program 20

Same as Program 19

* Program 21: Implement a PL/SQL block of code using explicit Cursor, that will merge the data available in the newly created table N\_RollCall with the data available in the table O\_RollCall. If the data in the first table already exist in the second table then that data should be skipped.

> create table new\_roll(roll int,name varchar(10));

Table created.

> create table old\_roll(roll int,name varchar(10));

Table created.

SQL> insert into new\_roll values(2,'b');

1 row created.

SQL> insert into old\_roll values(4,'d'); 1 row created.

SQL> insert into old\_roll values(3,'bcd'); 1 row created.

SQL> insert into old\_roll values(1,'bc'); 1 row created.

SQL> insert into old\_roll values(5,'bch'); 1 row created.

SQL> insert into new\_roll values(5,'bch'); 1 row created.

SQL> insert into new\_roll values(1,'bc'); 1 row created.

SQL> select \* from new\_roll; ROLL NAME ---------- ---------- 2 b 5 bch 1 bc

SQL> select \* from old\_roll; ROLL NAME ---------- ---------- 4 d 3 bcd 1 bc 5 bch

SQL> CREATE OR REPLACE PROCEDURE roll1\_list AS

2 a INT;

3 a1 VARCHAR2(10);

4 b INT;

5 b1 VARCHAR2(10);

6 CURSOR c1 IS SELECT roll, name FROM old\_roll;

7 CURSOR c2 IS SELECT roll, name FROM new\_roll;

8 BEGIN

9 OPEN c1;

10 OPEN c2;

11 12 LOOP

13 FETCH c1 INTO a, a1;

14 EXIT WHEN c1%NOTFOUND;

15 16 -- Check if a record with the same 'roll' exists in new\_roll

17 BEGIN

18 SELECT roll INTO b FROM new\_roll WHERE roll = a;

19 EXCEPTION

20 WHEN NO\_DATA\_FOUND THEN

21 -- If no matching record found, insert into new\_roll

22 INSERT INTO new\_roll (roll, name) VALUES (a, a1);

23 END;

24 END LOOP;

25 26 CLOSE c1; 27 CLOSE c2;

28 29 -- Commit the transaction to save changes permanently

30 COMMIT; 31 END; 32 / Procedure created.

SQL> call roll1\_list(); Call completed.

SQL> select \* from new\_roll; ROLL NAME ---------- ---------- 2 b 5 bch 1 bc 4 d 3 bcd

**Program 22:** Write a PL/SQL block of code for the following requirements:- Schema: Borrower(Rollin, Name, DateofIssue, NameofBook, Status) 2. Fine(Roll\_no,Date,Amt) • Accept roll\_no & name of book from user. • Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day. If condition of fine is true, then details will be stored into fine table.

CREATE TABLE borrower(roll\_no NUMBER , name VARCHAR2(25), dateofissue DATE, name\_of\_book VARCHAR2(25), status VARCHAR2(20));

Table created.

Cancel TO\_DATE

> INSERT INTO borrower VALUES(45,'ASHUTOSH',TO\_DATE('01-08-2022','DD-MM YYYY'),'HARRY POTTER','PENDING'); 1 row created.

> INSERT INTO borrower VALUES(46,'ARYAN',TO\_DATE('15-08-2022','DD-MM YYYY'),'DARK MATTER','PENDING');

1 row created.

> INSERT INTO borrower VALUES(47,'ROHAN',TO\_DATE('24-08-2022','DD-MM YYYY'),'SILENT HILL','PENDING'); 1 row created.

> INSERT INTO borrower VALUES(48,'SANKET',TO\_DATE('26-08-2022','DD-MM YYYY'),'GOD OF WAR','PENDING'); 1 row created.

> INSERT INTO borrower VALUES(49,'SARTHAK',TO\_DATE('09-09-2022','DD-MM YYYY'),'SPIDER-MAN','PENDING'); 1 row created.

> CREATE TABLE fine ( 2 roll\_no NUMBER, 3 return\_date DATE, 4 fine NUMBER 5 ); Table created.

> DECLARE

2 i\_roll\_no NUMBER;

3 name\_of\_book VARCHAR2(25);

4 no\_of\_days NUMBER;

5 return\_date DATE := TO\_DATE(SYSDATE,'DD-MM-YYYY');

6 temp NUMBER;

7 doi DATE;

8 fine NUMBER;

9 BEGIN

10 i\_roll\_no := &i\_roll\_no;

11 name\_of\_book := '&nameofbook';

12 dbms\_output.put\_line(return\_date);

13 SELECT to\_date(borrower.dateofissue,'DD-MM-YYYY') INTO doi FROM borrower WHERE borrower.roll\_no = i\_roll\_no AND borrower.name\_of\_book = name\_of\_book; 14 no\_of\_days := return\_date-doi;

15 dbms\_output.put\_line(no\_of\_days);

16 IF (no\_of\_days >15 AND no\_of\_days <=30) THEN

17 fine := 5\*no\_of\_days;

18 ELSIF (no\_of\_days>30 ) THEN

19 temp := no\_of\_days-30;

20 fine := 150 + temp\*50;

21 END IF;

22 dbms\_output.put\_line(fine);

23 INSERT INTO fine VALUES(i\_roll\_no,return\_date,fine);

24 UPDATE borrower SET status = 'RETURNED' WHERE borrower.roll\_no = i\_roll\_no;

25 END;

26 /

Enter value for i\_roll\_no: 46 Enter value for nameofbook: DARK MATTER 02-OCT-23 413 19300 PL/SQL procedure successfully completed.

> select \* from BORROWER;

ROLL\_NO NAME DATEOFISS NAME\_OF\_BOOK ---------- ------------------------- --------- ------------------------- STATUS -------------------- 45 ASHUTOSH PENDING 01-AUG-22 HARRY POTTER 46 ARYAN 15-AUG-22 DARK MATTER RETURNED 47 ROHAN 24-AUG-22 SILENT HILL PENDING ROLL\_NO NAME DATEOFISS NAME\_OF\_BOOK ---------- ------------------------- --------- ------------------------- STATUS -------------------- 48 SANKET 26-AUG-22 GOD OF WAR PENDING PENDING 49 SARTHAK

> select \* from FINE;

ROLL\_NO RETURN\_DA FINE ---------- --------- ---------- 46 02-OCT-23 19300

Program 23: **)** Implement Basic SQL queries.

1. Create table employee.

2. Insert 10 records in table.

3. Create a view emp\_vl of table employee which has emp\_id , name and dept-attributes.

4. Display name and department of employee working in Manager or Marketing department

5. Display employees who were hired earliest or latest.

6. Display name and department no of employees who are manager, market analysts. Use

Predicates

List employees hired in August.

List employees who are hired after 31/12/2006.

-- 1. Create the `employee` Table

CREATE TABLE employee (

emp\_id INT PRIMARY KEY,

name VARCHAR(100),

dept VARCHAR(50),

position VARCHAR(50),

hire\_date DATE,

dept\_no INT

);

-- 2. Insert 10 Records into the `employee` Table

INSERT INTO employee (emp\_id, name, dept, position, hire\_date, dept\_no) VALUES

(1, 'Alice Johnson', 'Marketing', 'Market Analyst', '2005-07-15', 101),

(2, 'Bob Smith', 'Sales', 'Sales Manager', '2008-09-12', 102),

(3, 'Charlie Brown', 'Finance', 'Accountant', '2010-03-22', 103),

(4, 'Daisy Lee', 'HR', 'HR Manager', '2007-12-18', 104),

(5, 'Evan Davis', 'Marketing', 'Marketing Executive', '2009-06-10', 101),

(6, 'Fay Turner', 'Engineering', 'Engineer', '2006-04-14', 105),

(7, 'George White', 'Marketing', 'Marketing Manager', '2012-11-05', 101),

(8, 'Holly Black', 'Finance', 'Financial Analyst', '2015-01-30', 103),

(9, 'Ivy Green', 'Sales', 'Market Analyst', '2017-08-25', 102),

(10, 'Jack Brown', 'Engineering', 'Engineer', '2020-05-20', 105);

-- 3. Create a View `emp\_vl` with `emp\_id`, `name`, and `dept` Attributes

CREATE VIEW emp\_vl AS

SELECT emp\_id, name, dept FROM employee;

-- 4. Display `name` and `department` of Employees Working in `Manager` or `Marketing` Department

SELECT name, dept FROM employee

WHERE position LIKE '%Manager%' OR dept = 'Marketing';

-- 5. Display Employees Who Were Hired Earliest or Latest

SELECT \* FROM employee

WHERE hire\_date = (SELECT MIN(hire\_date) FROM employee) OR hire\_date = (SELECT MAX(hire\_date) FROM employee);

-- 6. Display `name` and `dept\_no` of Employees Who Are Managers or Market Analysts

SELECT name, dept\_no FROM employee

WHERE position IN ('Manager', 'Market Analyst');

-- 7. List Employees Hired in August

SELECT name, dept, hire\_date FROM employee

WHERE MONTH(hire\_date) = 8;

-- 8. List Employees Who Were Hired After December 31, 2006

SELECT name, dept, hire\_date FROM employee WHERE hire\_date > '2006-12-31';

Program 24: Indexing and join: Consider the relation

employee (emp\_id,e\_name,salary ,Date of Joining,Dapt\_no,Designation)

Customer(c\_id, c\_name , email , city , pincode)Order(order\_id , date , amount , cust\_id.

a. create empid as primary key and indices on table employee.

b. create user defined index on any column

c. create sequence using auo-increment.

d. truncate table.

e. find list of customers who placed order and details of their orders.

f. find info of customers and append order details to the table

g. list down customers who haven’t placed order.

// 1. Create the `employee` Collection with Sample Data

db.employee.insertMany([{ emp\_id: 1, e\_name: "Alice Johnson", salary: 50000, Date\_of\_Joining: new Date("2020-05-20"), Dapt\_no: 101, Designation: "Manager" },

{ emp\_id: 2, e\_name: "Bob Smith", salary: 60000, Date\_of\_Joining: new Date("2019-06-15"), Dapt\_no: 102, Designation: "Analyst" },

{ emp\_id: 3, e\_name: "Charlie Brown", salary: 55000, Date\_of\_Joining: new Date("2021-01-10"), Dapt\_no: 103, Designation: "Engineer" }]);

// 2. Create the `customer` Collection with Sample Data

db.customer.insertMany([

{ c\_id: 1, c\_name: "John Doe", email: "john@example.com", city: "New York", pincode: "10001" },

{ c\_id: 2, c\_name: "Jane Doe", email: "jane@example.com", city: "Los Angeles", pincode: "90001" },

{ c\_id: 3, c\_name: "Sam Smith", email: "sam@example.com", city: "Chicago", pincode: "60601" }

]);

// 3. Create the `order` Collection with Sample Data

db.order.insertMany([

{ order\_id: 1, date: new Date("2023-10-05"), amount: 150, cust\_id: 1 },

{ order\_id: 2, date: new Date("2023-10-10"), amount: 200, cust\_id: 2 },

{ order\_id: 3, date: new Date("2023-10-15"), amount: 250, cust\_id: 1 }

]);

// a. Create `emp\_id` as a Primary Key and Indices on `employee`

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

// b. Create a User-Defined Index on `salary` Column

db.employee.createIndex({ salary: 1 });

// c. Create Auto-Increment Sequence

// First, create a `counters` collection to store sequence numbers.

db.counters.insertOne({ \_id: "employee\_id", seq: 0 });

// Function to get the next sequence value

function getNextSequence(name) {

const sequenceDocument = db.counters.findOneAndUpdate(

{ \_id: name },

{ $inc: { seq: 1 } },

{ returnNewDocument: true }

);

return sequenceDocument.seq;

}

// Example usage: Insert a new employee with auto-incremented `emp\_id`

db.employee.insertOne({

emp\_id: getNextSequence("employee\_id"),

e\_name: "New Employee",

salary: 60000,

Date\_of\_Joining: new Date("2022-02-15"),

Dapt\_no: 104,

Designation: "Developer" });

// d. Truncate the `employee` Collection

db.employee.deleteMany({});

// e. Find List of Customers Who Placed Orders and Their Order Details

db.customer.aggregate([

{ $lookup: {

from: "order",

localField: "c\_id",

foreignField: "cust\_id",

as: "order\_details"

} },

{

$match: { "order\_details": { $ne: [] } } // Only customers with orders

} ]);

// f. Find Info of Customers and Append Order Details

db.customer.aggregate([

{

$lookup: {

from: "order",

localField: "c\_id",

foreignField: "cust\_id",

as: "order\_details"

}

}

]);

// g. List Customers Who Haven’t Placed Orders

db.customer.aggregate([

{

$lookup: {

from: "order",

localField: "c\_id",

foreignField: "cust\_id",

as: "order\_details"

} },

{

$match: { "order\_details": { $size: 0 } } // Customers without orders

} ]);

* **Program 25**

Implement aggregation and indexing with suitable example using MongoDB

//USE DATABASE

> use comp;

switched to db comp

//CREATE COLLECTION WEBSITE

> db.createCollection('website');

{ "ok" : 1 }

//INSERT VALUES IN WEBSITE > db.website.insert({'roll':'1','name':'harsh','amount':1000,'ur l':'www.yahoo.com'}); WriteResult({ "nInserted" : 1 }) >db.website.insert({'roll':'2','name':'jitesh','amount':2000,'url':'www.yah oo.com '}); WriteResult({ "nInserted" : 1 }) >db.website.insert({'roll':'3','name':'rina','amount':3000,'url':'www.googl e.com' }); WriteResult({ "nInserted" : 1 }) >db.website.insert({'roll':'4','name':'ash','amount':4000,'url':'www.gmail. com'}) ; WriteResult({ "nInserted" : 1 }) >db.website.insert({'roll':'5','name':'ash','amount':1000,'url':'www.pvg.co m'}); WriteResult({ "nInserted" : 1 })

//SUM AGGREGATE

> db.website.aggregate({$group:{\_id:"$name","total":{$sum:"$amount"}}});

{ "\_id" : "ash", "total" : 5000 } { "\_id" : "rina", "total" : 3000 } { "\_id" : "jitesh", "total" : 2000 } { "\_id" : "harsh", "total" : 2000 }

//AVG AGGREGATE

> db.website.aggregate({$group:{\_id:"$name","total": {$avg:"$amount"}}}); { "\_id" : "ash", "total" : 2500 } { "\_id" : "rina", "total" : 3000 } { "\_id" : "jitesh", "total" : 2000 } { "\_id" : "harsh", "total" : 1000 }

//MIN AGGREGATION

> db.website.aggregate({$group:{\_id:"$name","total":{$min:"$amount"}}}); { "\_id" : "ash", "total" : 1000 } { "\_id" : "rina", "total" : 3000 } { "\_id" : "jitesh", "total" : 2000 } { "\_id" : "harsh", "total" : 1000 }

//MAX AGGREGATION

> db.website.aggregate({$group:{\_id:"$name","total":{$max:"$amount"}}}); { "\_id" : "ash", "total" : 4000 } { "\_id" : "rina", "total" : 3000 } { "\_id" : "jitesh", "total" : 2000 } { "\_id" : "harsh", "total" : 1000 }

//FIRST AGGREGATION

> db.website.aggregate({$group:{\_id:"$name","total":{$first:"$amount"}}}); { "\_id" : "ash", "total" : 4000 } { "\_id" : "rina", "total" : 3000 } { "\_id" : "jitesh", "total" : 2000 } { "\_id" : "harsh", "total" : 1000 }

//LAST AGGREGATION

> db.website.aggregate({$group:{\_id:"$name","total":{$last:"$amount"}}}); { "\_id" : "ash", "total" : 1000 } { "\_id" : "rina", "total" : 3000 } { "\_id" : "jitesh", "total" : 2000 } { "\_id" : "harsh", "total" : 1000 }

//PUSH AGGREGATION

> db.website.aggregate({$group:{\_id:"$name","total": {$push:"$amount"}}}); { "\_id" : "ash", "total" : [ 4000, 1000 ] } { "\_id" : "rina", "total" : [ 3000 ] } { "\_id" : "jitesh", "total" : [ 2000 ] } { "\_id" : "harsh", "total" : [ 1000, 1000 ] }

//COUNT AGGREGATION

> db.website.aggregate({$group:{\_id:"$name","total": {$sum:1}}}); { "\_id" : "ash", "total" : 2 } { "\_id" : "rina", "total" : 1 } { "\_id" : "jitesh", "total" : 1 } { "\_id" : "harsh", "total" : 2 } //ADDTOSET AGGREGATE

> db.website.aggregate({$group: {\_id:"$name","total"{$addToSet:"$amount"}}}); { "\_id" : "ash", "total" : [ 1000, 4000 ] } { "\_id" : "rina", "total" : [ 3000 ] } { "\_id" : "jitesh", "total" : [ 2000 ] } { "\_id" : "harsh", "total" : [ 1000 ] }

//INDEXING

> db.createCollection('website1'); { "ok" : 1 } > db.website1.insert({'r':1,'name':'harsh'}); WriteResult({ "nInserted" : 1 })

> db.website1.find().pretty() { "\_id" : ObjectId("5ba3509a444926329738012d"), "roll" : 1, "name" : "harsh" } { "\_id" : ObjectId("5ba35293444926329738012e"), "roll" : 1, "name" : "harsh" }

> db.website1.createIndex({'name':1}) { "numIndexesBefore" : 2, "note" : "all indexes already exist", "ok" : 1 }

//CREATE INDEXING

> db.website1.createIndex({'name':-1}) { "createdCollectionAutomatically" : false, "numIndexesBefore" : 2, "numIndexesAfter" : 3, "ok" : 1 }

> db.website1.getIndexses() 2018-09-20T13:28:09.628+0530 TypeError: Property 'getIndexses' of object om.website is not a function

> db.website1.getIndexes() [ {"v" : 1, "key" : { "\_id" : 1 }, "name" : "\_id\_", "ns" : "harsh.website1" }, { "v" : 1, "key" : { "name" : 1 }, "name" : "name\_1", "ns" : "harsh.website1" }, { "v" : 1, "key" : { "name" : -1 }, "name" : "name\_-1", "ns" : "harsh.website1" } ]

> db.website1.createIndex({'name':-1}) { "numIndexesBefore" : 3, "note" : "all indexes already exist", "ok" : 1 }

//DROP INDEX > db.website.dropIndex({'name':-1}) { "nIndexesWas" : 3, "ok" : 1 }

> db.website1.dropIndex({'name':1}) { "nIndexesWas" : 2, "ok" : 1 }

> db.website1.dropIndex({'name':1}) { "nIndexesWas" : 1, "ok" : 0, "errmsg" : "can't find index with key:{ name: 1.0 }" }

//GET INDEXING

> db.website1.getIndexes() [ { "v" : 1, "key" : { "\_id" : 1 }, "name" : "\_id\_", "ns" : "harsh.website1" } ]

> db.website1.find().pretty()

{ "\_id" : ObjectId("5ba3509a444926329738012d"), "roll" : 1, "name" : "harsh" } { "\_id" : ObjectId("5ba35293444926329738012e"), "roll" : 1, "name" : "harsh" }

> db.website1.createIndex({'name':1}) { "createdCollectionAutomatically" : false, "numIndexesBefore" : 1, "numIndexesAfter" : 2, "ok" : 1 }

> db.website1.getIndexes() [ { "v" : 1, "key" : { "\_id" : 1 }, "name" : "\_id\_", "ns" : "harsh.website1" }, { "v" : 1, "key" : {"name" : 1 }, "name" : "name\_1", "ns" : "harsh.website1" } ]

> db.website1.dropIndex({'name':1}) { "nIndexesWas" : 2, "ok" : 1 }

> db.website1.getIndexes() [ { "v" : 1, "key" : { "\_id" : 1 }, "name" : "\_id\_", "ns" : "harsh.website1" } ]

> db.website1.createIndex({'name':1,'r':-1}) {"createdCollectionAutomatically" : false, "numIndexesBefore" : 1, "numIndexesAfter" : 2, "ok" : 1 } > db.website1.getIndexes() [ { "v" : 1, "key" : { "\_id" : 1 }, "name" : "\_id\_", "ns" : "harsh.website1" }, { "v" : 1, "key" : { "name" : 1, "r" : -1 }, "name" : "name\_1\_r\_-1", "ns" : "harsh.website1" } ] (i search)`db.website1.insert({'roll':1,'name':'harsh'});':