Q: Draw an ER for Bank database with atleast 5 entities and convert them into tables.
Perform DDL on above converted tables.
1.Create tables with all constraints
2. Create views on any two tables using join conditions
3. Create index called CustomerId. Entries should be in ascending order by customer name.
4. Create sequence on Acctno.

```
Create Table customer(
        custid int primary key,
    Name varchar(100),
    phoneNo int,
    Address varchar(225)
);



create table branch(
        branchId int primary key,
    branchName varchar(100),
    location varchar(100)
);

create table account(
        acctNo int primary key,
    customerId int,
    branchId int,
    amount decimal(10,2),
        foreign key (customerID) references customer(custid),
    foreign key (branchId) references branch(branchId)
);

-- addmore data in it
```

```
create view customeraccount as
select c.custid,c.Name,c.address,b.branchName,a.amount
from customer c
join account a on c.custid = a.customerId
join branch b on b.branchId = a.branchId;

-- Insert data into customer table
INSERT INTO customer (custid, Name, phoneNo, Address) VALUES
(101, 'John Smith', 9876543, '123 Oak, New, NY'),
(102, 'Sarah Johnson', 876543, '456 Maple, Chic, IL'),
(103, 'Michael Chen', 765432, '789 Pine, San, CA'),
(104, 'Priya Sharma', 654321, '234 Birc, Aus, TX'),
(105, 'David Rodriguez', 543210, '567 Cedar, Mi, FL'),
(106, 'Emma Wilson', 43210, '890 Elm, Sea, WA'),
(107, 'Ahmed Hassan', 32109, '345 Willow, Bost, MA'),
(108, 'Maria Garcia', 210987, '678 Spruce, Den, CO'),
(109, 'James Taylor', 10987, '901 Redwood, Port, OR'),
(110, 'Sophia Lee', 90876, '234 Aspen, Atla, GA');

-- Insert data into branch table
INSERT INTO branch (branchId, branchName, location) VALUES
(201, 'Downtown Branch', 'Central City'),
(202, 'Westside Branch', 'Western District'),
(203, 'Northgate Branch', 'Northern Heights'),
(204, 'Eastview Branch', 'Eastern Suburbs'),
(205, 'Southpoint Branch', 'Southern Plaza');

-- Insert data into account table
INSERT INTO account (acctNo, customerId, branchId, amount) VALUES
(30001, 101, 201, 5750.75),
(30002, 102, 202, 12450.50),
(30003, 103, 203, 8975.25),
(30004, 104, 201, 6200.00),
(30005, 105, 204, 15325.75),
```

(30006, 106, 202, 4890.50),
(30007, 107, 205, 9240.25),
(30008, 108, 203, 7670.00),
(30009, 109, 204, 11580.75),
(30010, 110, 205, 6345.50),
(30011, 101, 203, 2500.00), -- Second account for customer 101
(30012, 103, 201, 10800.25), -- Second account for customer 103
(30013, 105, 202, 3750.60); -- Second account for customer 105

select * from customeraccount;

create index customerId on customer(name asc);

——

Q2**2:** Draw an ER for Company database with atleast 4 entities and convert them into tables.
Perform DDL on Above converted tables.
1. Create tables with all constraints
2. create views on any two tables using conditions
3. create index called EmployeeId for the department table. Entries should be in ascending order by department id and then by employee id within each department.
4. create sequence on Employee id.

```
-- Department Table
CREATE TABLE Department (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(50) NOT NULL,
    Location VARCHAR(50)
```

```
);

-- Employee Table
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    DepartmentID INT,
    Salary DECIMAL(10, 2),
    HireDate DATE,
                    FOREIGN KEY (DepartmentID) REFERENCES
Department(DepartmentID)
);

-- Project Table
CREATE TABLE Project (
    ProjectID INT PRIMARY KEY,
    ProjectName VARCHAR(100),
    Budget DECIMAL(12,2),
    DepartmentID INT,
                    FOREIGN KEY (DepartmentID) REFERENCES
Department(DepartmentID)
);

-- WorksOn Table (many-to-many)
CREATE TABLE WorksOn (
    EmployeeID INT,
    ProjectID INT,
    HoursWorked DECIMAL(5,2),
    PRIMARY KEY (EmployeeID, ProjectID),
    FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID),
    FOREIGN KEY (ProjectID) REFERENCES Project(ProjectID)
);
```

```
-- View 1: View employees with salary above 50,000
CREATE VIEW HighPaidEmployees AS
SELECT EmployeeID, FirstName, LastName, Salary
FROM Employee
WHERE Salary > 50000;

-- View 2: Projects with budget over 1 million
CREATE VIEW BigBudgetProjects AS
SELECT ProjectID, ProjectName, Budget
FROM Project
WHERE Budget > 1000000;

-- Composite index: First by DepartmentID, then by EmployeeID (assume an EmployeeID exists in Department for indexing purpose)
CREATE INDEX EmployeeId
ON Department (DepartmentID ASC);

CREATE INDEX Dept_Emp_Index
ON Employee (DepartmentID ASC, EmployeeID ASC);

INSERT INTO Employee (EmployeeID, FirstName, LastName, DepartmentID, Salary, HireDate)
VALUES (Emp_ID_Seq.NEXTVAL, 'John', 'Doe', 1, 60000, '2024-05-20');
```

——

Q3 **w**rite a trigger for **Library (bid, bname, doi, status)** to update the number of copies (noc) according to ISSUE & RETURN status on update or insert query. Increase the noc if status is RETURN, Decrease noc if status is ISSUE in **Library_Audit table(bid,bname,noc,timestampofquery).**

Write a trigger after update on Library such that if doi is more than 20 days ago then status should be FINE and in the Library_Audit table fine should be equal to no. of days * 10.

```
-- 1. Create and use the database
CREATE DATABASE IF NOT EXISTS FINAL;
USE FINAL;

-- 2. Drop tables if they exist
DROP TABLE IF EXISTS lib;
DROP TABLE IF EXISTS libraryA;

-- 3. Create lib table
CREATE TABLE lib (
    bid INT PRIMARY KEY,
    bname VARCHAR(100),
    doi DATE,
    bstatus VARCHAR(50)
);

-- 4. Create libraryA table
CREATE TABLE libraryA (
    bid INT PRIMARY KEY,
    bname VARCHAR(100),
    doi DATE,
    noc INT DEFAULT 0,
    tsq TIMESTAMP
);

-- 5. Create trigger for insertion
DELIMITER $$

CREATE TRIGGER on_insert
AFTER INSERT ON lib
FOR EACH ROW
```

```
BEGIN
    IF NEW.bstatus = 'I' THEN
        INSERT INTO libraryA(bid, bname, doi, noc, tsq)
        VALUES (NEW.bid, NEW.bname, NEW.doi, -1, NOW());
    ELSEIF NEW.bstatus = 'R' THEN
        INSERT INTO libraryA(bid, bname, doi, noc, tsq)
        VALUES (NEW.bid, NEW.bname, NEW.doi, 1, NOW());
    END IF;
END $$

DELIMITER ;

-- 6. Insert sample data
INSERT INTO lib(bid, bname, doi, bstatus) VALUES
(101, 'DBMS', '1998-01-01', 'I'),
(102, 'OS', '1998-02-03', 'I'),
(103, 'DBMS', '1998-04-05', 'R'),
(104, 'DBMS', '1998-06-07', 'R'),
(105, 'CN', '2000-01-01', 'I');

-- 7. Check data in libraryA
SELECT * FROM libraryA;
```

——

Q4 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 DATABASE FINAL;
USE FINAL;
DROP DATABASE FINAL;
DROP TRIGGER ON_DELETE;
```

```
CREATE TABLE lib(
        bid int primary key,
    bname varchar(100),
    doi date,
    bstatus varchar(50)
);

CREATE TABLE libraryA(
        bid int primary key,
    bname varchar(100),
    tsq timestamp,
    action varchar(50)
);

insert into lib(bid,bname,doi,bstatus)
values(101,"DBMS",'1998-01-01',"I"),
        (102,"os",'1998-02-03',"I"),
        (103,"DBMS",'1998-04-05',"r"),
        (104,"DBMS",'1998-06-07',"r");

DELIMITER $$
create trigger on_update
after update on lib
for each row
begin
        insert into libraryA(bid,bname,tsq,action)
    values(old.bid,old.bname,NOW(),"UPDATE");
END$$
DELIMITER ;

DELIMITER $$
create trigger on_delete
after delete on lib
```

```
for each row
begin
        insert into libraryA(bid,bname,tsq,action)
     values(old.bid,old.bname,NOW(),"DELETE");
END$$
DELIMITER ;

select * from libraryA;
select * from lib;
SET SQL_SAFE_UPDATES = 0;
update lib set bid = 105 where bid = 104;

delete from lib where bname = 'os';
```

——

Q5Create a collection sites(url,dateofaccess). Write a MapReduce function to find the no. of times a site was accessed in a month.

```
// 1. Sample data insert into 'sites' collection
db.sites.insertMany([
  { url: "https://example.com", dateofaccess: new Date("2025-03-01") },
  { url: "https://example.com", dateofaccess: new Date("2025-03-15") },
  { url: "https://example.com", dateofaccess: new Date("2025-04-01") },
  { url: "https://example.org", dateofaccess: new Date("2025-03-10") },
  { url: "https://example.org", dateofaccess: new Date("2025-03-12") },
  { url: "https://example.org", dateofaccess: new Date("2025-04-15") },
  { url: "https://example.net", dateofaccess: new Date("2025-03-20") }
]);

// 2. Map Function
const mapFunction = function () {
    const month = this.dateofaccess.getMonth() + 1; // Months are 0-
```

```
indexed
  const year = this.dateofaccess.getFullYear();
  const key = `${this.url}_${year}-${month.toString().padStart(2, '0')}`;
  emit(key, 1);
};

// 3. Reduce Function
const reduceFunction = function (key, values) {
  return Array.sum(values);
};

// 4. Run MapReduce
db.sites.mapReduce(
  mapFunction,
  reduceFunction,
  {
    out: "site_access_per_month"
  }
);

// 5. View result
db.site_access_per_month.find().pretty();
```

——

Q6 Create tables CitiesIndia(**pincode**,nameofcity,earliername,area,population,avgrainfall) Categories(Type,pincode) ***Note:- Enter data only in CitiesIndia***

Write PL/SQL Procedure & function to find the population density of the cities. If the population density is above 3000 then Type of city must be entered as High Density in Category table. Between 2999 to 1000 as Moderate and below 999 as Low Density. Error must be displayed for population less than 10 or greater than 25718.

```
drop procedure main;

-- Create CitiesIndia table with INTEGER values
CREATE TABLE CitiesIndia (
    pincode VARCHAR(10) PRIMARY KEY,
    nameofcity VARCHAR(50) NOT NULL,
    earliername VARCHAR(50),
    area INTEGER NOT NULL CHECK (area > 0),
    population INTEGER NOT NULL,
    avgrainfall INTEGER
);

-- Create Categories table
CREATE TABLE Category (
    pincode VARCHAR(10),
    Type VARCHAR(20),
    PRIMARY KEY (pincode),
    FOREIGN KEY (pincode) REFERENCES CitiesIndia(pincode)
);

-- Insert sample data into CitiesIndia with INTEGER values
INSERT INTO CitiesIndia VALUES
('400001', 'Mumbai', 'Bombay', 603, 12442373, 242),
('110001', 'Delhi', 'Indraprastha', 1484, 11007835, 797),
('700001', 'Kolkata', 'Calcutta', 205, 4496694, 158),
('600001', 'Chennai', 'Madras', 426, 4646732, 141),
('560001', 'Bangalore', 'Bengaluru', 709, 8443675, 860),
('380001', 'Ahmedabad', 'Karnavati', 464, 5570585, 803),
('500001', 'Hyderabad', 'Bhagyanagar', 650, 6993262, 756),
```

```
('282001', 'Agra', '', 188, 1585704, 660),
('302001', 'Jaipur', 'Pink City', 468, 3073350, 557),
('800001', 'Patna', 'Pataliputra', 250, 1684222, 109);


DELIMITER $$
CREATE PROCEDURE main()
begin
        declare pin int;
    declare pop int;
    declare area int;
    declare class varchar(50);
    declare density decimal(10,2);
    declare done int default 0;

    declare cur cursor for
            select pincode,population,area from citiesIndia;
       declare continue handler for not found set done = 1;

    open cur;
            read_loop:LOOP
                    fetch cur into pin,pop,area;
            if done then
                        leave read_loop;
                    end if;
            set density = pop/area;
            set class = cat(density);
            insert into category(pincode,type)
            values(pin,class);
             end loop;
```

```
        close cur;
end$$
delimiter ;

delimiter $$

create function cat(density decimal(10,2))
returns varchar(100)
deterministic
begin
        declare class varchar(100);
     if density > 3000 then
                set class = "High density";
        elseif density>1000 and density<2999 then
                set class = "Moderate density";
        else
                set class = "Low density";
        end if;
return class;
end $$
DELIMITER ;

DELIMITER $$

select * from category;
 CALL main();
```

Q7Write PL/SQL Procedure & function to find class [Distinction (Total marks from 1499 to 990) ,First Class(899 to 900) Higher Second (899 to 825) ,Second,Pass (824 to 750)] of a student based on total marks from table **Student (rollno, name, Marks1, Marks2, Marks3, Marks4, Marks5).**

Use exception handling when negative marks are entered by user(Marks<0) or Marks more than 100 are entered by user.. Store the result into Result table recording RollNo,total marks, and class for each student .

```
-- Create Student table

CREATE TABLE Student (

rollno INT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

marks1 INT CHECK (marks1 BETWEEN 0 AND 100),

marks2 INT CHECK (marks2 BETWEEN 0 AND 100),

marks3 INT CHECK (marks3 BETWEEN 0 AND 100),

marks4 INT CHECK (marks4 BETWEEN 0 AND 100),

marks5 INT CHECK (marks5 BETWEEN 0 AND 100)

);


-- Create Result table

CREATE TABLE Result (

rollno INT PRIMARY KEY,

totalMarks INT,

class VARCHAR(100),

FOREIGN KEY (rollno) REFERENCES Student(rollno)

);
```

```
-- Insert sample student data
INSERT INTO Student VALUES
(101, 'Rahul Sharma', 85, 78, 92, 88, 76),
(102, 'Priya Patel', 72, 65, 68, 74, 70),
(103, 'Amit Singh', 45, 52, 38, 60, 42),
(104, 'Neha Gupta', 92, 95, 88, 90, 94),
(105, 'Vikram Joshi', 35, 28, 42, 30, 25),
(106, 'Ananya Reddy', 78, 82, 75, 80, 85),
(107, 'Karan Malhotra', 65, 70, 68, 72, 60),
(108, 'Divya Iyer', 55, 48, 52, 60, 58),
(109, 'Rohan Verma', 25, 30, 22, 28, 20),
(110, 'Sneha Desai', 95, 92, 98, 90, 96);

Delimiter $$
create function category(totalM int)
returns varchar(100)
deterministic
begin
declare class varchar(100);
if totalM > 400 then
set class = "First class";
elseif totalM between 300 and 400 then
set class = "Second class";
elseif totalM between 200 and 299 then
```

```
set class = "Third class";
else
set class = "Fail";
end if;
return class;
end$$

DELIMITER ;

DELIMITER $$

CREATE PROCEDURE student()
begin
declare m1 int;
declare m2 int;
declare m3 int;
declare m4 int;
declare m5 int;
declare totalM int;
declare roll int;
declare class varchar(100);
declare done int default 0;

declare cur cursor for
select rollno,marks1,marks2,marks3,marks4,marks5 from student ;
```

```
declare continue handler for not found set done = 1;


open cur;

read_loop:LOOP

fetch cur into roll,m1,m2,m3,m4,m5;

if done then

leave read_loop;

end if;


IF m1<0 or m1>100 or m2<0 or m2>100 or m3<0 or m3>100 or m4<0 or
m4>100 or m5<0 or m5>100 then

SIGNAL SQLSTATE '45000'

SET MESSAGE_TEXT = 'ERROR IN MARKS';

END IF;


set totalM = m1+m2+m3+m4+m5;

set class = category(totalM);


insert into result(rollno,totalMarks,class)

values(roll,totalM,class);

end loop;

close cur;

end $$


DELIMITER ;
```

call student();

select * from result;

——

Q8 Draw ER for Library database with atleast 5 entities and convert them into tables.

Perform DDL on above converted tables.

1. Create tables with all constraints (Based on ERD cardinalities)
2. Create views on any two tables using join condition
3. Create index called Lib_Index1. Entries should be in ascending order by Author name.
4. Create sequence on Bookid.

```
-- 1. Create Author table
CREATE TABLE Author (
    AuthorID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Country VARCHAR(50)
);

-- 2. Create Publisher table
CREATE TABLE Publisher (
    PublisherID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Address VARCHAR(150)
);

-- 3. Create Book table
CREATE TABLE Book (
    BookID INT PRIMARY KEY,
    Title VARCHAR(150) NOT NULL,
    ISBN VARCHAR(20) UNIQUE,
    PublisherID INT,
    AuthorID INT,
```

```
    FOREIGN KEY (PublisherID) REFERENCES Publisher(PublisherID),
    FOREIGN KEY (AuthorID) REFERENCES Author(AuthorID)
);

-- 4. Create Member table
CREATE TABLE Member (
    MemberID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE,
    Phone VARCHAR(15)
);

-- 5. Create Issue table
CREATE TABLE Issue (
    IssueID INT PRIMARY KEY,
    BookID INT,
    MemberID INT,
    IssueDate DATE,
    ReturnDate DATE,
    FOREIGN KEY (BookID) REFERENCES Book(BookID),
    FOREIGN KEY (MemberID) REFERENCES Member(MemberID)
);

-- View 1: Book and Author details
CREATE VIEW Book_Author_View AS
SELECT b.BookID, b.Title, a.Name AS AuthorName, a.Country
FROM Book b
JOIN Author a ON b.AuthorID = a.AuthorID;

-- View 2: Issue and Member details
CREATE VIEW Issue_Member_View AS
SELECT i.IssueID, m.Name AS MemberName, b.Title, i.IssueDate,
i.ReturnDate
FROM Issue i
JOIN Member m ON i.MemberID = m.MemberID
JOIN Book b ON i.BookID = b.BookID;
```

——

——

Q9PL/SQL code block: Use of Control structure and Exception handling is mandatory. Write a PL/SQL block of code for the following requirements:-
Schema:
1. Borrower(Rollin, Name, DateofIssue, NameofBook)
2. Fine(Roll_no,Date,Amt)
3. Library (bid, bname, doi, status,noc)
4. transaction (tid,bid, bname, status)

1. Accept roll_no & name of book from user.
2. Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day.
3. If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.
4. After submitting the book, status will change from I to R.
5. Update the noc in library according to the transaction made. Increase the noc if status is RETURN, Decrease noc if status is ISSUE.
6. If condition of fine is true, then details will be stored into fine table.

```
CREATE TABLE Borrower (
   Rollin INT,
   Name VARCHAR(100),
   DateofIssue DATE,
   NameofBook VARCHAR(100),
   Status CHAR(1)
);

CREATE TABLE Fine (
   Roll_no INT,
   Date DATE,
   Amt INT
);
```

```
CREATE TABLE Lib (
   bid INT PRIMARY KEY,
   bname VARCHAR(100),
   doi DATE,
   bstatus VARCHAR(10),
   noc INT
);

CREATE TABLE trans (
   tid INT AUTO_INCREMENT PRIMARY KEY,
   bid INT,
   bname VARCHAR(100),
   status VARCHAR(10)
);




DELIMITER $$
create procedure libra(IN roll int,IN nameB varchar(50))
begin
        declare issueDate DATE;
     declare countD int;
     declare fine int default 0;
     declare bookid int;

     select DateofIssue into issueDate from borrower where roll = rollin
and nameB = NameofBook;

     set countD = DATEDIFF(CURDATE(),issueDate);
     if countD>30 then
              set fine = (CountD-30)*50+15*5;
        elseif countD>15 then
              set fine = countD*5;
```

```
        end if;

    update borrower
        set status = 'R'
    where nameB = nameOfBook AND nameB=nameofbook;

    select bid into bookid from lib where nameB=nameofbook;

    update lib
    set noc = noc+1
    where bid = bookid;

    insert into transaction(bid,bname,status) values (bookid,nameB,'R');

    if fine > 0 then
            insert into fine(roll_no,date,amt) values(roll,CURDATE(),fine);
        end if;

END $$
DELIMITER ;

call libra(101,'DBMS');
```

——

Q10Implement SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym for following relational schema:

Borrower(Rollin, Name, DateofIssue, NameofBook, Status)

```
-- Drop table if it exists
DROP TABLE IF EXISTS Borrower;
```

```
-- 1. Create table with constraints
CREATE TABLE Borrower (
    Rollin INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    DateofIssue DATE NOT NULL,
    NameofBook VARCHAR(100) NOT NULL,
    Status VARCHAR(1) CHECK (Status IN ('I', 'R'))
);

-- 2. Insert sample data
INSERT INTO Borrower (Name, DateofIssue, NameofBook, Status)
VALUES
('Anil', '2024-04-01', 'DBMS', 'I'),
('Sunita', '2024-03-10', 'OS', 'R'),
('Rahul', '2024-04-15', 'CN', 'I'),
('Sneha', '2024-02-28', 'DSA', 'R');

-- 3. Create view for issued books
CREATE OR REPLACE VIEW IssuedBooks AS
SELECT Rollin, Name, NameofBook, DateofIssue
FROM Borrower
WHERE Status = 'I';

-- 4. Create view for returned books
CREATE OR REPLACE VIEW ReturnedBooks AS
SELECT Rollin, Name, NameofBook, DateofIssue
FROM Borrower
WHERE Status = 'R';

-- 5. Create index on NameofBook
CREATE INDEX idx_nameofbook ON Borrower(NameofBook ASC);

-- 6. Create composite index on Status and DateofIssue
```

```
CREATE INDEX idx_status_date ON Borrower(Status, DateofIssue);

-- 7. Alter table to add Email column
ALTER TABLE Borrower
ADD COLUMN Email VARCHAR(100);

-- 8. Update nulls to avoid NOT NULL error
UPDATE Borrower SET Email = 'unknown@example.com' WHERE Email IS
NULL;

-- 9. Modify Email column to NOT NULL
ALTER TABLE Borrower
MODIFY COLUMN Email VARCHAR(100) NOT NULL;
```

——

Q10)Design at least 10 SQL queries for suitable database application using SQL DML statements: all types of Join, Sub-Query and View.

```
-- Sample tables
CREATE TABLE Department (
    DeptID INT PRIMARY KEY,
    DeptName VARCHAR(50)
);

CREATE TABLE Employee (
    EmpID INT PRIMARY KEY,
    EmpName VARCHAR(50),
    DeptID INT,
    Salary DECIMAL(10,2),
    ManagerID INT,
    FOREIGN KEY (DeptID) REFERENCES Department(DeptID),
    FOREIGN KEY (ManagerID) REFERENCES Employee(EmpID)
);
```

```
-- Insert sample data
INSERT INTO Department VALUES
(1, 'HR'),
(2, 'Finance'),
(3, 'IT');

INSERT INTO Employee VALUES
(101, 'Alice', 1, 70000, NULL),
(102, 'Bob', 2, 80000, 101),
(103, 'Charlie', 3, 90000, 101),
(104, 'David', 3, 85000, 103),
(105, 'Eve', 1, 60000, 101);

-- 1. Inner Join: Get employees with their department names
SELECT e.EmpName, d.DeptName
FROM Employee e
INNER JOIN Department d ON e.DeptID = d.DeptID;

-- 2. Left Join: List all employees and their managers' names (if any)
SELECT e.EmpName, m.EmpName AS ManagerName
FROM Employee e
LEFT JOIN Employee m ON e.ManagerID = m.EmpID;

-- 3. Right Join: List all departments and any employees in them
SELECT d.DeptName, e.EmpName
FROM Department d
RIGHT JOIN Employee e ON d.DeptID = e.DeptID;

-- 4. Full Outer Join (simulate in MySQL using UNION)
SELECT e.EmpName, d.DeptName
FROM Employee e
LEFT JOIN Department d ON e.DeptID = d.DeptID
UNION
```

```
SELECT e.EmpName, d.DeptName
FROM Employee e
RIGHT JOIN Department d ON e.DeptID = d.DeptID;

-- 5. Subquery: Find employees earning more than average salary
SELECT EmpName, Salary
FROM Employee
WHERE Salary > (SELECT AVG(Salary) FROM Employee);

-- 6. Subquery with IN: Employees working in departments with name 'IT' or 'Finance'
SELECT EmpName
FROM Employee
WHERE DeptID IN (SELECT DeptID FROM Department WHERE DeptName IN ('IT', 'Finance'));

-- 7. View: Create a view for Employee and Department details
CREATE OR REPLACE VIEW EmpDeptView AS
SELECT e.EmpName, d.DeptName, e.Salary
FROM Employee e
JOIN Department d ON e.DeptID = d.DeptID;

-- Use the view
SELECT * FROM EmpDeptView;

-- 8. Aggregate with Group By: Average salary per department
SELECT d.DeptName, AVG(e.Salary) AS AvgSalary
FROM Employee e
JOIN Department d ON e.DeptID = d.DeptID
GROUP BY d.DeptName;

-- 9. Correlated Subquery: Employees with salary higher than their manager
SELECT e.EmpName, e.Salary
```

```
FROM Employee e
WHERE e.ManagerID IS NOT NULL
AND e.Salary > (
    SELECT Salary FROM Employee m WHERE m.EmpID = e.ManagerID
);

-- 10. Self-Join: List employees and their direct reports
SELECT m.EmpName AS Manager, e.EmpName AS Employee
FROM Employee e
JOIN Employee m ON e.ManagerID = m.EmpID;
```

——

Q12mplement Indexing and querying with MongoDB using following example.

Students(stud_id, stud_name,stud_addr,stud_marks)

```
// 1. Create Students collection and insert sample data
db.Students.insertMany([
  { stud_id: 101, stud_name: "Alice", stud_addr: "New York", stud_marks: 85 },
  { stud_id: 102, stud_name: "Bob", stud_addr: "Los Angeles", stud_marks: 90 },
  { stud_id: 103, stud_name: "Charlie", stud_addr: "Chicago", stud_marks: 78 },
  { stud_id: 104, stud_name: "David", stud_addr: "New York", stud_marks: 92 },
  { stud_id: 105, stud_name: "Eva", stud_addr: "Los Angeles", stud_marks: 88 }
]);

// 2. Create indexes

// Create an index on stud_id (unique index)
db.Students.createIndex({ stud_id: 1 }, { unique: true });
```

```
// Create an index on stud_name (for fast name-based queries)
db.Students.createIndex({ stud_name: 1 });

// Create a compound index on stud_addr and stud_marks (for queries filtering by address and marks)
db.Students.createIndex({ stud_addr: 1, stud_marks: -1 });

// 3. Sample queries

// Query 1: Find student by stud_id
const studentById = db.Students.findOne({ stud_id: 102 });
printjson(studentById);

// Query 2: Find students from "New York"
const studentsFromNY = db.Students.find({ stud_addr: "New York" }).toArray();
printjson(studentsFromNY);

// Query 3: Find students with marks greater than or equal to 85, sorted by marks descending
const topStudents = db.Students.find({ stud_marks: { $gte: 85 } }).sort({ stud_marks: -1 }).toArray();
printjson(topStudents);
```

——

Q13**13** Create the instance of the COMPANY which consists of the following tables:

EMPLOYEE(Fname, Minit, Lname, Ssn, Bdate, Address, Sex, Salary, Dno)
DEPARTEMENT(Dname, Dno, Mgr_ssn, Mgr_start_date)
DEPT_LOCATIONS(Dnumber, Dlocation)
PROJECT(Pname, Pnumber, Plocation, Dno)
WORKS_ON(Essn, Pno, Hours)
DEPENDENT(Essn, Dependent_name, Sex, Bdate, Relationship)

```
-- EMPLOYEE table
CREATE TABLE EMPLOYEE (
    Fname VARCHAR(15),
    Minit CHAR(1),
    Lname VARCHAR(15),
    Ssn CHAR(9) PRIMARY KEY,
    Bdate DATE,
    Address VARCHAR(50),
    Sex CHAR(1),
    Salary DECIMAL(10, 2),
    Dno INT
);

-- DEPARTMENT table
CREATE TABLE DEPARTMENT (
    Dname VARCHAR(15),
    Dno INT PRIMARY KEY,
    Mgr_ssn CHAR(9),
    Mgr_start_date DATE
);

-- DEPT_LOCATIONS table
CREATE TABLE DEPT_LOCATIONS (
    Dnumber INT,
    Dlocation VARCHAR(20),
    FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dno)
);

-- PROJECT table
CREATE TABLE PROJECT (
    Pname VARCHAR(15),
    Pnumber INT PRIMARY KEY,
    Plocation VARCHAR(20),
    Dno INT,
    FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dno)
);

-- WORKS_ON table
```

```
CREATE TABLE WORKS_ON (
    Essn CHAR(9),
    Pno INT,
    Hours DECIMAL(3,1),
    PRIMARY KEY (Essn, Pno),
    FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
    FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber)
);

-- DEPENDENT table
CREATE TABLE DEPENDENT (
    Essn CHAR(9),
    Dependent_name VARCHAR(15),
    Sex CHAR(1),
    Bdate DATE,
    Relationship VARCHAR(10),
    PRIMARY KEY (Essn, Dependent_name),
    FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn)
);
-- DEPARTMENT
INSERT INTO DEPARTMENT VALUES ('Research', 1, '123456789', '2020-01-01');
INSERT INTO DEPARTMENT VALUES ('Administration', 2, '987654321', '2019-05-10');

-- EMPLOYEE
INSERT INTO EMPLOYEE VALUES ('John', 'B', 'Smith', '123456789', '1990-01-15', '123 Main St', 'M', 50000, 1);
INSERT INTO EMPLOYEE VALUES ('Alice', 'M', 'Johnson', '987654321', '1985-07-20', '456 Oak Ave', 'F', 60000, 2);
INSERT INTO EMPLOYEE VALUES ('Robert', 'T', 'Brown', '654987321', '1992-11-11', '789 Pine Rd', 'M', 55000, 1);

-- DEPT_LOCATIONS
INSERT INTO DEPT_LOCATIONS VALUES (1, 'Houston');
INSERT INTO DEPT_LOCATIONS VALUES (2, 'New York');

-- PROJECT
INSERT INTO PROJECT VALUES ('ProductX', 101, 'Bellaire', 1);
```

```
INSERT INTO PROJECT VALUES ('ProductY', 102, 'Sugarland', 1);
INSERT INTO PROJECT VALUES ('ProductZ', 103, 'Houston', 2);

-- WORKS_ON
INSERT INTO WORKS_ON VALUES ('123456789', 101, 32.5);
INSERT INTO WORKS_ON VALUES ('654987321', 102, 20.0);
INSERT INTO WORKS_ON VALUES ('987654321', 103, 40.0);

-- DEPENDENT
INSERT INTO DEPENDENT VALUES ('123456789', 'Anna', 'F', '2010-03-14',
'Daughter');
INSERT INTO DEPENDENT VALUES ('987654321', 'Tom', 'M', '2008-12-02',
'Son');



-- 1
select p.Pnumber,d.Dno,e.lname,e.address,e.Bdate
from project p
join department d on p.Dno = d.Dno
join employee e on d.mgr_ssn = e.ssn
where p.Plocation = 'Sugarland';

-- 2
select p.Pnumber
from project p
join works_on w on p.Pnumber = w.pno
join employee e on w.essn = e.ssn
where e.Lname = "Smith"
union
select p.Pnumber
from project p
join department d on p.Dno = d.Dno
join employee e on d.mgr_ssn = e.ssn
where e.lname = "Smith";

-- 3
select fname,lname,address
from employee
where address like '123 Main St';
```

```
-- 4
select e.salary as old_salary,
        e.salary*1.10 as new_salary
from employee e
join works_on w on w.essn = e.ssn
join project p on p.Pnumber=w.pno
where Pname = "Project X";
```

———

Q14 14 Implement all SQL DML opeartions with operators, functions, and set operator for given schema:

Account(Acc_no, branch_name,balance)
branch(branch_name,branch_city,assets)
customer(cust_name,cust_street,cust_city)
Depositor(cust_name,acc_no)
Loan(loan_no,branch_name,amount)
Borrower(cust_name,loan_no)

Solve following query:

1. Find the average account balance at each branch
2. Find no. of depositors at each branch.
3. Find the branches where average account balance > 12000.
4. Find number of tuples in customer relation.

```
drop database final2;
create database final2;
use final2;

CREATE TABLE Branch (
    branch_name VARCHAR(50) PRIMARY KEY,
    branch_city VARCHAR(50),
    assets DECIMAL(15, 2)
);

CREATE TABLE Customer (
    cust_name VARCHAR(50) PRIMARY KEY,
```

```
    cust_street VARCHAR(100),
    cust_city VARCHAR(50)
);

CREATE TABLE Account (
    acc_no INT PRIMARY KEY,
    branch_name VARCHAR(50),
    balance DECIMAL(10, 2),
    FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

CREATE TABLE Loan (
    loan_no INT PRIMARY KEY,
    branch_name VARCHAR(50),
    amount DECIMAL(10, 2),
    FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

CREATE TABLE Depositor (
    cust_name VARCHAR(50),
    acc_no INT,
    FOREIGN KEY (cust_name) REFERENCES Customer(cust_name),
    FOREIGN KEY (acc_no) REFERENCES Account(acc_no)
);

CREATE TABLE Borrower (
    cust_name VARCHAR(50),
    loan_no INT,
    FOREIGN KEY (cust_name) REFERENCES Customer(cust_name),
    FOREIGN KEY (loan_no) REFERENCES Loan(loan_no)
);

INSERT INTO Branch VALUES ('Central', 'Mumbai', 10000000);
INSERT INTO Branch VALUES ('South', 'Pune', 5000000);

INSERT INTO Account VALUES (101, 'Central', 15000);
INSERT INTO Account VALUES (102, 'Central', 12000);
INSERT INTO Account VALUES (103, 'South', 10000);
```

```
INSERT INTO Customer VALUES ('Alice', 'MG Road', 'Pune');
INSERT INTO Customer VALUES ('Bob', 'Linking Rd', 'Mumbai');

INSERT INTO Depositor VALUES ('Alice', 101);
INSERT INTO Depositor VALUES ('Bob', 102);

INSERT INTO Loan VALUES (201, 'Central', 25000);
INSERT INTO Borrower VALUES ('Alice', 201);

-- 3
SELECT branch_name
FROM Account
GROUP BY branch_name
HAVING AVG(balance) > 12000;

-- 2
SELECT A.branch_name, COUNT(DISTINCT D.cust_name) AS
        num_depositors
FROM Depositor D
JOIN Account A ON D.acc_no = A.acc_no
GROUP BY A.branch_name;

-- 1
SELECT branch_name, AVG(balance) AS avg_balance
FROM Account
GROUP BY branch_name;
```

Q15 easy

——

Q16Implement Map reduce operation with following example using MongoDB

Students(stud_id,
stud_name,stud_addr,stud_marks)

```
// 1. Sample data insert (if not already present)
```

```
db.Students.insertMany([
  { stud_id: 101, stud_name: "Alice", stud_addr: "New York", stud_marks:
      85 },
  { stud_id: 102, stud_name: "Bob", stud_addr: "Los Angeles", stud_marks:
      90 },
  { stud_id: 103, stud_name: "Charlie", stud_addr: "Chicago", stud_marks:
      78 },
  { stud_id: 104, stud_name: "David", stud_addr: "New York", stud_marks:
      92 },
  { stud_id: 105, stud_name: "Eva", stud_addr: "Los Angeles", stud_marks:
      88 }
]);

// 2. Map function: emits (key=stud_addr, value={count:1,
      totalMarks:stud_marks})
const mapFunction = function () {
  emit(this.stud_addr, { count: 1, totalMarks: this.stud_marks });
};

// 3. Reduce function: sums up count and totalMarks for each key
const reduceFunction = function (key, values) {
  let reducedVal = { count: 0, totalMarks: 0 };
  values.forEach(function (value) {
    reducedVal.count += value.count;
    reducedVal.totalMarks += value.totalMarks;
  });
  return reducedVal;
};

// 4. Finalize function: computes average marks
const finalizeFunction = function (key, reducedVal) {
  if (reducedVal.count > 0) {
    reducedVal.avgMarks = reducedVal.totalMarks / reducedVal.count;
  } else {
    reducedVal.avgMarks = 0;
  }
  return reducedVal;
};
```

```
// 5. Run MapReduce
db.Students.mapReduce(
  mapFunction,
  reduceFunction,
  {
    out: "avg_marks_per_address",
    finalize: finalizeFunction
  }
);

// 6. Show results
db.avg_marks_per_address.find().forEach(printjson);
```

——

Q17 Create following collection and using MongoDB implement all CRUD operations.

Orders(cust_id, amount, status)

```
// 1. Insert (Create) - Add multiple orders
db.Orders.insertMany([
  { cust_id: 1, amount: 250, status: "pending" },
  { cust_id: 2, amount: 450, status: "completed" },
  { cust_id: 3, amount: 300, status: "pending" },
  { cust_id: 1, amount: 150, status: "shipped" }
]);

// 2. Read - Find all orders
print("All Orders:");
db.Orders.find().forEach(printjson);

// Find orders by a specific customer
print("Orders for cust_id = 1:");
db.Orders.find({ cust_id: 1 }).forEach(printjson);

// 3. Update - Update order status by cust_id and amount
```

```
print("Updating order status for cust_id=1 and amount=250...");
db.Orders.updateOne(
   { cust_id: 1, amount: 250 },
   { $set: { status: "completed" } }
);

// Verify update
print("Updated Order:");
printjson(db.Orders.findOne({ cust_id: 1, amount: 250 }));

// 4. Delete - Delete orders with status 'pending'
print("Deleting orders with status 'pending'...");
db.Orders.deleteMany({ status: "pending" });

// Verify deletion
print("Remaining Orders after deletion:");
db.Orders.find().forEach(printjson);
```

——

Q18 Implement all SQL DML opeartions with operators, functions, and set operator for given schema:

Account(Acc_no, branch_name,balance)
branch(branch_name,branch_city,assets)
customer(cust_name,cust_street,cust_city)
Depositor(cust_name,acc_no)
Loan(loan_no,branch_name,amount)

Create above tables with appropriate constraints like primary key, foreign key, check constrains, not null etc.Solve following query:

1. Find all customers who have an account or loan or both at bank.
2. Find all customers who have both account and loan at bank.
3. Find all customer who have account but no loan at the bank.
4. Find average account balance at Akurdi branch.

```
-- Create tables with constraints

CREATE TABLE Account (
    Acc_no INT PRIMARY KEY,
    branch_name VARCHAR(50) NOT NULL,
    balance DECIMAL(15,2) CHECK (balance >= 0),
    CONSTRAINT fk_account_branch FOREIGN KEY (branch_name)
      REFERENCES branch(branch_name)
);

CREATE TABLE branch (
    branch_name VARCHAR(50) PRIMARY KEY,
    branch_city VARCHAR(50) NOT NULL,
    assets DECIMAL(20,2) CHECK (assets >= 0)
);

CREATE TABLE customer (
    cust_name VARCHAR(50) PRIMARY KEY,
    cust_street VARCHAR(100),
    cust_city VARCHAR(50)
);

CREATE TABLE Depositor (
    cust_name VARCHAR(50),
    acc_no INT,
    PRIMARY KEY (cust_name, acc_no),
    CONSTRAINT fk_depositor_customer FOREIGN KEY (cust_name)
      REFERENCES customer(cust_name),
    CONSTRAINT fk_depositor_account FOREIGN KEY (acc_no)
      REFERENCES Account(Acc_no)
);

CREATE TABLE Loan (
    loan_no INT PRIMARY KEY,
    branch_name VARCHAR(50) NOT NULL,
    amount DECIMAL(15,2) CHECK (amount >= 0),
    CONSTRAINT fk_loan_branch FOREIGN KEY (branch_name)
      REFERENCES branch(branch_name)
```

```
);

-- Queries --

-- 1. Find all customers who have an account or loan or both at the bank.

-- Customers with accounts (via Depositor table)
SELECT DISTINCT c.cust_name
FROM customer c
JOIN Depositor d ON c.cust_name = d.cust_name

UNION

-- Customers who have loans (assuming customers associated with loans
        via branch?
-- Usually Loan table does not mention customer directly, so assuming
        you want customers with loans at branch where they have account
        or depositor)
-- Since Loan table doesn't directly link to customer, typically, loans are
        linked to customers via another table (e.g. borrower).
-- If not given, we can only get customers with accounts or loans via
        branch_name matching.

-- If no link customer-loan, then skip or if you want customers who have
        loan and account at same branch:

SELECT DISTINCT c.cust_name
FROM customer c
JOIN Depositor d ON c.cust_name = d.cust_name
JOIN Account a ON d.acc_no = a.Acc_no
JOIN Loan l ON a.branch_name = l.branch_name;

-- 2. Find all customers who have both account and loan at the bank.

SELECT DISTINCT c.cust_name
FROM customer c
JOIN Depositor d ON c.cust_name = d.cust_name
JOIN Account a ON d.acc_no = a.Acc_no
JOIN Loan l ON a.branch_name = l.branch_name
```

INTERSECT

```
SELECT DISTINCT c.cust_name
FROM customer c
JOIN Depositor d ON c.cust_name = d.cust_name;

-- 3. Find all customers who have account but no loan at the bank.

SELECT DISTINCT c.cust_name
FROM customer c
JOIN Depositor d ON c.cust_name = d.cust_name
WHERE d.acc_no NOT IN (
    SELECT a.Acc_no
    FROM Account a
    JOIN Loan l ON a.branch_name = l.branch_name
);

-- 4. Find average account balance at Akurdi branch.

SELECT AVG(balance) AS avg_balance_akurdi
FROM Account
WHERE branch_name = 'Akurdi';
```

——

Q19Implement all SQL DML operations with operators, functions, and set operator for given schema:

Account(Acc_no, branch_name,balance)
branch(branch_name,branch_city,assets)
customer(cust_name,cust_street,cust_city)
Depositor(cust_name,acc_no)
Loan(loan_no,branch_name,amount)
Borrower(cust_name,loan_no)

Solve following query:

1. Calculate total loan amount given by bank.
2. Delete all loans with loan amount between 1300 and 1500.
3. Delete all tuples at every branch located in Nigdi.

```
drop database final2;
create database final2;
use final2;

-- Branch Table
CREATE TABLE Branch (
    branch_name VARCHAR(50) PRIMARY KEY,
    branch_city VARCHAR(50),
    assets DECIMAL(15, 2) CHECK (assets >= 0)
);

-- Account Table
CREATE TABLE Account (
    acc_no INT PRIMARY KEY,
    branch_name VARCHAR(50),
    balance DECIMAL(10, 2) CHECK (balance >= 0),
    FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

-- Customer Table
CREATE TABLE Customer (
    cust_name VARCHAR(50) PRIMARY KEY,
    cust_street VARCHAR(100),
    cust_city VARCHAR(50)
);

-- Depositor Table
CREATE TABLE Depositor (
    cust_name VARCHAR(50),
    acc_no INT,
    FOREIGN KEY (cust_name) REFERENCES Customer(cust_name),
    FOREIGN KEY (acc_no) REFERENCES Account(acc_no),
    PRIMARY KEY (cust_name, acc_no)
```

```
);

-- Loan Table
CREATE TABLE Loan (
    loan_no INT PRIMARY KEY,
    branch_name VARCHAR(50),
    amount DECIMAL(10, 2) CHECK (amount > 0),
    FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

-- Borrower Table
CREATE TABLE Borrower (
    cust_name VARCHAR(50),
    loan_no INT,
    FOREIGN KEY (cust_name) REFERENCES Customer(cust_name),
    FOREIGN KEY (loan_no) REFERENCES Loan(loan_no),
    PRIMARY KEY (cust_name, loan_no)
);

INSERT INTO Branch VALUES
('Akurdi', 'Pune', 1000000),
('Nigdi', 'Pune', 800000),
('Shivajinagar', 'Pune', 1200000);

INSERT INTO Customer VALUES
('Alice', 'Street1', 'Pune'),
('Bob', 'Street2', 'Pune'),
('Charlie', 'Street3', 'Pune');

INSERT INTO Account VALUES
(101, 'Akurdi', 15000),
(102, 'Nigdi', 12000),
(103, 'Nigdi', 13000);

INSERT INTO Depositor VALUES
('Alice', 101),
('Bob', 102);

INSERT INTO Loan VALUES
```

```
(201, 'Akurdi', 2000),
(202, 'Nigdi', 1400),
(203, 'Nigdi', 1600);

INSERT INTO Borrower VALUES
('Charlie', 201),
('Bob', 202),
('Alice', 203);


-- 1
select sum(amount)
from loan;
SET SQL_SAFE_UPDATES = 0;
-- 2
delete from borrower
where loan_no in(select loan_no from loan where amount>1300 and
        amount<1500);

delete from loan
where amount between 1300 and 1500;

-- 3
DELETE FROM Borrower
WHERE loan_no IN (
     SELECT loan_no FROM Loan WHERE branch_name = "nigdi");

delete from loan
where branch_name = "nigdi";

delete from depositor
where acc_no in(
 select acc_no from account where branch_name = "nigdi"
);

delete from account
where branch_name in (
select branch_name from branch where branch_name="nigdi");
```

```
delete from branch
where branch_name = "nigdi";
```
——

Q20Create the following tables.

1. Deposit (actno,cname,bname,amount,adate)
2. Branch (bname,city)
3. Customers (cname, city)
4. Borrow(loanno,cname,bname, amount)

Add primary key and foreign key wherever applicable.Insert data into the above created tables.

1. Display account date of customers “ABC”.
2. Modify the size of attribute of amount in deposit
3. Display names of customers living in city pune.
4. Display name of the city where branch “OBC” is located.
5. Find the number of tuples in the *customer* relation

```
-- 1. Create Tables with Primary and Foreign Keys

CREATE TABLE Branch (
    bname VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50) NOT NULL
);

CREATE TABLE Customers (
    cname VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50) NOT NULL
);

CREATE TABLE Deposit (
    actno INT PRIMARY KEY,
    cname VARCHAR(50) NOT NULL,
    bname VARCHAR(50) NOT NULL,
    amount DECIMAL(10,2) NOT NULL,
    adate DATE NOT NULL,
    CONSTRAINT fk_deposit_customer FOREIGN KEY (cname) REFERENCES
```

```
        Customers(cname),
    CONSTRAINT fk_deposit_branch FOREIGN KEY (bname) REFERENCES
        Branch(bname)
);

CREATE TABLE Borrow (
    loanno INT PRIMARY KEY,
    cname VARCHAR(50) NOT NULL,
    bname VARCHAR(50) NOT NULL,
    amount DECIMAL(10,2) NOT NULL,
    CONSTRAINT fk_borrow_customer FOREIGN KEY (cname) REFERENCES
        Customers(cname),
    CONSTRAINT fk_borrow_branch FOREIGN KEY (bname) REFERENCES
        Branch(bname)
);

-- 2. Insert sample data into tables

INSERT INTO Branch VALUES
('OBC', 'Mumbai'),
('SBI', 'Pune'),
('HDFC', 'Pune');

INSERT INTO Customers VALUES
('ABC', 'Mumbai'),
('DEF', 'Pune'),
('GHI', 'Pune'),
('JKL', 'Delhi');

INSERT INTO Deposit VALUES
(101, 'ABC', 'OBC', 1500.50, '2024-01-15'),
(102, 'DEF', 'SBI', 2500.00, '2024-02-10'),
(103, 'GHI', 'HDFC', 3000.75, '2024-03-20');

INSERT INTO Borrow VALUES
(201, 'ABC', 'OBC', 50000.00),
(202, 'JKL', 'SBI', 60000.00);

-- 3. Display account date of customers “ABC”
```

```
SELECT adate FROM Deposit WHERE cname = 'ABC';

-- 4. Modify the size of attribute amount in Deposit table

-- Assuming you want to increase precision or size of amount column in
        Deposit

ALTER TABLE Deposit
MODIFY amount DECIMAL(15,2);

-- 5. Display names of customers living in city Pune

SELECT cname FROM Customers WHERE city = 'Pune';

-- 6. Display name of the city where branch “OBC” is located

SELECT city FROM Branch WHERE bname = 'OBC';

-- 7. Find the number of tuples in the customer relation

SELECT COUNT(*) AS num_customers FROM Customers;
```

——

Q21Create following tables:

1. Deposit (actno,cname,bname,amount,adate)
2. Branch (bname,city)
3. Customers (cname, city)
4. Borrow(loanno,cname,bname, amount)

Add primary key and foreign key wherever applicable. Insert data into the above created tables.

1. Display customer name having living city Bombay and branch city Nagpur
2. Display customer name having same living city as their branch city
3. Display customer name who are borrowers as well as depositors and having living city Nagpur.

```
CREATE TABLE Branch (
    bname VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50) NOT NULL
);

CREATE TABLE Customers (
    cname VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50) NOT NULL
);

CREATE TABLE Deposit (
    actno INT PRIMARY KEY,
    cname VARCHAR(50),
    bname VARCHAR(50),
    amount DECIMAL(10,2),
    adate DATE,
    FOREIGN KEY (cname) REFERENCES Customers(cname),
    FOREIGN KEY (bname) REFERENCES Branch(bname)
);

CREATE TABLE Borrow (
    loanno INT PRIMARY KEY,
    cname VARCHAR(50),
    bname VARCHAR(50),
    amount DECIMAL(10,2),
    FOREIGN KEY (cname) REFERENCES Customers(cname),
    FOREIGN KEY (bname) REFERENCES Branch(bname)
);

-- Branches
INSERT INTO Branch VALUES ('Akurdi', 'Pune'),
                          ('MG Road', 'Bombay'),
                          ('Sitabuldi', 'Nagpur');

-- Customers
INSERT INTO Customers VALUES ('Alice', 'Bombay'),
                             ('Bob', 'Nagpur'),
```

```
                                    ('Charlie', 'Nagpur'),
                                    ('David', 'Pune'),
                                    ('Eva', 'Nagpur');

-- Deposit
INSERT INTO Deposit VALUES (101, 'Alice', 'Sitabuldi', 10000, '2024-01-01'),
                           (102, 'Bob', 'MG Road', 12000, '2024-01-02'),
                           (103, 'Charlie', 'Sitabuldi', 8000, '2024-01-03');

-- Borrow
INSERT INTO Borrow VALUES (201, 'Bob', 'Sitabuldi', 15000),
                          (202, 'Charlie', 'Sitabuldi', 20000),
                          (203, 'Eva', 'MG Road', 10000);

— 1
SELECT DISTINCT c.cname
FROM Customers c
JOIN Deposit d ON c.cname = d.cname
JOIN Branch b ON d.bname = b.bname
WHERE c.city = 'Bombay' AND b.city = 'Nagpur';

— 2
SELECT DISTINCT c.cname
FROM Customers c
JOIN Deposit d ON c.cname = d.cname
JOIN Branch b ON d.bname = b.bname
WHERE c.city = b.city;

—3
SELECT cname
FROM Customers
WHERE city = 'Nagpur'
  AND cname IN (SELECT cname FROM Deposit)
  AND cname IN (SELECT cname FROM Borrow);

————————————————————————————————————————
————————————————————————————————————————
————————————————————————————
```

Q22Create the following tables.

1. Deposit (actno,cname,bname,amount,adate)
2. Branch (bname,city)
3. Customers (cname, city)
4. Borrow(loanno,cname,bname, amount)

Add primary key and foreign key wherever applicable.
Insert data into the above created tables.

1. Display loan no and loan amount of borrowers having the same branch as that of sunil.
2. Display deposit and loan details of customers in the city where pramod is living.
3. Display borrower names having deposit amount greater than 1000 and having the same living city as pramod.
4. Display branch and living city of 'ABC'

```
CREATE TABLE Branch (
    bname VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50) NOT NULL
);

CREATE TABLE Customers (
    cname VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50) NOT NULL
);

CREATE TABLE Deposit (
    actno INT PRIMARY KEY,
    cname VARCHAR(50),
    bname VARCHAR(50),
    amount DECIMAL(10, 2),
    adate DATE,
    FOREIGN KEY (cname) REFERENCES Customers(cname),
    FOREIGN KEY (bname) REFERENCES Branch(bname)
);

CREATE TABLE Borrow (
    loanno INT PRIMARY KEY,
    cname VARCHAR(50),
```

```
    bname VARCHAR(50),
    amount DECIMAL(10, 2),
    FOREIGN KEY (cname) REFERENCES Customers(cname),
    FOREIGN KEY (bname) REFERENCES Branch(bname)
);

-- Branches
INSERT INTO Branch VALUES
('Akurdi', 'Pune'),
('MG Road', 'Mumbai'),
('Sitabuldi', 'Nagpur');

-- Customers
INSERT INTO Customers VALUES
('Sunil', 'Pune'),
('Pramod', 'Nagpur'),
('Ravi', 'Nagpur'),
('Anita', 'Mumbai'),
('ABC', 'Pune');

-- Deposit
INSERT INTO Deposit VALUES
(101, 'Sunil', 'Akurdi', 1500, '2024-01-05'),
(102, 'Pramod', 'Sitabuldi', 5000, '2024-01-10'),
(103, 'Ravi', 'Sitabuldi', 1200, '2024-02-15'),
(104, 'Anita', 'MG Road', 3000, '2024-03-12'),
(105, 'ABC', 'Akurdi', 1800, '2024-01-20');

-- Borrow
INSERT INTO Borrow VALUES
(201, 'Sunil', 'Akurdi', 7000),
(202, 'Pramod', 'Sitabuldi', 6000),
(203, 'Ravi', 'Sitabuldi', 8000),
(204, 'Anita', 'MG Road', 5000);

-- 1
SELECT loanno, amount
FROM Borrow
WHERE bname = (
```

```
    SELECT bname
    FROM Deposit
    WHERE cname = 'Sunil'
);

-- 2
-- Deposit details
SELECT d.*
FROM Deposit d
JOIN Customers c ON d.cname = c.cname
WHERE c.city = (
    SELECT city FROM Customers WHERE cname = 'Pramod'
);

-- Loan details
SELECT b.*
FROM Borrow b
JOIN Customers c ON b.cname = c.cname
WHERE c.city = (
    SELECT city FROM Customers WHERE cname = 'Pramod'
);


-- 3
SELECT DISTINCT b.cname
FROM Borrow b
JOIN Deposit d ON b.cname = d.cname
JOIN Customers c ON b.cname = c.cname
WHERE d.amount > 1000
  AND c.city = (
      SELECT city FROM Customers WHERE cname = 'Pramod'
  );

-- 4
SELECT d.bname, c.city
FROM Deposit d
JOIN Customers c ON d.cname = c.cname
WHERE d.cname = 'ABC';
```

——

Q23 Implement all Aggregation operations and types of indexing with following collection using MongoDB

Employee(emp_id, emp_name,emp_dept,salary)

```
// Use your database
use CompanyDB

// Drop the collection if it already exists
db.Employee.drop()

// Insert sample documents
db.Employee.insertMany([
  { emp_id: 1, emp_name: "Alice", emp_dept: "HR", salary: 5000 },
  { emp_id: 2, emp_name: "Bob", emp_dept: "IT", salary: 7000 },
  { emp_id: 3, emp_name: "Charlie", emp_dept: "IT", salary: 8000 },
  { emp_id: 4, emp_name: "David", emp_dept: "Finance", salary: 6000 },
  { emp_id: 5, emp_name: "Eve", emp_dept: "HR", salary: 5500 },
  { emp_id: 6, emp_name: "Frank", emp_dept: "Finance", salary: 7500 }
])

// -------------------
// AGGREGATION OPERATIONS
// -------------------

// 1. $match - filter documents
print("Match - Employees from IT department")
db.Employee.aggregate([
  { $match: { emp_dept: "IT" } }
])
```

```
// 2. $group - group by department and calculate total salary
print("Group - Total salary by department")
db.Employee.aggregate([
  {
    $group: {
      _id: "$emp_dept",
      total_salary: { $sum: "$salary" },
      avg_salary: { $avg: "$salary" },
      count: { $sum: 1 }
    }
  }
])

// 3. $project - include/exclude or compute new fields
print("Project - Show name and computed yearly salary")
db.Employee.aggregate([
  {
    $project: {
      emp_name: 1,
      emp_dept: 1,
      monthly_salary: "$salary",
      yearly_salary: { $multiply: ["$salary", 12] },
      _id: 0
    }
  }
])

// 4. $sort - sort by salary descending
print("Sort - Employees sorted by salary")
db.Employee.aggregate([
  { $sort: { salary: -1 } }
])

// 5. $limit - get top 3 earners
print("Limit - Top 3 highest paid employees")
db.Employee.aggregate([
  { $sort: { salary: -1 } },
  { $limit: 3 }
])
```

```
// 6. $skip - skip first 2 and show next
print("Skip - Skip first 2 and show next")
db.Employee.aggregate([
  { $sort: { salary: -1 } },
  { $skip: 2 },
  { $limit: 3 }
])

// 7. $count - count total employees
print("Count - Total number of employees")
db.Employee.aggregate([
  { $count: "total_employees" }
])

// 8. $addFields - Add a new field
print("AddFields - Add bonus field (10% of salary)")
db.Employee.aggregate([
  {
    $addFields: {
      bonus: { $multiply: ["$salary", 0.10] }
    }
  }
])

// 9. $unwind - example with array (for demo purpose, add array field)
print("Unwind - Example with array field")
db.Employee.updateOne(
  { emp_id: 1 },
  { $set: { skills: ["Communication", "Management"] } }
)

db.Employee.aggregate([
  { $match: { emp_id: 1 } },
  { $unwind: "$skills" }
])

// ------------------
// INDEXING TYPES
```

```
// -------------------

// 1. Single Field Index
db.Employee.createIndex({ emp_name: 1 })

// 2. Compound Index
db.Employee.createIndex({ emp_dept: 1, salary: -1 })

// 3. Unique Index
db.Employee.createIndex({ emp_id: 1 }, { unique: true })

// 4. Text Index (for full text search)
db.Employee.createIndex({ emp_name: "text" })

// 5. Hashed Index (for sharding or hash-based lookup)
db.Employee.createIndex({ emp_id: "hashed" })

// 6. Wildcard Index (for dynamic schemas)
db.Employee.createIndex({ "$**": 1 })

print("All indexes created successfully.")
```

——

Q2424 Create the following tables.

1. Deposit (actno,cname,bname,amount,adate)
2. Branch (bname,city)
3. Customers (cname, city)
4. Borrow(loanno,cname,bname, amount)

Add primary key and foreign key wherever applicable. Insert data into the above created tables.

1. Display amount for depositors living in the city where Anil is living.
2. Display total loan and maximum loan taken from KAROLBAGH branch.
3. Display total deposit of customers having account date later than ‘1-jan-98’.
4. Display maximum deposit of customers living in PUNE.

```
drop database final2;
create database final2;
use final2;

CREATE TABLE Branch (
    bname VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50) NOT NULL
);

CREATE TABLE Customers (
    cname VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50) NOT NULL
);

CREATE TABLE Deposit (
    actno INT PRIMARY KEY,
    cname VARCHAR(50),
    bname VARCHAR(50),
    amount DECIMAL(10,2),
    adate DATE,
    FOREIGN KEY (cname) REFERENCES Customers(cname),
    FOREIGN KEY (bname) REFERENCES Branch(bname)
);

CREATE TABLE Borrow (
    loanno INT PRIMARY KEY,
    cname VARCHAR(50),
    bname VARCHAR(50),
    amount DECIMAL(10,2),
    FOREIGN KEY (cname) REFERENCES Customers(cname),
    FOREIGN KEY (bname) REFERENCES Branch(bname)
);

-- Branch
INSERT INTO Branch VALUES
('KAROLBAGH', 'DELHI'),
('FC ROAD', 'PUNE'),
```

```
('BASAVANAGUDI', 'BANGALORE');

-- Customers
INSERT INTO Customers VALUES
('Anil', 'DELHI'),
('Ravi', 'PUNE'),
('Sneha', 'PUNE'),
('Fatima', 'DELHI'),
('Raj', 'BANGALORE');

-- Deposit
INSERT INTO Deposit VALUES
(101, 'Anil', 'KAROLBAGH', 3000, '1998-02-15'),
(102, 'Ravi', 'FC ROAD', 5000, '1999-03-10'),
(103, 'Sneha', 'FC ROAD', 8000, '1997-12-25'),
(104, 'Fatima', 'KAROLBAGH', 4000, '1998-05-20'),
(105, 'Raj', 'BASAVANAGUDI', 4500, '1998-11-10');

-- Borrow
INSERT INTO Borrow VALUES
(201, 'Anil', 'KAROLBAGH', 12000),
(202, 'Fatima', 'KAROLBAGH', 15000),
(203, 'Ravi', 'FC ROAD', 8000),
(204, 'Raj', 'BASAVANAGUDI', 9000);

-- 1
SELECT d.amount
FROM Deposit d
JOIN Customers c ON d.cname = c.cname
WHERE c.city = (
    SELECT city FROM Customers WHERE cname = 'Anil'
);

-- 2
SELECT
    SUM(amount) AS Total_Loan,
    MAX(amount) AS Max_Loan
FROM Borrow
WHERE bname = 'KAROLBAGH';
```

```
-- 3
SELECT SUM(amount) AS Total_Deposit
FROM Deposit
WHERE adate > '1998-01-01';

-- 4
SELECT MAX(d.amount) AS Max_Deposit
FROM Deposit d
JOIN Customers c ON d.cname = c.cname
WHERE c.city = 'PUNE';
```

———

Q25 games vala mongo db

```
// Use a database
use GameDB

// Drop the collection if it exists
db.games.drop()

// Insert 5 game documents
db.games.insertMany([
  {
    name: "Cyber Rush",
    gametype: "Action",
    score: 85,
    achievements: ["Speed Demon", "Sharp Shooter"]
  },
  {
    name: "Mystic Valley",
    gametype: "Adventure",
    score: 92,
    achievements: ["Explorer", "Puzzle Solver", "Game Master"]
  },
  {
    name: "Racing Riot",
```

```
    gametype: "Racing",
    score: 97,
    achievements: ["Speed Demon", "Game Master", "Champion"]
  },
  {
    name: "Puzzle Panic",
    gametype: "Puzzle",
    score: 88,
    achievements: ["Brainiac", "Puzzle Solver"]
  },
  {
    name: "Stealth Ops",
    gametype: "Stealth",
    score: 79,
    achievements: ["Silent Killer", "Game Master"]
  }
])

// ----------------------------
// QUERIES
// ----------------------------

// 1. Return all the games
print("All games:")
db.games.find().pretty()

// 2. Return the 3 highest scored games
print("Top 3 highest scored games:")
db.games.find().sort({ score: -1 }).limit(3).pretty()

// 3. Return games with both 'Game Master' and 'Speed Demon'
        achievements
print("Games with both 'Game Master' and 'Speed Demon':")
db.games.find({
  achievements: { $all: ["Game Master", "Speed Demon"] }
}).pretty()

// 4. Return all games with a score above 85
print("Games with score > 85:")
```

```
db.games.find({ score: { $gt: 85 } }).pretty()

// 5. Return all games of type 'Puzzle'
print("Games with gametype 'Puzzle':")
db.games.find({ gametype: "Puzzle" }).pretty()
```

Q26 Write a PL/SQL code to calculate tax for an employee of an organization ABC and to display his/her name & tax, by creating a table under employee database as below:
Employee_salary(emp_no,basic,HRA,DA,Total_deduction,net_salary,gross_Salary)

```
CREATE TABLE Employee_salary (
    empno INT PRIMARY KEY,
    ename VARCHAR(100),
    basic INT,
    HRA INT,
    DA INT,
    total_deduction INT
);

INSERT INTO Employee_salary (empno, ename, basic, HRA, DA,
    total_deduction) VALUES
(1, 'Alice', 20000, 5000, 3000, 2000),     -- Net: 26000 -> No tax
(2, 'Bob', 25000, 7000, 4000, 3000),       -- Net: 33000 -> 10% tax
(3, 'Charlie', 30000, 10000, 7000, 5000),-- Net: 42000 -> 10% tax
(4, 'David', 40000, 15000, 10000, 5000), -- Net: 60000 -> 20% tax
(5, 'Eve', 10000, 2000, 1000, 1000);       -- Net: 12000 -> No tax
```

```
DELIMITER $$

CREATE PROCEDURE cal_tax(empo int)
begin
declare v_gross int;
    declare v_net int;
    declare tax decimal(10,2);

    select basic+HRA+DA,basic+HRA+DA-total_deduction
    into v_gross,v_net
    from Employee_salary
    where empno = empo;

    if v_net > 50000 then
       set tax = v_net * 0.20;
elseif v_net>30000 and v_net<=50000 then
       set tax = v_net * 0.10;
end if;

    select empo as Employee_no,
       v_net as Net_salary,
         tax as tax_amount;

end $$
DELIMITER ;

call cal_tax(2);
```

——

Q28 Write a PL/SQL block of code using parameterized 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 n_rollcall (
    rollno INT,
```

```
    name VARCHAR(100)
);

CREATE TABLE o_rollcall (
    rollno INT,
    name VARCHAR(100)
);

INSERT INTO n_rollcall (rollno, name) VALUES
(101, 'John Smith'),
(102, 'Emma Johnson'),
(103, 'Michael Brown'),
(104, 'Sophia Davis'),
(105, 'William Wilson');

-- Insert some initial data into o_rollcall (some overlap with n_rollcall)
INSERT INTO o_rollcall (rollno, name) VALUES
(101, 'John Smith'),
(103, 'Michael Brown'),
(106, 'Olivia Martin');




DELIMITER $$
CREATE PROCEDURE main()
begin
declare roll int;
    declare sname varchar(100);
    declare done int default 0;

    declare cur cursor for
        select rollno,name from n_rollcall;
declare continue handler for not found set done = 1;

    open cur;
    read_loop:LOOP
        fetch cur into roll,sname;
```

```
        if done then
            leave read_loop;
        end if;

        if not exists(select 1 from o_rollcall where rollno = roll AND name =
        sname) then
            insert into o_rollcall(rollno,name)
                values(roll,sname);
        end if;
end loop;
    close cur;
end $$
DELIMITER ;

call main();
select * from o_rollcall order by rollno;
```

——

Q29Writ a PL/SQL procedure to find the number of students ranging from 100-70%, 69-60%, 59-50% & below 49% in each course from the student_course table given by the procedure as parameter.
Schema: Student (ROLL_NO ,COURSE, COURSE_COD ,SEM ,TOTAL_MARKS, PERCENTAGE)

```
use final;
drop table student;
CREATE TABLE stud (
    rollno INT PRIMARY KEY,
    name VARCHAR(100),
    course VARCHAR(50),
    percentage DECIMAL(5,2)
);
INSERT INTO stud (rollno, name, course, percentage) VALUES
(1, 'Alice', 'CS', 72.5),
(2, 'Bob', 'CS', 68.0),
(3, 'Charlie', 'CS', 55.0),
```

```
(4, 'David', 'CS', 42.0),
(5, 'Eve', 'CS', 87.0),
(6, 'Frank', 'Math', 74.0),
(7, 'Grace', 'Math', 61.0),
(8, 'Heidi', 'Math', 53.0),
(9, 'Ivan', 'Math', 45.0),
(10, 'Judy', 'Math', 91.0);



DELIMITER $$

CREATE PROCEDURE COUNT_IN_PERCENTAGE(inp_course VARCHAR(50))
begin
 declare count70 int default 0;
      declare count60 int default 0;
      declare count50 int default 0;
      declare count0 int default 0;

      select sum(case when percentage between 70 and 100 then 1 else 0
         end),
      sum(case when percentage between 60 and 69 then 1 else 0 end),
      sum(case when percentage between 50 and 59 then 1 else 0 end),
      sum(case when percentage between 0 and 49 then 1 else 0 end)
      into count70,count60,count50,count0
      from stud
      where course = inp_course;

      select count70,count60,count50,count0;
end$$
DELIMITER ;

call COUNT_IN_PERCENTAGE("CS");
```

——

Q30 Write a 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 sclass, if marks 899 and 825 category is Higher Second Class .

Consider Schema as Stud_Marks(name, total_marks) and Result(Roll,Name, Class)

```
create table stud_marks(
        name varchar(100),
    total_marks int
);
create table Result(
        roll int AUTO_INCREMENT PRIMARY KEY,
    name varchar(100),
    class varchar(150)
);

INSERT INTO stud_marks (name, total_marks) VALUES
('John Smith', 1200),         -- Distinction
('Emma Davis', 1020),          -- Distinction
('Michael Johnson', 989),   -- First Class
('Sarah Wilson', 950),        -- First Class
('David Brown', 900),          -- First Class
('Jennifer Lee', 880),        -- Higher Second Class
('Robert Miller', 825),       -- Higher Second Class
('Lisa Garcia', 800),         -- Pass/Fail
('Kevin Martinez', 750),     -- Pass/Fail
('Sophia Taylor', 1450);     -- Distinction

DELIMITER $$

create procedure category()
begin
        declare sname varchar(100);
```

```
    declare tmarks INT;
    declare class varchar(100);
    declare done int default 0;

    declare cur cursor for
            select name,total_marks from stud_marks;
       declare continue handler for not found set done = 1;

    open cur;
    read_loop:LOOP
            fetch cur into sname,tmarks;
        if done then
                leave read_loop;
            end if;

        if tmarks between 990 and 1400 then
                set class = 'Distinction';
            elseif tmarks between 900 and 989 then
                set class = 'First class';
            elseif tmarks between 800 and 899 then
                set class = 'Second Higher class';
            else
                set class = 'Fail';
            end if;

        insert into result(name,class) values(sname,class);
       end loop;
    close cur;
END $$

DELIMITER ;

CALL CATEGORY();
SELECT * FROM RESULT;
```

——

Q31
.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.

```
// Use the desired database
use WorldDB

// Drop the collection if it exists
db.Citydetails.drop()

// Insert sample city documents
db.Citydetails.insertMany([
  {
    _id: 1,
    name: "Pune",
    area: 500,
    population: {
      total: 7000000,
      Adults: 5000000,
      seniorcitizens: 800000,
      sexratio: 930
    },
    geography: {
      avgtemp: 26,
      avgrainfall: 850,
      longitude: 73.8567,
      latitude: 18.5204
```

```
  }
},
{
  _id: 2,
  name: "Mumbai",
  area: 603,
  population: {
    total: 20400000,
    Adults: 15000000,
    seniorcitizens: 2500000,
    sexratio: 920
  },
  geography: {
    avgtemp: 28,
    avgrainfall: 2400,
    longitude: 72.8777,
    latitude: 19.0760
  }
},
{
  _id: 3,
  name: "Delhi",
  area: 1484,
  population: {
    total: 31800000,
    Adults: 25000000,
    seniorcitizens: 3000000,
    sexratio: 940
  },
  geography: {
    avgtemp: 25,
    avgrainfall: 800,
    longitude: 77.1025,
    latitude: 28.7041
  }
},
{
  _id: 4,
  name: "Nagpur",
```

```
      area: 227,
      population: {
        total: 3100000,
        Adults: 2200000,
        seniorcitizens: 400000,
        sexratio: 950
      },
      geography: {
        avgtemp: 27,
        avgrainfall: 1000,
        longitude: 79.0882,
        latitude: 21.1458
      }
    }
])

// ------------------------------
// QUERIES
// ------------------------------

// 1. Find the total population in Pune
print("Total population in Pune:")
db.Citydetails.find(
  { name: "Pune" },
  { _id: 0, name: 1, "population.total": 1 }
)

// 2. Return all cities with total population greater than 10 million
print("Cities with population > 10 million:")
db.Citydetails.find(
  { "population.total": { $gt: 10000000 } },
  { _id: 0, name: 1, "population.total": 1 }
)

// 3. Return the average population for each city
// (showing just the average for context – could be rephrased to compute
average across all cities too)
print("Average population per city (returning total):")
db.Citydetails.aggregate([
```

```
  {
    $project: {
      _id: 0,
      name: 1,
                          avg_population:    {    $avg:    ["$population.Adults",
"$population.seniorcitizens"] }
    }
  }
])

// 4. Return the city with minimum and maximum population
// Get both using $group and $accumulator logic
print("City with Min and Max population:")
db.Citydetails.aggregate([
  {
    $group: {
      _id: null,
      minCity: {
        $min: {
          total: "$population.total",
          name: "$name"
        }
      },
      maxCity: {
        $max: {
          total: "$population.total",
          name: "$name"
        }
      }
    }
  }
])
```

——

Q3232.Create database :**Citydetails(_id,name,area,population(total,Adults,seniorcitizens,sex**

ratio), 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.

```
// Use database
use WorldDB

// Drop existing collection if any
db.Citydetails.drop()

// Insert documents
db.Citydetails.insertMany([
  {
    _id: 1,
    name: "Mumbai",
    area: 603,
    population: {
      total: 20400000,
      Adults: 15000000,
      seniorcitizens: 2500000,
      sexratio: 920
    },
    geography: {
      avgtemp: 28,
      avgrainfall: 2400,
      longitude: 72.8777,
      latitude: 19.0760
    }
  },
  {
    _id: 2,
    name: "Pune",
    area: 500,
    population: {
      total: 7000000,
      Adults: 5000000,
```

```
      seniorcitizens: 800000,
      sexratio: 930
    },
    geography: {
      avgtemp: 26,
      avgrainfall: 850,
      longitude: 73.8567,
      latitude: 18.5204
    }
  },
  {
    _id: 3,
    name: "Nagpur",
    area: 227,
    population: {
      total: 3100000,
      Adults: 2200000,
      seniorcitizens: 400000,
      sexratio: 950
    },
    geography: {
      avgtemp: 27,
      avgrainfall: 1000,
      longitude: 79.0882,
      latitude: 21.1458
    }
  },
  {
    _id: 4,
    name: "Delhi",
    area: 1484,
    population: {
      total: 31800000,
      Adults: 25000000,
      seniorcitizens: 3000000,
      sexratio: 940
    },
    geography: {
      avgtemp: 25,
```

```
        avgrainfall: 800,
        longitude: 77.1025,
        latitude: 28.7041
      }
    }
])

// ----------------------------
// 1. Find area-wise total population and sort by area ascending
print("Area-wise total population (sorted):")
db.Citydetails.aggregate([
  {
    $project: {
      _id: 0,
      area: 1,
      total_population: "$population.total"
    }
  },
  { $sort: { area: 1 } }
])

// ----------------------------
// 2. Retrieve name and area where average rainfall is greater than 60
print("Cities with average rainfall > 60:")
db.Citydetails.find(
  { "geography.avgrainfall": { $gt: 60 } },
  { _id: 0, name: 1, area: 1 }
)

// ----------------------------
// 3. Create index on city (name) and area, then find max population in Mumbai
db.Citydetails.createIndex({ name: 1, area: 1 })

print("Max population in Mumbai:")
db.Citydetails.find(
  { name: "Mumbai" },
  { _id: 0, name: 1, "population.total": 1 }
).sort({ "population.total": -1 }).limit(1)
```

```
// ----------------------------
// 4. Create index on name
db.Citydetails.createIndex({ name: 1 })

print("Indexes created successfully.")
```