1: 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.

```
-- Drop tables if they exist (for rerun safety)
DROP TABLE IF EXISTS Transaction;
DROP TABLE IF EXISTS Loan;
DROP TABLE IF EXISTS Account;
DROP TABLE IF EXISTS Customer;
DROP TABLE IF EXISTS Branch;

-- 1. Customer Table
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Address VARCHAR(200),
    Phone VARCHAR(15) UNIQUE
);

-- 2. Branch Table
CREATE TABLE Branch (
    BranchID INT PRIMARY KEY,
    BranchName VARCHAR(100) NOT NULL,
    Location VARCHAR(100)
);

-- 3. Account Table (with AUTO_INCREMENT and constraints)
CREATE TABLE Account (
    AcctNo INT AUTO_INCREMENT PRIMARY KEY,
    CustomerID INT,
    BranchID INT,
    Balance DECIMAL(12, 2) DEFAULT 0.0,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    FOREIGN KEY (BranchID) REFERENCES Branch(BranchID)
);

-- Set AUTO_INCREMENT start value to 1001
ALTER TABLE Account AUTO_INCREMENT = 1001;

-- 4. Loan Table
CREATE TABLE Loan (
    LoanID INT PRIMARY KEY,
    CustomerID INT,
    BranchID INT,
    Amount DECIMAL(12, 2),
    Status VARCHAR(20),
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    FOREIGN KEY (BranchID) REFERENCES Branch(BranchID)
);
```

```
-- 5. Transaction Table
CREATE TABLE Transaction (
    TransID INT PRIMARY KEY,
    AcctNo INT,
    TransDate DATE,
    Amount DECIMAL(10, 2),
    Type VARCHAR(10),
    FOREIGN KEY (AcctNo) REFERENCES Account(AcctNo),
    CHECK (Type IN ('Credit', 'Debit'))
);

-- 6. Views

-- View 1: Customer and Account Info
CREATE VIEW Customer_Account_View AS
SELECT c.CustomerID, c.Name, a.AcctNo, a.Balance
FROM Customer c
JOIN Account a ON c.CustomerID = a.CustomerID;

-- View 2: Loan and Branch Info
CREATE VIEW Loan_Branch_View AS
SELECT l.LoanID, c.Name AS CustomerName, b.BranchName, l.Amount, l.Status
FROM Loan l
JOIN Customer c ON l.CustomerID = c.CustomerID
JOIN Branch b ON l.BranchID = b.BranchID;

-- 7. Create Index on Customer Name
CREATE INDEX CustomerId ON Customer(Name ASC);
```

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.

```
-- Clean-up: Drop existing tables if they exist
DROP TABLE IF EXISTS WorksOn;
DROP TABLE IF EXISTS Project;
DROP TABLE IF EXISTS Employee;
DROP TABLE IF EXISTS Department;

-- 1. Department Table
CREATE TABLE Department (
    DeptID INT PRIMARY KEY,
    DeptName VARCHAR(100) NOT NULL UNIQUE,
    Location VARCHAR(100)
);

-- 2. Employee Table with AUTO_INCREMENT (as sequence)
CREATE TABLE Employee (
    EmpID INT AUTO_INCREMENT PRIMARY KEY,
    EmpName VARCHAR(100) NOT NULL,
    DeptID INT,
    Salary DECIMAL(10,2),
    FOREIGN KEY (DeptID) REFERENCES Department(DeptID)
);

-- Set starting value for Employee ID sequence
ALTER TABLE Employee AUTO_INCREMENT = 1001;

-- 3. Project Table
CREATE TABLE Project (
    ProjID INT PRIMARY KEY,
    ProjName VARCHAR(100) NOT NULL,
    DeptID INT,
    Budget DECIMAL(12,2),
    FOREIGN KEY (DeptID) REFERENCES Department(DeptID)
);

-- 4. WorksOn Table (Many-to-Many: Employee <-> Project)
CREATE TABLE WorksOn (
    EmpID INT,
    ProjID INT,
    HoursWorked DECIMAL(5,2),
    PRIMARY KEY (EmpID, ProjID),
    FOREIGN KEY (EmpID) REFERENCES Employee(EmpID),
    FOREIGN KEY (ProjID) REFERENCES Project(ProjID)
);

-- 5. View 1: High-paid employees in departments
CREATE VIEW HighBudgetEmployees AS
SELECT e.EmpID, e.EmpName, d.DeptName, e.Salary
FROM Employee e
JOIN Department d ON e.DeptID = d.DeptID
WHERE e.Salary > 80000;

-- 6. View 2: Project assignments with more than 20 hours
CREATE VIEW ActiveAssignments AS
SELECT w.EmpID, e.EmpName, w.ProjID, p.ProjName, w.HoursWorked
FROM WorksOn w
JOIN Employee e ON w.EmpID = e.EmpID
JOIN Project p ON w.ProjID = p.ProjID
```

```
WHERE w.HoursWorked > 20;

-- 7. Composite Index: DeptID + EmpID (named EmployeeId)
CREATE INDEX EmployeeId ON Employee (DeptID ASC, EmpID ASC);
```

3: write 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.

```
-- Drop tables if they exist
DROP TABLE IF EXISTS Library_Audit;
DROP TABLE IF EXISTS Library;

-- 1. Library Table
CREATE TABLE Library (
   bid INT PRIMARY KEY,
   bname VARCHAR(100),
   doi DATE,
   status VARCHAR(20)
);

-- 2. Library_Audit Table
CREATE TABLE Library_Audit (
   bid INT,
   bname VARCHAR(100),
   noc INT,
   timestampofquery TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   fine INT DEFAULT 0
);
DELIMITER $$

CREATE TRIGGER trg_after_insert_update_status
AFTER INSERT ON Library
FOR EACH ROW
BEGIN
   DECLARE copies INT DEFAULT 0;

   -- If status is ISSUE: decrease number of copies by 1
   IF NEW.status = 'ISSUE' THEN
      SET copies = -1;
   -- If status is RETURN: increase number of copies by 1
   ELSEIF NEW.status = 'RETURN' THEN
      SET copies = 1;
   ELSE
      SET copies = 0;
   END IF;
```

```
    INSERT INTO Library_Audit (bid, bname, noc)
    VALUES (NEW.bid, NEW.bname, copies);
END$$

DELIMITER ;
DELIMITER $$

CREATE TRIGGER trg_after_update_fine
AFTER UPDATE ON Library
FOR EACH ROW
BEGIN
    DECLARE days_diff INT;
    DECLARE fine_amt INT;

    SET days_diff = DATEDIFF(CURRENT_DATE, NEW.doi);

    IF days_diff > 20 THEN
        -- Update status to FINE in Library
        UPDATE Library
        SET status = 'FINE'
        WHERE bid = NEW.bid;

        SET fine_amt = days_diff * 10;

        -- Insert audit record with fine
        INSERT INTO Library_Audit (bid, bname, noc, fine)
        VALUES (NEW.bid, NEW.bname, 0, fine_amt);
    END IF;
END$$

DELIMITER ;
```

4: 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.

```
-- Clean-up: Drop existing tables if they exist
DROP TABLE IF EXISTS Library_Audit;
DROP TABLE IF EXISTS Library;

-- Step 1: Create the main Library table
CREATE TABLE Library (
   bid INT PRIMARY KEY,
   bname VARCHAR(100),
   doi DATE,
   status VARCHAR(20)
);

-- Step 2: Create the audit table to store old values and operations
CREATE TABLE Library_Audit (
   bid INT,
   bname VARCHAR(100),
   doi DATE,
   status VARCHAR(20),
   operation_type VARCHAR(10),  -- 'UPDATE' or 'DELETE'
   action_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Step 3: Trigger to capture old values after UPDATE
DELIMITER $$

CREATE TRIGGER trg_after_update_library
AFTER UPDATE ON Library
FOR EACH ROW
BEGIN
   INSERT INTO Library_Audit (bid, bname, doi, status, operation_type)
   VALUES (OLD.bid, OLD.bname, OLD.doi, OLD.status, 'UPDATE');
END$$

DELIMITER ;

-- Step 4: Trigger to capture old values after DELETE
DELIMITER $$

CREATE TRIGGER trg_after_delete_library
AFTER DELETE ON Library
FOR EACH ROW
BEGIN
   INSERT INTO Library_Audit (bid, bname, doi, status, operation_type)
   VALUES (OLD.bid, OLD.bname, OLD.doi, OLD.status, 'DELETE');
END$$

DELIMITER ;
```

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

```
// Step 1: Switch to or create a database
use webanalytics;

// Step 2: Create the 'sites' collection with sample access logs
db.sites.drop();
db.sites.insertMany([
  { url: "example.com", dateofaccess: ISODate("2025-05-01T10:00:00Z") },
  { url: "example.com", dateofaccess: ISODate("2025-05-03T12:00:00Z") },
  { url: "openai.com", dateofaccess: ISODate("2025-05-02T11:00:00Z") },
  { url: "openai.com", dateofaccess: ISODate("2025-04-20T09:00:00Z") },
  { url: "example.com", dateofaccess: ISODate("2025-04-01T14:00:00Z") },
  { url: "openai.com", dateofaccess: ISODate("2025-04-15T14:00:00Z") },
  { url: "example.com", dateofaccess: ISODate("2025-03-10T16:00:00Z") }
]);

// Step 3: Define the map function
var mapFunction = function() {
    var month = this.dateofaccess.getMonth() + 1;  // JS months are 0-based
    var year = this.dateofaccess.getFullYear();
    emit({ url: this.url, month: month, year: year }, 1);
};

// Step 4: Define the reduce function
var reduceFunction = function(key, values) {
    return Array.sum(values);
};

// Step 5: Run the MapReduce job
```

```
db.sites.mapReduce(
    mapFunction,
    reduceFunction,
    {
        out: "site_monthly_access"
    }
);

// Step 6: View the results
db.site_monthly_access.find().pretty();
```

6 Create tables CitiesIndia(<u>pincode</u>,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.

```
-- Step 1: Drop tables if they exist
DROP TABLE IF EXISTS Categories;
DROP TABLE IF EXISTS CitiesIndia;

-- Step 2: Create CitiesIndia table
CREATE TABLE CitiesIndia (
   pincode INT PRIMARY KEY,
   nameofcity VARCHAR(50),
   earliername VARCHAR(50),
   area FLOAT,         -- in square km
   population INT,
   avgrainfall FLOAT   -- in mm
);

-- Step 3: Create Categories table
CREATE TABLE Categories (
   Type VARCHAR(20),
   pincode INT,
   FOREIGN KEY (pincode) REFERENCES CitiesIndia(pincode)
);

-- Step 4: Insert sample data (modify as needed)
INSERT INTO CitiesIndia VALUES (411001, 'Pune', 'Poona', 150, 22000, 700);
INSERT INTO CitiesIndia VALUES (400001, 'Mumbai', 'Bombay', 603, 25000, 845);
INSERT INTO CitiesIndia VALUES (110001, 'Delhi', 'Indraprastha', 1484, 12000, 800);
INSERT INTO CitiesIndia VALUES (800001, 'Patna', 'Pataliputra', 320, 18000, 1005);
INSERT INTO CitiesIndia VALUES (500001, 'Hyderabad', 'Bhagyanagar', 650, 9, 950);  -- will trigger error

-- Step 5: Create a function to compute density
DELIMITER //
CREATE FUNCTION get_density(pop INT, area FLOAT)
RETURNS FLOAT
DETERMINISTIC
BEGIN
   IF pop < 10 OR pop > 25718 THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Population must be between 10 and 25718';
   END IF;
   RETURN pop / area;
END;
//
DELIMITER ;

-- Step 6: Create procedure to categorize cities based on density
DELIMITER //
```

```
CREATE PROCEDURE categorize_density()
BEGIN
   DECLARE done INT DEFAULT FALSE;
   DECLARE p_pin INT;
   DECLARE p_pop INT;
   DECLARE p_area FLOAT;
   DECLARE density FLOAT;
   DECLARE cur CURSOR FOR SELECT pincode, population, area FROM CitiesIndia;
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

   OPEN cur;
   read_loop: LOOP
      FETCH cur INTO p_pin, p_pop, p_area;
      IF done THEN
         LEAVE read_loop;
      END IF;

      BEGIN
         -- Use function to calculate density
         SET density = get_density(p_pop, p_area);

         IF density > 3000 THEN
            INSERT INTO Categories VALUES ('High Density', p_pin);
         ELSEIF density >= 1000 AND density <= 3000 THEN
            INSERT INTO Categories VALUES ('Moderate', p_pin);
         ELSE
            INSERT INTO Categories VALUES ('Low Density', p_pin);
         END IF;

      END;
   END LOOP;
   CLOSE cur;
END;
//
DELIMITER ;

-- Step 7: Call the procedure
CALL categorize_density();

-- Step 8: View results
SELECT * FROM Categories;
```

7 Write 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 .

```
-- Drop tables if exist
DROP TABLE IF EXISTS Student;
DROP TABLE IF EXISTS Result;

-- Create Student table
CREATE TABLE Student (
  rollno INT PRIMARY KEY,
  name VARCHAR(50),
  Marks1 INT,
  Marks2 INT,
  Marks3 INT,
  Marks4 INT,
  Marks5 INT
);

-- Create Result table
CREATE TABLE Result (
  rollno INT PRIMARY KEY,
  total_marks INT,
  class VARCHAR(20)
);

-- Insert sample data
INSERT INTO Student VALUES (1, 'Alice', 95, 98, 97, 100, 96);
INSERT INTO Student VALUES (2, 'Bob', 85, 80, 79, 88, 82);
INSERT INTO Student VALUES (3, 'Charlie', 65, 70, 75, 60, 68);
INSERT INTO Student VALUES (4, 'David', 101, 90, 85, 88, 92); -- Invalid marks to test error
INSERT INTO Student VALUES (5, 'Eve', 80, -5, 90, 85, 87); -- Invalid marks to test error

-- Create function to calculate total marks and validate input
DELIMITER //
CREATE FUNCTION validate_total( m1 INT, m2 INT, m3 INT, m4 INT, m5 INT ) RETURNS INT
BEGIN
  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 = 'Marks must be between 0 and 100';
  END IF;
  RETURN m1 + m2 + m3 + m4 + m5;
END;
//
DELIMITER ;

-- Create procedure to classify and insert results
DELIMITER //
CREATE PROCEDURE classify_students()
```

```
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE r INT;
  DECLARE n VARCHAR(50);
  DECLARE m1, m2, m3, m4, m5 INT;
  DECLARE total INT;
  DECLARE class_result VARCHAR(20);

  DECLARE cur CURSOR FOR SELECT rollno, name, Marks1, Marks2, Marks3, Marks4, Marks5 FROM Student;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN
    -- Skip invalid rows with marks error
    SET done = TRUE;
  END;

  DELETE FROM Result; -- clear old data

  OPEN cur;

  read_loop: LOOP
    FETCH cur INTO r, n, m1, m2, m3, m4, m5;
    IF done THEN
      LEAVE read_loop;
    END IF;

    -- Validate marks and calculate total
    BEGIN
      SET total = validate_total(m1, m2, m3, m4, m5);
    EXCEPTION
      WHEN SQLEXCEPTION THEN
        -- Skip student with invalid marks
        ITERATE read_loop;
    END;

    -- Determine class based on total marks
    IF total BETWEEN 990 AND 1499 THEN
      SET class_result = 'Distinction';
    ELSEIF total BETWEEN 900 AND 989 THEN
      SET class_result = 'First Class';
    ELSEIF total BETWEEN 825 AND 899 THEN
      SET class_result = 'Higher Second';
    ELSEIF total BETWEEN 750 AND 824 THEN
      SET class_result = 'Second';
    ELSEIF total < 750 THEN
      SET class_result = 'Pass';
    ELSE
      SET class_result = 'Fail';
    END IF;

    -- Insert result
    INSERT INTO Result(rollno, total_marks, class) VALUES (r, total, class_result);

  END LOOP;

  CLOSE cur;
END;
//
DELIMITER ;
```

```
-- Call procedure
CALL classify_students();

-- View results
SELECT * FROM Result;
```

8 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.

```
DROP TABLE IF EXISTS Loan;
DROP TABLE IF EXISTS Book;
DROP TABLE IF EXISTS Member;
DROP TABLE IF EXISTS Author;
DROP TABLE IF EXISTS Publisher;

CREATE TABLE Author (
  Authorid INT PRIMARY KEY AUTO_INCREMENT,
  AuthorName VARCHAR(100) NOT NULL,
  Bio TEXT
```

```
);

CREATE TABLE Publisher (
  Publisherid INT PRIMARY KEY AUTO_INCREMENT,
  PublisherName VARCHAR(100) NOT NULL,
  Address VARCHAR(200)
);

CREATE TABLE Book (
  Bookid INT PRIMARY KEY AUTO_INCREMENT,
  Title VARCHAR(150) NOT NULL,
  Authorid INT,
  Publisherid INT,
  Year YEAR,
  FOREIGN KEY (Authorid) REFERENCES Author(Authorid),
  FOREIGN KEY (Publisherid) REFERENCES Publisher(Publisherid)
);

CREATE TABLE Member (
  Memberid INT PRIMARY KEY AUTO_INCREMENT,
  Name VARCHAR(100) NOT NULL,
  Address VARCHAR(200),
  Phone VARCHAR(15)
);

CREATE TABLE Loan (
  Loanid INT PRIMARY KEY AUTO_INCREMENT,
  Bookid INT,
  Memberid INT,
  LoanDate DATE,
  ReturnDate DATE,
  FOREIGN KEY (Bookid) REFERENCES Book(Bookid),
  FOREIGN KEY (Memberid) REFERENCES Member(Memberid)
);

CREATE VIEW BookDetails AS
SELECT b.Bookid, b.Title, a.AuthorName, p.PublisherName, b.Year
FROM Book b
JOIN Author a ON b.Authorid = a.Authorid
JOIN Publisher p ON b.Publisherid = p.Publisherid;

CREATE VIEW LoanDetails AS
SELECT l.Loanid, m.Name AS MemberName, b.Title AS BookTitle, l.LoanDate, l.ReturnDate
FROM Loan l
JOIN Member m ON l.Memberid = m.Memberid
JOIN Book b ON l.Bookid = b.Bookid;

CREATE INDEX Lib_Index1 ON Author (AuthorName ASC);
```

9.PL/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, Status)
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.**

```
DELIMITER $$

CREATE PROCEDURE ReturnBook(
    IN p_roll_no INT,
    IN p_book_name VARCHAR(255)
)
BEGIN
    DECLARE v_date_of_issue DATE;
    DECLARE v_status CHAR(1);
    DECLARE v_days INT;
    DECLARE v_fine_amt INT DEFAULT 0;
    DECLARE v_bid INT;
    DECLARE v_noc INT;
    DECLARE exit handler FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error occurred during the process.' AS Message;
    END;

    START TRANSACTION;

    -- Get borrower's issue date and status
    SELECT DateofIssue, Status INTO v_date_of_issue, v_status
    FROM Borrower
    WHERE Rollin = p_roll_no AND NameofBook = p_book_name
    LIMIT 1;

    IF v_status = 'R' THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Book already returned.';
    END IF;

    -- Calculate days difference
    SET v_days = DATEDIFF(CURDATE(), v_date_of_issue);

    -- Calculate fine amount
    IF v_days BETWEEN 15 AND 30 THEN
        SET v_fine_amt = v_days * 5;
    ELSEIF v_days > 30 THEN
        SET v_fine_amt = (30 * 5) + ((v_days - 30) * 50);
    ELSE
        SET v_fine_amt = 0;
    END IF;

    -- Update Borrower status to 'R' (Returned)
    UPDATE Borrower
    SET Status = 'R'
    WHERE Rollin = p_roll_no AND NameofBook = p_book_name;

    -- Get book id and noc from Library
    SELECT bid, noc INTO v_bid, v_noc
    FROM Library
    WHERE bname = p_book_name
    LIMIT 1;

    -- Increase noc by 1
    UPDATE Library
    SET noc = noc + 1
```

```
    WHERE bid = v_bid;

    -- Insert transaction record
    INSERT INTO transaction (bid, bname, status)
    VALUES (v_bid, p_book_name, 'RETURN');

    -- Insert fine record if fine_amt > 0
    IF v_fine_amt > 0 THEN
        INSERT INTO Fine (Roll_no, Date, Amt)
        VALUES (p_roll_no, CURDATE(), v_fine_amt);
    END IF;

    COMMIT;

    SELECT CONCAT('Book returned successfully. Fine amount: Rs ', v_fine_amt) AS Message;

END $$

DELIMITER ;
CALL ReturnBook(101, 'The Great Gatsby');
```

10 Implement 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)

```
-- 1. Create Table Borrower

CREATE TABLE Borrower (

    Rollin INT NOT NULL,

    Name VARCHAR(100) NOT NULL,

    DateofIssue DATE NOT NULL,

    NameofBook VARCHAR(255) NOT NULL,

    Status CHAR(1) CHECK (Status IN ('I', 'R')), -- Issued or Returned

    PRIMARY KEY (Rollin, NameofBook)

);
```

```
-- 2. Create View to show currently issued books with borrower details
CREATE VIEW IssuedBooks AS
SELECT Rollin, Name, NameofBook, DateofIssue
FROM Borrower
WHERE Status = 'I';

-- 3. Create View as a synonym-like object for Borrower (synonym not supported in MySQL)
CREATE VIEW Borrower_Synonym AS
SELECT * FROM Borrower;

-- 4. Create Index on Rollin to speed up queries by Rollin
CREATE INDEX idx_Rollin ON Borrower (Rollin);

-- 5. Create Index on NameofBook for faster book searches
CREATE INDEX idx_BookName ON Borrower (NameofBook);

-- 6. Sequence simulation: For Rollin, if needed, create an auto-increment helper table or use AUTO_INCREMENT
-- Since Rollin is NOT auto-increment here (assuming user inputs it),
-- if you want a sequence for another ID, you create a table with AUTO_INCREMENT.

-- Example: Create a sequence table to generate unique transaction IDs
CREATE TABLE BorrowerSeq (
    id INT NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id)
);

-- To get next sequence value:
-- INSERT INTO BorrowerSeq VALUES (NULL);
```

```
-- SELECT LAST_INSERT_ID();
```

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

```
-- DROP tables if exist (for clean run)
DROP TABLE IF EXISTS Employee;
DROP TABLE IF EXISTS Department;

-- 1. Create Department table
CREATE TABLE Department (
    DeptID INT PRIMARY KEY,
    DeptName VARCHAR(100) NOT NULL
);

-- 2. Create Employee table
CREATE TABLE Employee (
    EmpID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    DeptID INT,
    FOREIGN KEY (DeptID) REFERENCES Department(DeptID)
);

-- 3. Insert sample data into Department
INSERT INTO Department VALUES
(1, 'HR'),
(2, 'Sales'),
(3, 'IT'),
(4, 'Marketing');

-- 4. Insert sample data into Employee
INSERT INTO Employee VALUES
(101, 'Alice', 1),
(102, 'Bob', 2),
(103, 'Charlie', 2),
(104, 'David', NULL),   -- No department
(105, 'Eva', 3),
(106, 'Frank', 5);      -- Invalid DeptID to demonstrate joins

-- ===== JOINS =====

-- INNER JOIN: Employees with valid departments
SELECT e.Name AS Employee, d.DeptName
FROM Employee e
INNER JOIN Department d ON e.DeptID = d.DeptID;

-- LEFT JOIN: All employees with their departments if any
SELECT e.Name AS Employee, d.DeptName
```

```
FROM Employee e
LEFT JOIN Department d ON e.DeptID = d.DeptID;

-- RIGHT JOIN: All departments with employees if any
SELECT e.Name AS Employee, d.DeptName
FROM Employee e
RIGHT JOIN Department d ON e.DeptID = d.DeptID;

-- FULL OUTER JOIN workaround using UNION of LEFT and RIGHT JOINs
SELECT e.Name AS Employee, d.DeptName
FROM Employee e
LEFT JOIN Department d ON e.DeptID = d.DeptID
UNION
SELECT e.Name AS Employee, d.DeptName
FROM Employee e
RIGHT JOIN Department d ON e.DeptID = d.DeptID;

-- ===== SUBQUERIES =====

-- Scalar subquery: Department name per employee
SELECT Name,
   (SELECT DeptName FROM Department WHERE DeptID = Employee.DeptID) AS DeptName
FROM Employee;

-- Correlated subquery: Employees in departments with more than 1 employee
SELECT Name, DeptID
FROM Employee e1
WHERE (SELECT COUNT(*) FROM Employee e2 WHERE e2.DeptID = e1.DeptID) > 1;

-- IN subquery: Employees in Sales or Marketing departments
SELECT Name FROM Employee
WHERE DeptID IN (SELECT DeptID FROM Department WHERE DeptName IN ('Sales', 'Marketing'));

-- EXISTS subquery: Employees with a valid department
SELECT Name FROM Employee e
WHERE EXISTS (SELECT 1 FROM Department d WHERE d.DeptID = e.DeptID);

-- ===== VIEWS =====

-- Create view showing employee with department name
CREATE OR REPLACE VIEW EmpDeptView AS
SELECT e.EmpID, e.Name, d.DeptName
FROM Employee e
LEFT JOIN Department d ON e.DeptID = d.DeptID;

-- Query the view for employees in 'Sales'
SELECT * FROM EmpDeptView WHERE DeptName = 'Sales';
```

12 Implement Indexing and querying with MongoDB using following example.

Students(stud_id, stud_name,stud_addr,stud_marks)

```
// Switch to or create a database
use schoolDB

// Insert documents into Students collection
db.Students.insertMany([
  { stud_id: 1, stud_name: "Alice", stud_addr: "New York", stud_marks: 85 },
  { stud_id: 2, stud_name: "Bob", stud_addr: "Chicago", stud_marks: 90 },
  { stud_id: 3, stud_name: "Charlie", stud_addr: "New York", stud_marks: 78 },
  { stud_id: 4, stud_name: "David", stud_addr: "San Francisco", stud_marks: 92 },
  { stud_id: 5, stud_name: "Eva", stud_addr: "Chicago", stud_marks: 88 }
])

// Create index on stud_name (ascending)
db.Students.createIndex({ stud_name: 1 })

// Find students from Chicago
db.Students.find({ stud_addr: "Chicago" })
```

```
// Find student named Alice
db.Students.find({ stud_name: "Alice" })

// Find students with marks > 80
db.Students.find({ stud_marks: { $gt: 80 } })

// Check if index is used (optional)
db.Students.find({ stud_name: "Bob" }).explain("executionStats")
```

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)

Perform following queries

1. **For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name,address, and birth date.**
2. **Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.**
3. **Retrieve all employees whose address is in Houston, Texas.**
4. **Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.**

```
-- DROP tables if they already exist to avoid errors
DROP TABLE IF EXISTS DEPENDENT, WORKS_ON, PROJECT, DEPT_LOCATIONS, DEPARTMENT, EMPLOYEE;

-- 1. CREATE TABLES
CREATE TABLE EMPLOYEE (
  Fname VARCHAR(20),
  Minit CHAR(1),
  Lname VARCHAR(20),
  Ssn CHAR(9) PRIMARY KEY,
  Bdate DATE,
  Address VARCHAR(100),
  Sex CHAR(1),
  Salary DECIMAL(10,2),
  Dno INT
);

CREATE TABLE DEPARTMENT (
  Dname VARCHAR(50),
  Dno INT PRIMARY KEY,
  Mgr_ssn CHAR(9),
  Mgr_start_date DATE,
  FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
);

CREATE TABLE DEPT_LOCATIONS (
  Dnumber INT,
  Dlocation VARCHAR(50),
  FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dno)
);

CREATE TABLE PROJECT (
  Pname VARCHAR(50),
  Pnumber INT PRIMARY KEY,
  Plocation VARCHAR(50),
  Dno INT,
  FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dno)
);

CREATE TABLE WORKS_ON (
  Essn CHAR(9),
  Pno INT,
  Hours DECIMAL(5,2),
  FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
  FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber)
);

CREATE TABLE DEPENDENT (
  Essn CHAR(9),
  Dependent_name VARCHAR(30),
  Sex CHAR(1),
  Bdate DATE,
  Relationship VARCHAR(20),
  FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn)
);

-- 2. INSERT SAMPLE DATA
INSERT INTO EMPLOYEE VALUES
```

```
('John', 'B', 'Smith', '123456789', '1990-01-09', 'Houston, Texas', 'M', 50000, 1),
('Alice', 'J', 'Brown', '987654321', '1985-03-17', 'Dallas, Texas', 'F', 75000, 2),
('Robert', 'K', 'Taylor', '456789123', '1979-07-23', 'Stafford, Texas', 'M', 65000, 1);

INSERT INTO DEPARTMENT VALUES
('Research', 1, '123456789', '2020-01-01'),
('Administration', 2, '987654321', '2019-06-15');

INSERT INTO DEPT_LOCATIONS VALUES
(1, 'Stafford'),
(2, 'Houston');

INSERT INTO PROJECT VALUES
('ProductX', 101, 'Stafford', 1),
('ProductY', 102, 'Dallas', 2),
('ProductZ', 103, 'Stafford', 1);

INSERT INTO WORKS_ON VALUES
('123456789', 101, 10.0),
('456789123', 101, 12.5),
('987654321', 102, 8.0);

INSERT INTO DEPENDENT VALUES
('123456789', 'Grace', 'F', '2010-08-12', 'Daughter'),
('456789123', 'Kyle', 'M', '2005-05-23', 'Son');

-- 3. QUERIES

-- a) For every project in ‘Stafford’, list project number, dept number, manager last name, address, birth date
SELECT
  P.Pnumber,
  D.Dno,
  E.Lname AS ManagerLastName,
  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 = 'Stafford';

-- b) List of all project numbers for projects involving employee named ‘Smith’ (as worker or manager)
SELECT DISTINCT P.Pnumber
FROM PROJECT P
LEFT JOIN DEPARTMENT D ON P.Dno = D.Dno
LEFT JOIN EMPLOYEE MGR ON D.Mgr_ssn = MGR.Ssn
LEFT JOIN WORKS_ON W ON P.Pnumber = W.Pno
LEFT JOIN EMPLOYEE EMP ON W.Essn = EMP.Ssn
WHERE
  MGR.Lname = 'Smith' OR EMP.Lname = 'Smith';

-- c) Retrieve all employees whose address is in Houston, Texas
SELECT *
FROM EMPLOYEE
WHERE Address LIKE '%Houston, Texas%';

-- d) Show salaries if every employee working on ‘ProductX’ gets 10% raise
```

```
SELECT
  E.Fname,
  E.Lname,
  E.Salary AS OldSalary,
  (E.Salary * 1.10) AS NewSalary
FROM EMPLOYEE E
JOIN WORKS_ON W ON E.Ssn = W.Essn
JOIN PROJECT P ON W.Pno = P.Pnumber
WHERE P.Pname = 'ProductX';
```

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 tables to avoid duplicates
DROP TABLE IF EXISTS Borrower, Loan, Depositor, Account, Customer, Branch;

-- 1. CREATE TABLES
CREATE TABLE Branch (
  branch_name VARCHAR(50) PRIMARY KEY,
  branch_city VARCHAR(50),
  assets DECIMAL(12,2)
);

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 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,
  FOREIGN KEY (cust_name) REFERENCES Customer(cust_name),
  FOREIGN KEY (acc_no) REFERENCES Account(acc_no)
);

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

-- 2. INSERT SAMPLE DATA
INSERT INTO Branch VALUES
('Pune Main', 'Pune', 5000000),
('Mumbai Central', 'Mumbai', 8000000),
('Delhi Hub', 'Delhi', 7000000);

INSERT INTO Account VALUES
(101, 'Pune Main', 10000),
(102, 'Pune Main', 15000),
(103, 'Mumbai Central', 12000),
(104, 'Mumbai Central', 20000),
(105, 'Delhi Hub', 8000);

INSERT INTO Customer VALUES
('Alice', 'MG Road', 'Pune'),
('Bob', 'Linking Road', 'Mumbai'),
('Charlie', 'Connaught Place', 'Delhi'),
('David', 'FC Road', 'Pune'),
('Eva', 'Colaba', 'Mumbai');

INSERT INTO Depositor VALUES
('Alice', 101),
('Bob', 103),
('Charlie', 105),
('David', 102),
```

```
('Eva', 104);

INSERT INTO Loan VALUES
(201, 'Pune Main', 50000),
(202, 'Mumbai Central', 75000),
(203, 'Delhi Hub', 40000);

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

-- 3. DML: UPDATE
UPDATE Account SET balance = balance + 1000 WHERE acc_no = 101;

-- 4. DML: DELETE
DELETE FROM Depositor WHERE cust_name = 'David';

-- 5. DML: SELECT + FUNCTIONS + OPERATORS

-- a) Average account balance at each branch
SELECT branch_name, AVG(balance) AS avg_balance
FROM Account
GROUP BY branch_name;

-- b) Number of depositors at each branch
SELECT A.branch_name, COUNT(DISTINCT D.cust_name) AS num_depositors
FROM Account A
JOIN Depositor D ON A.acc_no = D.acc_no
GROUP BY A.branch_name;

-- c) Branches where average account balance > 12000
SELECT branch_name
FROM Account
GROUP BY branch_name
HAVING AVG(balance) > 12000;

-- d) Number of tuples in Customer relation
SELECT COUNT(*) AS total_customers FROM Customer;

-- 6. Set Operator: Customers who are both depositors and borrowers
SELECT cust_name FROM Depositor
INTERSECT
SELECT cust_name FROM Borrower;

-- 7. UNION Example: All customer names involved in either loan or deposit
SELECT cust_name FROM Depositor
UNION
SELECT cust_name FROM Borrower;
```

15 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)

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

Solve following query:

1. **Find the names of all branches in loan relation.**
2. **Find all loan numbers for loans made at Akurdi Branch with loan amount > 12000.**
3. **Find all customers who have a loan from bank.**
4. **Find their names,loan_no and loan amount.**

```
-- DROP TABLES IF THEY EXIST
DROP TABLE IF EXISTS Borrower, Loan, Depositor, Account, Customer, Branch;

-- 1. CREATE TABLES WITH CONSTRAINTS
CREATE TABLE Branch (
  branch_name VARCHAR(50) PRIMARY KEY,
```

```
  branch_city VARCHAR(50) NOT NULL,
  assets DECIMAL(12,2) CHECK (assets >= 0)
);

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)
);

CREATE TABLE Customer (
  cust_name VARCHAR(50) PRIMARY KEY,
  cust_street VARCHAR(100) NOT NULL,
  cust_city VARCHAR(50) NOT NULL
);

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)
);

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)
);

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)
);

-- 2. INSERT SAMPLE DATA
INSERT INTO Branch VALUES
('Akurdi', 'Pune', 3000000),
('Deccan', 'Pune', 4000000),
('ShivajiNagar', 'Pune', 3500000);

INSERT INTO Account VALUES
(1001, 'Akurdi', 12000),
(1002, 'Akurdi', 8000),
(1003, 'Deccan', 15000),
(1004, 'ShivajiNagar', 20000);

INSERT INTO Customer VALUES
('Rahul', 'MG Road', 'Pune'),
('Sneha', 'Baner Road', 'Pune'),
('Amit', 'FC Road', 'Pune'),
('Neha', 'Kothrud', 'Pune');
```

```
INSERT INTO Depositor VALUES
('Rahul', 1001),
('Sneha', 1002),
('Amit', 1003),
('Neha', 1004);

INSERT INTO Loan VALUES
(2001, 'Akurdi', 25000),
(2002, 'Deccan', 10000),
(2003, 'Akurdi', 13000);

INSERT INTO Borrower VALUES
('Rahul', 2001),
('Neha', 2002),
('Sneha', 2003);

-- 3. DML: UPDATE + DELETE
UPDATE Loan SET amount = amount + 1000 WHERE loan_no = 2002;

DELETE FROM Borrower WHERE cust_name = 'Neha' AND loan_no = 2002;

-- 4. SOLVE REQUIRED QUERIES

-- a) Find the names of all branches in loan relation
SELECT DISTINCT branch_name FROM Loan;

-- b) Find all loan numbers for loans made at Akurdi Branch with loan amount > 12000
SELECT loan_no FROM Loan
WHERE branch_name = 'Akurdi' AND amount > 12000;

-- c) Find all customers who have a loan from bank
SELECT DISTINCT cust_name FROM Borrower;

-- d) Find their names, loan_no and loan amount
SELECT B.cust_name, B.loan_no, L.amount
FROM Borrower B
JOIN Loan L ON B.loan_no = L.loan_no;

-- 5. Set operator example: Customers who are both depositors and borrowers
SELECT cust_name FROM Depositor
INTERSECT
SELECT cust_name FROM Borrower;

-- 6. Function example: Find max, min, avg loan amount
SELECT MAX(amount) AS MaxLoan, MIN(amount) AS MinLoan, AVG(amount) AS AvgLoan FROM
Loan;

-- 7. Arithmetic & logical operator example: Loans with even amount and > 12000
SELECT loan_no, amount FROM Loan
WHERE MOD(amount, 2) = 0 AND amount > 12000;
```

16 Implement Map reduce operation with following example using MongoDB

Students(stud_id, stud_name,stud_addr,stud_marks)

AND

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

```
// Use your database
use school;

// Drop collection if it exists
db.Students.drop();

// Insert sample data
db.Students.insertMany([
  { stud_id: 1, stud_name: "Rahul", stud_addr: "Pune", stud_marks: 85 },
  { stud_id: 2, stud_name: "Sneha", stud_addr: "Mumbai", stud_marks: 78 },
  { stud_id: 3, stud_name: "Amit", stud_addr: "Delhi", stud_marks: 92 },
  { stud_id: 4, stud_name: "Neha", stud_addr: "Pune", stud_marks: 88 },
  { stud_id: 5, stud_name: "Karan", stud_addr: "Mumbai", stud_marks: 80 }
]);

// Define Map function
var mapFunction = function () {
```

```
  emit(this.stud_addr, this.stud_marks);
};

// Define Reduce function
var reduceFunction = function (key, values) {
  return Array.sum(values);
};

// Execute MapReduce
db.Students.mapReduce(
  mapFunction,
  reduceFunction,
  {
    out: "total_marks_per_city"
  }
);

// View result
db.total_marks_per_city.find().pretty();

-- Table Creation
CREATE TABLE student_marks (
  rollno NUMBER PRIMARY KEY,
  name VARCHAR2(50),
  sub1 NUMBER(3),
  sub2 NUMBER(3),
  sub3 NUMBER(3),
  sub4 NUMBER(3)
);

-- Sample Data
INSERT INTO student_marks VALUES (1, 'Rahul', 78, 82, 69, 75);
INSERT INTO student_marks VALUES (2, 'Sneha', 85, 88, 91, 89);
COMMIT;

-- PL/SQL Block for Total and Percentage
SET SERVEROUTPUT ON;

DECLARE
  v_roll student_marks.rollno%TYPE := 1; -- change roll number here
```

```
  v_name student_marks.name%TYPE;
  s1 student_marks.sub1%TYPE;
  s2 student_marks.sub2%TYPE;
  s3 student_marks.sub3%TYPE;
  s4 student_marks.sub4%TYPE;
  total NUMBER(4);
  percent NUMBER(5,2);
BEGIN
  SELECT name, sub1, sub2, sub3, sub4
  INTO v_name, s1, s2, s3, s4
  FROM student_marks
  WHERE rollno = v_roll;

  IF s1 < 0 OR s2 < 0 OR s3 < 0 OR s4 < 0 OR
     s1 > 100 OR s2 > 100 OR s3 > 100 OR s4 > 100 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Invalid Marks Entered');
  END IF;

  total := s1 + s2 + s3 + s4;
  percent := total / 4;

  DBMS_OUTPUT.PUT_LINE('Roll No : ' || v_roll);
  DBMS_OUTPUT.PUT_LINE('Name    : ' || v_name);
  DBMS_OUTPUT.PUT_LINE('Total   : ' || total);
  DBMS_OUTPUT.PUT_LINE('Percent : ' || percent || '%');

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No student found with given roll number.');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

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

Orders(cust_id, amount, status)

```
// Use your database
use ecommerce;

// Drop Orders collection if it exists
db.Orders.drop();

// Create & Insert Documents (Create)
db.Orders.insertMany([
  { cust_id: 101, amount: 2500, status: "Pending" },
  { cust_id: 102, amount: 4800, status: "Shipped" },
  { cust_id: 103, amount: 1500, status: "Delivered" }
]);

// Read: Find all orders
db.Orders.find().pretty();

// Read: Find specific order by customer ID
db.Orders.find({ cust_id: 102 });

// Update: Change status of an order
db.Orders.updateOne(
  { cust_id: 101 },
  { $set: { status: "Shipped" } }
);

// Delete: Remove an order
db.Orders.deleteOne({ cust_id: 103 });

// Verify all documents after operations
db.Orders.find().pretty();
```

18 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)

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.**

```
-- 1. Create tables with constraints

CREATE TABLE branch (
   branch_name VARCHAR(50) PRIMARY KEY,
   branch_city VARCHAR(50) NOT NULL,
   assets DECIMAL(15,2) NOT NULL CHECK (assets >= 0)
);

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

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),
   FOREIGN KEY (cust_name) REFERENCES customer(cust_name),
   FOREIGN KEY (acc_no) REFERENCES Account(Acc_no)
);

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

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

-- 2. Insert sample data
```

```
INSERT INTO branch VALUES
('Akurdi', 'Pune', 10000000),
('Hinjewadi', 'Pune', 15000000);

INSERT INTO Account VALUES
(101, 'Akurdi', 15000),
(102, 'Akurdi', 8000),
(103, 'Hinjewadi', 20000);

INSERT INTO customer VALUES
('Alice', 'Street 1', 'Pune'),
('Bob', 'Street 2', 'Mumbai'),
('Charlie', 'Street 3', 'Pune'),
('David', 'Street 4', 'Pune');

INSERT INTO Depositor VALUES
('Alice', 101),
('Bob', 102),
('Charlie', 103);

INSERT INTO Loan VALUES
(201, 'Akurdi', 25000),
(202, 'Hinjewadi', 30000);

INSERT INTO Borrower VALUES
('Alice', 201),
('David', 202);

-- 3. Queries

-- a) Find all customers who have an account or loan or both
SELECT DISTINCT cust_name
FROM (
   SELECT cust_name FROM Depositor
   UNION
   SELECT cust_name FROM Borrower
) AS all_customers;

-- b) Find all customers who have both account and loan
SELECT cust_name
FROM Depositor
WHERE cust_name IN (SELECT cust_name FROM Borrower);

-- c) Find all customers who have account but no loan
SELECT cust_name
FROM Depositor
WHERE cust_name NOT IN (SELECT cust_name FROM Borrower);

-- d) Find average account balance at Akurdi branch
SELECT AVG(balance) AS avg_balance
FROM Account
WHERE branch_name = 'Akurdi';
```

19 Implement 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.**

```
-- 1. Create tables (if not exists) with constraints

CREATE TABLE IF NOT EXISTS branch (
   branch_name VARCHAR(50) PRIMARY KEY,
   branch_city VARCHAR(50) NOT NULL,
   assets DECIMAL(15,2) NOT NULL CHECK (assets >= 0)
);

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

CREATE TABLE IF NOT EXISTS customer (
   cust_name VARCHAR(50) PRIMARY KEY,
   cust_street VARCHAR(100),
   cust_city VARCHAR(50)
);

CREATE TABLE IF NOT EXISTS Depositor (
   cust_name VARCHAR(50),
   acc_no INT,
   PRIMARY KEY (cust_name, acc_no),
   FOREIGN KEY (cust_name) REFERENCES customer(cust_name),
   FOREIGN KEY (acc_no) REFERENCES Account(Acc_no)
);
```

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

CREATE TABLE IF NOT EXISTS Borrower (
   cust_name VARCHAR(50),
   loan_no INT,
   PRIMARY KEY (cust_name, loan_no),
   FOREIGN KEY (cust_name) REFERENCES customer(cust_name),
   FOREIGN KEY (loan_no) REFERENCES Loan(loan_no)
);

-- 2. Insert sample data (if needed)

INSERT IGNORE INTO branch VALUES
('Nigdi', 'Pune', 8000000),
('Akurdi', 'Pune', 10000000);

INSERT IGNORE INTO Loan VALUES
(301, 'Nigdi', 1400),
(302, 'Akurdi', 1600),
(303, 'Nigdi', 1200),
(304, 'Akurdi', 1800);

-- 3. Calculate total loan amount given by bank

SELECT SUM(amount) AS total_loan_amount FROM Loan;

-- 4. Delete all loans with loan amount between 1300 and 1500

DELETE FROM Loan WHERE amount BETWEEN 1300 AND 1500;

-- 5. Delete all tuples at every branch located in Nigdi

-- First delete dependent data to avoid FK issues
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 = 'Nigdi';

DELETE FROM branch WHERE branch_name = 'Nigdi';
```

20 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 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 constraints

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

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

CREATE TABLE IF NOT EXISTS 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 IF NOT EXISTS 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)
);

-- 2. Insert sample data

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

INSERT IGNORE INTO Customers VALUES
('ABC', 'Pune'),
('XYZ', 'Mumbai'),
('DEF', 'Pune');

INSERT IGNORE INTO Deposit VALUES
(101, 'ABC', 'OBC', 5000.00, '2024-05-10'),
(102, 'XYZ', 'SBI', 7500.00, '2024-04-15'),
(103, 'DEF', 'OBC', 3000.00, '2024-03-20');

INSERT IGNORE INTO Borrow VALUES
(201, 'ABC', 'OBC', 20000.00),
(202, 'XYZ', 'SBI', 15000.00);

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

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

```
-- 4. Modify the size of attribute of amount in deposit (increase decimal places to 15 digits total and 2 decimals)

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 total_customers FROM Customers;
```

21 Create following tables:

6. **Deposit (actno,cname,bname,amount,adate)**
7. **Branch (bname,city)**
8. **Customers (cname, city)**
9. **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.**

```
-- 1. Create tables with primary and foreign keys

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

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

CREATE TABLE IF NOT EXISTS 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 IF NOT EXISTS 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)
);
```

```
-- 2. Insert sample data

INSERT IGNORE INTO Branch VALUES
('B1', 'Nagpur'),
('B2', 'Mumbai'),
('B3', 'Pune');

INSERT IGNORE INTO Customers VALUES
('Alice', 'Bombay'),
('Bob', 'Nagpur'),
('Charlie', 'Nagpur'),
('David', 'Pune'),
('Eve', 'Mumbai');

INSERT IGNORE INTO Deposit VALUES
(1, 'Alice', 'B1', 5000.00, '2024-01-15'),
(2, 'Bob', 'B1', 7000.00, '2024-02-10'),
(3, 'Charlie', 'B3', 6500.00, '2024-03-05'),
(4, 'David', 'B2', 8000.00, '2024-01-20');

INSERT IGNORE INTO Borrow VALUES
(101, 'Bob', 'B1', 15000.00),
(102, 'Charlie', 'B3', 12000.00),
(103, 'Eve', 'B2', 10000.00);

-- 3. Display customer name having living city Bombay and branch city Nagpur

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

-- 4. Display customer name having same living city as their branch city

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

-- 5. Display customer name who are borrowers as well as depositors and having living city Nagpur

SELECT DISTINCT c.cname
FROM Customers c
JOIN Deposit d ON c.cname = d.cname
JOIN Borrow b ON c.cname = b.cname
WHERE c.city = 'Nagpur';
```

22 Create the following tables.

4. **Deposit (actno,cname,bname,amount,adate)**
5. **Branch (bname,city)**
6. **Customers (cname, city)**
7. **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'**

```
-- 1. Create tables with constraints

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

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

CREATE TABLE IF NOT EXISTS 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 IF NOT EXISTS 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)
);

-- 2. Insert sample data

INSERT IGNORE INTO Branch VALUES
('B1', 'Pune'),
('B2', 'Mumbai'),
('B3', 'Nagpur');

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

INSERT IGNORE INTO Deposit VALUES
(101, 'ABC', 'B2', 2000.00, '2024-01-01'),
(102, 'Pramod', 'B3', 3000.00, '2024-02-01'),
(103, 'Kiran', 'B1', 1500.00, '2024-03-01'),
(104, 'Ravi', 'B3', 1200.00, '2024-04-01');

INSERT IGNORE INTO Borrow VALUES
(201, 'Sunil', 'B1', 5000.00),
(202, 'Pramod', 'B3', 7000.00),
```

```
(203, 'Ravi', 'B3', 8000.00),
(204, 'Kiran', 'B1', 6500.00);

-- 3. Display loan no and loan amount of borrowers having the same branch as that of Sunil

SELECT loanno, amount
FROM Borrow
WHERE bname = (
    SELECT bname FROM Borrow WHERE cname = 'Sunil'
);

-- 4. Display deposit and loan details of customers in the city where Pramod is living

SELECT d.actno, d.cname, d.bname, d.amount AS deposit_amount, d.adate
FROM Deposit d
JOIN Customers c ON d.cname = c.cname
WHERE c.city = (SELECT city FROM Customers WHERE cname = 'Pramod');

SELECT b.loanno, b.cname, b.bname, b.amount AS loan_amount
FROM Borrow b
JOIN Customers c ON b.cname = c.cname
WHERE c.city = (SELECT city FROM Customers WHERE cname = 'Pramod');

-- 5. Display borrower names having deposit amount > 1000 and living in the same city as Pramod

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

-- 6. Display branch and living city of ‘ABC’

SELECT d.bname, c.city
FROM Deposit d
```

```
JOIN Customers c ON d.cname = c.cname
WHERE d.cname = 'ABC';
```

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

Employee(emp_id, emp_name,emp_dept,salary)

```
// 1. Create & use database
use companyDB

// 2. Create Employee collection with sample documents
db.Employee.insertMany([
  { emp_id: 101, emp_name: "Alice", emp_dept: "HR", salary: 45000 },
  { emp_id: 102, emp_name: "Bob", emp_dept: "IT", salary: 60000 },
  { emp_id: 103, emp_name: "Charlie", emp_dept: "IT", salary: 70000 },
  { emp_id: 104, emp_name: "David", emp_dept: "HR", salary: 48000 },
  { emp_id: 105, emp_name: "Eva", emp_dept: "Finance", salary: 52000 },
  { emp_id: 106, emp_name: "Frank", emp_dept: "Finance", salary: 57000 }
])

// 3. AGGREGATION OPERATIONS

// a. Total salary paid to all employees
db.Employee.aggregate([
  { $group: { _id: null, total_salary: { $sum: "$salary" } } }
])

// b. Average salary in each department
db.Employee.aggregate([
  { $group: { _id: "$emp_dept", avg_salary: { $avg: "$salary" } } }
])

// c. Minimum and maximum salary in each department
```

```
db.Employee.aggregate([
  { $group: {
     _id: "$emp_dept",
     min_salary: { $min: "$salary" },
     max_salary: { $max: "$salary" }
  } }
])

// d. Count of employees in each department
db.Employee.aggregate([
  { $group: { _id: "$emp_dept", count: { $sum: 1 } } }
])

// e. Sort employees by salary in descending order
db.Employee.aggregate([
  { $sort: { salary: -1 } }
])

// 4. INDEXING

// a. Create index on emp_id (single field)
db.Employee.createIndex({ emp_id: 1 })

// b. Create compound index on emp_dept and salary
db.Employee.createIndex({ emp_dept: 1, salary: -1 })

// c. View all indexes on Employee collection
db.Employee.getIndexes()

// d. Find employees in IT department using index
db.Employee.find({ emp_dept: "IT" }).explain("executionStats")
```

24 Create the following tables.

5. **Deposit (actno,cname,bname,amount,adate)**
6. **Branch (bname,city)**
7. **Customers (cname, city)**
8. **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.**

```
-- 1. Create Tables

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) CHECK (amount > 0),
   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) CHECK (amount > 0),
   FOREIGN KEY (cname) REFERENCES Customers(cname),
   FOREIGN KEY (bname) REFERENCES Branch(bname)
);

-- 2. Insert Sample Data
```

```
INSERT INTO Branch VALUES
('KAROLBAGH', 'DELHI'),
('SHIVAJINAGAR', 'PUNE'),
('MGROAD', 'BANGALORE');

INSERT INTO Customers VALUES
('Anil', 'PUNE'),
('Sunita', 'DELHI'),
('Raj', 'PUNE'),
('Priya', 'BANGALORE');

INSERT INTO Deposit VALUES
(101, 'Anil', 'SHIVAJINAGAR', 1500, '1999-02-15'),
(102, 'Raj', 'SHIVAJINAGAR', 2000, '1998-03-01'),
(103, 'Sunita', 'KAROLBAGH', 2500, '1997-11-20'),
(104, 'Priya', 'MGROAD', 1800, '2000-06-25');

INSERT INTO Borrow VALUES
(201, 'Anil', 'KAROLBAGH', 3000),
(202, 'Sunita', 'KAROLBAGH', 4500),
(203, 'Raj', 'SHIVAJINAGAR', 2800);

-- 3. Queries

-- Q1: Display amount for depositors living in the city where Anil is living
SELECT D.cname, D.amount
FROM Deposit D
JOIN Customers C1 ON D.cname = C1.cname
JOIN Customers C2 ON C2.cname = 'Anil'
WHERE C1.city = C2.city;

-- Q2: Display total loan and maximum loan taken from KAROLBAGH branch
SELECT SUM(amount) AS total_loan, MAX(amount) AS max_loan
FROM Borrow
WHERE bname = 'KAROLBAGH';

-- Q3: Display total deposit of customers having account date later than ‘1-jan-98’
SELECT SUM(amount) AS total_deposit
FROM Deposit
WHERE adate > '1998-01-01';

-- Q4: Display maximum deposit of customers living in PUNE
SELECT MAX(D.amount) AS max_deposit
FROM Deposit D
JOIN Customers C ON D.cname = C.cname
WHERE C.city = 'PUNE';
```

25 Design and Implement any 5 query using MongoDB

1.Create a collection called ‘games’.

2.Add 5 games to the database. Give each document the following properties: name, gametype, score (out of 100), achievements

3.Write a query that returns all the games

4.Write a query that returns the 3 highest scored games.

5.Write a query that returns all the games that have both the ‘Game Maser’ and the ‘Speed Demon’ achievements.

```
// Create the collection and insert data
db.games.insertMany([
  { name: "Apex Legends", gametype: "Shooter", score: 88, achievements:
["Speed Demon", "Sniper"] },
  { name: "FIFA 23", gametype: "Sports", score: 92, achievements: ["Team
Player", "Goal Master"] },
  { name: "Valorant", gametype: "Shooter", score: 95, achievements: ["Game
Master", "Speed Demon"] },
  { name: "Minecraft", gametype: "Adventure", score: 85, achievements:
["Builder", "Game Master"] },
  { name: "Asphalt 9", gametype: "Racing", score: 91, achievements: ["Speed
Demon", "Drift King"] }
]);

// a) Return all games
db.games.find();

// b) Return top 3 highest scored games
db.games.find().sort({ score: -1 }).limit(3);

// c) Return games with both “Game Master” and “Speed Demon”
db.games.find({ achievements: { $all: ["Game Master", "Speed Demon"] } });
```

26 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 (
  emp_no INT PRIMARY KEY,
  emp_name VARCHAR(50),
  basic DECIMAL(10,2),
  HRA DECIMAL(10,2),
  DA DECIMAL(10,2),
  Total_deduction DECIMAL(10,2),
  net_salary DECIMAL(10,2),
  gross_salary DECIMAL(10,2)
);

DELIMITER $$

CREATE PROCEDURE CalculateTax(IN empNo INT)
BEGIN
  DECLARE v_gross_salary DECIMAL(10,2);
  DECLARE v_emp_name VARCHAR(50);
  DECLARE v_tax DECIMAL(10,2);

  SELECT emp_name, (basic + HRA + DA) INTO v_emp_name, v_gross_salary
  FROM Employee_salary
  WHERE emp_no = empNo;

  SET v_tax = v_gross_salary * 0.10; -- 10% tax example

  SELECT CONCAT('Employee: ', v_emp_name, ', Tax: ', v_tax) AS Result;
END$$

DELIMITER ;

-- To call the procedure:
CALL CalculateTax(101);
```

27 Create PL/SQL code block: Write a PL/SQL block of code for the following schema:

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

Fine(Roll_no,Date,Amt)

Solve following queries:

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. **If condition of fine is true, then details will be stored into fine table.**

Use of Control structure and Exception handling is mandatory.

```
CREATE TABLE Borrower (
  Rollin INT,
  Name VARCHAR(50),
  DateofIssue DATE,
  NameofBook VARCHAR(100),
  Status CHAR(1),
  PRIMARY KEY (Rollin, NameofBook)
);

CREATE TABLE Fine (
  Roll_no INT,
  Date DATE,
  Amt DECIMAL(10,2)
);
DELIMITER $$

CREATE PROCEDURE CalculateFine(IN p_roll_no INT, IN p_book_name VARCHAR(100))
BEGIN
  DECLARE v_date_of_issue DATE;
  DECLARE v_status CHAR(1);
  DECLARE v_days INT;
  DECLARE v_fine_amt INT DEFAULT 0;

  SELECT DateofIssue, Status INTO v_date_of_issue, v_status
  FROM Borrower
  WHERE Rollin = p_roll_no AND NameofBook = p_book_name;

  SET v_days = DATEDIFF(CURDATE(), v_date_of_issue);

  IF v_days < 15 THEN
    SET v_fine_amt = 0;
  ELSEIF v_days BETWEEN 15 AND 30 THEN
    SET v_fine_amt = v_days * 5;
  ELSE
    SET v_fine_amt = v_days * 50;
  END IF;
```

```
  IF v_status = 'I' THEN
    UPDATE Borrower SET Status = 'R' WHERE Rollin = p_roll_no AND NameofBook = p_book_name;
  END IF;

  IF v_fine_amt > 0 THEN
    INSERT INTO Fine(Roll_no, Date, Amt) VALUES (p_roll_no, CURDATE(), v_fine_amt);
  END IF;

  SELECT CONCAT('Fine amount: ', v_fine_amt) AS FineMessage;
END$$

DELIMITER ;

-- Call example:
CALL CalculateFine(123, 'Data Structures');
```

28 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 (
  roll_no INT PRIMARY KEY,
  name VARCHAR(50)
);

CREATE TABLE O_RollCall (
  roll_no INT PRIMARY KEY,
  name VARCHAR(50)
);
DELIMITER $$

CREATE PROCEDURE MergeNRollCall()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE v_roll_no INT;
  DECLARE v_name VARCHAR(100);

  DECLARE cur CURSOR FOR SELECT roll_no, name FROM N_RollCall;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
```

```
  OPEN cur;
  read_loop: LOOP
    FETCH cur INTO v_roll_no, v_name;
    IF done THEN
      LEAVE read_loop;
    END IF;

    IF NOT EXISTS (SELECT 1 FROM O_RollCall WHERE roll_no = v_roll_no) THEN
      INSERT INTO O_RollCall(roll_no, name) VALUES (v_roll_no, v_name);
    END IF;
  END LOOP;

  CLOSE cur;
END$$

DELIMITER ;

-- Call:
CALL MergeNRollCall();
```

29 Writ 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)

```
CREATE TABLE Student (
  ROLL_NO INT PRIMARY KEY,
  COURSE VARCHAR(50),
  COURSE_COD VARCHAR(20),
  SEM INT,
  TOTAL_MARKS INT,
  PERCENTAGE DECIMAL(5,2)
);
DELIMITER $$

CREATE PROCEDURE CountStudentsByRange(IN p_course VARCHAR(50))
BEGIN
  SELECT
```

```
    SUM(CASE WHEN percentage BETWEEN 70 AND 100 THEN 1 ELSE 0 END) AS Range_70_100,
    SUM(CASE WHEN percentage BETWEEN 60 AND 69 THEN 1 ELSE 0 END) AS Range_60_69,
    SUM(CASE WHEN percentage BETWEEN 50 AND 59 THEN 1 ELSE 0 END) AS Range_50_59,
    SUM(CASE WHEN percentage < 50 THEN 1 ELSE 0 END) AS Below_49
  FROM Student
  WHERE course = p_course;
END$$

DELIMITER ;

-- Call example:
CALL CountStudentsByRange('Computer Science');
```

30 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 class, 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 (
  roll_no INT PRIMARY KEY,
  name VARCHAR(50),
  total_marks INT
);

CREATE TABLE Result (
  Roll INT PRIMARY KEY,
  Name VARCHAR(50),
  Class VARCHAR(30)
);
DELIMITER $$
```

```
CREATE PROCEDURE proc_Grade()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE v_roll_no INT;
  DECLARE v_name VARCHAR(50);
  DECLARE v_total_marks INT;
  DECLARE v_class VARCHAR(30);

  DECLARE cur CURSOR FOR SELECT roll_no, name, total_marks FROM Stud_Marks;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN cur;
  read_loop: LOOP
    FETCH cur INTO v_roll_no, v_name, v_total_marks;
    IF done THEN
      LEAVE read_loop;
    END IF;

    IF v_total_marks BETWEEN 990 AND 1500 THEN
      SET v_class = 'Distinction';
    ELSEIF v_total_marks BETWEEN 900 AND 989 THEN
      SET v_class = 'First Class';
    ELSEIF v_total_marks BETWEEN 825 AND 899 THEN
      SET v_class = 'Higher Second Class';
    ELSE
      SET v_class = 'Others';
    END IF;

    INSERT INTO Result(Roll, Name, Class) VALUES (v_roll_no, v_name, v_class);
  END LOOP;

  CLOSE cur;
END$$

DELIMITER ;

-- Call:
CALL proc_Grade();
```

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

1. Find the total population in pune.
2. returns all city with total population greater than 10 million
3. returns the average populations for each city.
4. returns the minimum and maximum cities by population for each city.

```
// Insert sample documents
db.Citydetails.insertMany([
  {
    _id: 1,
    name: "Pune",
    area: 500,
    population: { total: 6000000, Adults: 4000000, seniorcitizens: 500000, sexratio: 950 },
    geography: { avgtemp: 25, avgrainfall: 70, longitude: 73.8567, latitude: 18.5204 }
  },
  {
    _id: 2,
    name: "Mumbai",
    area: 600,
      population: { total: 20000000, Adults: 13000000, seniorcitizens: 1500000, sexratio:
920 },
    geography: { avgtemp: 27, avgrainfall: 80, longitude: 72.8777, latitude: 19.0760 }
  }
]);

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

// Query 2: Cities with population > 10 million
db.Citydetails.find({ "population.total": { $gt: 10000000 } });

// Query 3: Average population per city (Adults + seniorcitizens)/2 as an example
db.Citydetails.aggregate([
  { $project: {
    name: 1,
    avg_population: { $avg: ["$population.Adults", "$population.seniorcitizens"] }
  }}
```

```
]);

// Query 4: Min and max cities by population
db.Citydetails.aggregate([
  { $group: {
    _id: null,
    maxPop: { $max: "$population.total" },
    minPop: { $min: "$population.total" }
  }}
]);
```

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

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

```
use CityDB;

// Insert sample data
db.Citydetails.insertMany([
  {
    _id: 1,
    name: "Mumbai",
    area: "Mumbai Metro",
      population: { total: 20411000, Adults: 15000000, seniorcitizens: 2000000, sexratio: 940.00 },
    geography: { avgtemp: 27.5, avgrainfall: 80.0, longitude: 72.8777, latitude: 19.0760 }
  },
  {
    _id: 2,
    name: "Pune",
    area: "Pune Metro",
    population: { total: 6500000, Adults: 4800000, seniorcitizens: 700000, sexratio: 930.00 },
```

```
    geography: { avgtemp: 24.0, avgrainfall: 65.5, longitude: 73.8567, latitude: 18.5204 }
  },
  {
    _id: 3,
    name: "Nagpur",
    area: "Vidarbha",
    population: { total: 2400000, Adults: 1800000, seniorcitizens: 250000, sexratio: 920.00
},
    geography: { avgtemp: 26.0, avgrainfall: 55.0, longitude: 79.0882, latitude: 21.1458 }
  },
  {
    _id: 4,
    name: "Nashik",
    area: "North Maharashtra",
    population: { total: 1800000, Adults: 1400000, seniorcitizens: 180000, sexratio: 935.00
},
    geography: { avgtemp: 25.0, avgrainfall: 62.0, longitude: 73.7898, latitude: 20.0110 }
  }
]);

// 1. Find area wise total population and sort ascending
print("Area wise total population sorted increasing:");
db.Citydetails.aggregate([
  { $group: { _id: "$area", totalPopulation: { $sum: "$population.total" } } },
  { $sort: { totalPopulation: 1 } }
]).forEach(printjson);

// 2. Retrieve name and area where avg rainfall > 60
print("Cities with avg rainfall > 60:");
db.Citydetails.find(
  { "geography.avgrainfall": { $gt: 60 } },
  { name: 1, area: 1, _id: 0 }
).forEach(printjson);

// 3. Create indexes on name and area
db.Citydetails.createIndex({ name: 1 });
db.Citydetails.createIndex({ area: 1 });

// Find max population in Mumbai
print("Max population in Mumbai:");
```

```
db.Citydetails.aggregate([
  { $match: { name: "Mumbai" } },
  { $group: { _id: "$name", maxPopulation: { $max: "$population.total" } } }
]).forEach(printjson);

// 4. Create index on name (already created above)
```