```
-- ========================================================
-- ✅ ASSIGNMENT  1: MYSQL DATABASE CREATION AND QUERIES
-- ========================================================

CREATE DATABASE LibraryDB;
USE LibraryDB;

-- ========================================================
-- 1️⃣ TABLE CREATION (With Referential Integrity & ON DELETE CASCADE)
-- ========================================================

CREATE TABLE author (
    author_no INT PRIMARY KEY AUTO_INCREMENT,
    author_name VARCHAR(100),
    country VARCHAR(50)
);

CREATE TABLE publisher (
    publisher_no INT PRIMARY KEY AUTO_INCREMENT,
    publisher_name VARCHAR(100),
    publisher_addr VARCHAR(150),
    year INT
);

CREATE TABLE book (
    ISBN INT PRIMARY KEY,
    title VARCHAR(150),
    unit_price DECIMAL(10,2),
    author_no INT,
    publisher_no INT,
    pub_year INT,
```

```
    FOREIGN KEY (author_no) REFERENCES author(author_no) ON DELETE CASCADE,
    FOREIGN KEY (publisher_no) REFERENCES publisher(publisher_no) ON DELETE CASCADE
);

CREATE TABLE customer (
    cust_no INT PRIMARY KEY AUTO_INCREMENT,
    cust_fname VARCHAR(50),
    cust_lname VARCHAR(50),
    cust_company VARCHAR(100),
    cust_addr VARCHAR(150),
    city VARCHAR(50),
    cust_phone VARCHAR(15)
);

CREATE TABLE orders (
    order_no INT PRIMARY KEY AUTO_INCREMENT,
    cust_no INT,
    ISBN INT,
    qty INT,
    odate DATE,
    FOREIGN KEY (cust_no) REFERENCES customer(cust_no) ON DELETE CASCADE,
    FOREIGN KEY (ISBN) REFERENCES book(ISBN) ON DELETE CASCADE
);

-- ========================================================
-- 2️⃣ INSERT SAMPLE RECORDS
-- ========================================================

INSERT INTO author (author_name, country) VALUES
('Chetan Bhagat', 'India'),
('JK Rowling', 'UK'),
```

('John Green', 'USA'),

('Ruskin Bond', 'India'),

('Mark Manson', 'Australia');

INSERT INTO publisher (publisher_name, publisher_addr, year) VALUES

('Penguin India', 'Delhi', 2015),

('Bloomsbury', 'London', 2016),

('HarperCollins', 'New York', 2014),

('Rupa Publications', 'Mumbai', 2015),

('Macmillan', 'Sydney', 2016);

INSERT INTO book (ISBN, title, unit_price, author_no, publisher_no, pub_year) VALUES

(1001, 'Half Girlfriend', 250, 1, 4, 2015),

(1002, 'Harry Potter', 500, 2, 2, 2016),

(1003, 'Paper Towns', 350, 3, 3, 2014),

(1004, 'The Monk Who Sold His Ferrari', 400, 4, 1, 2006),

(1005, 'The Subtle Art of Not Giving a F*ck', 380, 5, 5, 2015),

(1006, '2 States', 270, 1, 4, 2009),

(1007, 'Fantastic Beasts', 420, 2, 2, 2016),

(1008, 'Looking for Alaska', 310, 3, 3, 2004),

(1009, 'Blue Umbrella', 290, 4, 1, 2000),

(1010, 'Everything is F*cked', 390, 5, 5, 2019);

INSERT INTO customer (cust_fname, cust_lname, cust_company, cust_addr, city, cust_phone) VALUES

('Pranav', 'Shewale', 'TechCorp', 'Baner', 'Pune', '9876543210'),

('Harshad', 'Kavade', 'Innova', 'Thane', 'Mumbai', '9823456789'),

('Sagar', 'Mane', 'InfoTech', 'Karve Nagar', 'Pune', '9988776655'),

('Atharva', 'Patil', 'CodeX', 'Dadar', 'Mumbai', '8899776655'),

('Kunjal', 'Patil', 'Cloudify', 'Hadapsar', 'Pune', '7788990011'),

('Paras', 'Ghumanna', 'SoftHub', 'Andheri', 'Mumbai', '9876501234'),

('Vrushabh', 'Darekar', 'NetSolve', 'Kothrud', 'Pune', '9001122334'),

```
('Aditya', 'Magdum', 'Techie', 'Vile Parle', 'Mumbai', '9009988776'),
('Chaitanya', 'Pawar', 'BrightTech', 'FC Road', 'Pune', '7888997766'),
('Ayush', 'Mahadik', 'NextGen', 'Panvel', 'Mumbai', '7002233445');

INSERT INTO orders (cust_no, ISBN, qty, odate) VALUES
(1, 1001, 2, '2024-06-10'),
(2, 1002, 1, '2024-07-05'),
(3, 1003, 3, '2024-07-20'),
(4, 1005, 1, '2024-08-02'),
(5, 1004, 2, '2024-08-12'),
(6, 1008, 1, '2024-08-14'),
(7, 1009, 4, '2024-09-01'),
(8, 1010, 2, '2024-09-10'),
(9, 1006, 1, '2024-09-25'),
(10, 1007, 2, '2024-10-01');

-- ============================================================
-- 3️⃣ VIEW, INDEX, SEQUENCE, SYNONYM
-- ============================================================

-- VIEW: Show complete order summary
CREATE VIEW vw_order_summary AS
SELECT
   o.order_no, c.cust_fname, c.city, b.title, b.unit_price, o.qty, o.odate
FROM
   orders o
   JOIN customer c ON o.cust_no = c.cust_no
   JOIN book b ON o.ISBN = b.ISBN;

-- INDEX: For faster customer city lookups
CREATE INDEX idx_city ON customer(city);
```

```
-- SEQUENCE: Demonstrated via auto_increment substitute
CREATE TABLE seq_order (id INT AUTO_INCREMENT PRIMARY KEY);
INSERT INTO seq_order VALUES (NULL);

-- SYNONYM (MySQL workaround using view alias)
CREATE VIEW book_syn AS SELECT * FROM book;

-- ============================================================
-- 4️⃣ QUERIES
-- ============================================================

-- (2) Display all customers from Pune & Mumbai with first name starting with 'P' or 'H'
SELECT *
FROM customer
WHERE city IN ('Pune', 'Mumbai')
  AND (cust_fname LIKE 'P%' OR cust_fname LIKE 'H%');

-- (3) Number of different customer cities
SELECT COUNT(DISTINCT city) AS number_of_cities
FROM customer;

-- (4) 5% increase in price of books published in 2015
UPDATE book
SET unit_price = unit_price * 1.05
WHERE pub_year = 2015;

-- (5) Delete customer details living in Pune
DELETE FROM customer WHERE city = 'Pune';

-- (6) Names of authors living in India or Australia
```

```
SELECT author_name
FROM author
WHERE country IN ('India', 'Australia');

-- (7) Publishers established between 2015 and 2016
SELECT publisher_name
FROM publisher
WHERE year BETWEEN 2015 AND 2016;

-- (8) Book with maximum price and titles with price between 300 and 400
SELECT title, unit_price
FROM book
WHERE unit_price = (SELECT MAX(unit_price) FROM book);

SELECT title, unit_price
FROM book
WHERE unit_price BETWEEN 300 AND 400;

-- (9) Titles with price and published year in decreasing order of year
SELECT title, unit_price, pub_year
FROM book
ORDER BY pub_year DESC;

-- (10) Title, author_no, publisher_no of books published in 2000, 2004, 2006
SELECT title, author_no, publisher_no
FROM book
WHERE pub_year IN (2000, 2004, 2006);

-- ========================================================
-- ✅ END OF ASSIGNMENT
-- ========================================================
```

```
-- ============================================================
-- ✅ ASSIGNMENT 2: PROFESSORS & DEPARTMENTS DATABASE
-- ============================================================

CREATE DATABASE CollegeDB;
USE CollegeDB;

-- ============================================================
-- 1️⃣ TABLE CREATION (With Referential Integrity & ON DELETE CASCADE)
-- ============================================================

CREATE TABLE Departments (
    dept_id INT PRIMARY KEY AUTO_INCREMENT,
    dept_name VARCHAR(100)
);

CREATE TABLE Professors (
    prof_id INT PRIMARY KEY AUTO_INCREMENT,
    prof_fname VARCHAR(50),
    prof_lname VARCHAR(50),
    dept_id INT,
    designation VARCHAR(50),
    salary DECIMAL(10,2),
    doj DATE,
    email VARCHAR(100),
    phone VARCHAR(15),
    city VARCHAR(50),
    FOREIGN KEY (dept_id) REFERENCES Departments(dept_id) ON DELETE CASCADE
);
```

```
CREATE TABLE Works (
    prof_id INT,
    dept_id INT,
    duration VARCHAR(50),
    PRIMARY KEY (prof_id, dept_id),
    FOREIGN KEY (prof_id) REFERENCES Professors(prof_id) ON DELETE CASCADE,
    FOREIGN KEY (dept_id) REFERENCES Departments(dept_id) ON DELETE CASCADE
);

CREATE TABLE Shift (
    prof_id INT PRIMARY KEY,
    shift VARCHAR(20),
    working_hours INT,
    FOREIGN KEY (prof_id) REFERENCES Professors(prof_id) ON DELETE CASCADE
);

-- ============================================================
-- 2️⃣ INSERT SAMPLE RECORDS
-- ============================================================

-- DEPARTMENTS
INSERT INTO Departments (dept_name) VALUES
('Computer Engineering'),
('Information Technology'),
('Electronics'),
('Mechanical'),
('Civil');

-- PROFESSORS
INSERT INTO Professors (prof_fname, prof_lname, dept_id, designation, salary, doj, email, phone,
city) VALUES
```

('Amit', 'Patil', 1, 'Assistant Professor', 25000, '2015-01-01', 'amit.patil@college.com', '9876543210', 'Pune'),

('Dinesh', 'Sharma', 2, 'Associate Professor', 45000, '2014-05-12', 'dinesh.sharma@college.com', '9876501234', 'Mumbai'),

('Rajesh', 'Verma', 3, 'Professor', 60000, '2016-07-01', 'rajesh.verma@college.com', '9823456789', 'Delhi'),

('Anita', 'Joshi', 4, 'Assistant Professor', 18000, '2015-12-15', 'anita.joshi@college.com', '9988776655', 'Pune'),

('Deepak', 'Mehta', 5, 'Associate Professor', 32000, '2015-06-01', 'deepak.mehta@college.com', '8899776655', 'Mumbai'),

('Sneha', 'Kulkarni', 1, 'Professor', 70000, '2017-03-10', 'sneha.kulkarni@college.com', '7788990011', 'Nagpur'),

('Prakash', 'Naik', 2, 'Assistant Professor', 15000, '2014-10-05', 'prakash.naik@college.com', '9009988776', 'Pune'),

('Devika', 'Rao', 3, 'Associate Professor', 40000, '2016-01-10', 'devika.rao@college.com', '7888997766', 'Mumbai'),

('Anand', 'Deshmukh', 4, 'Professor', 50000, '2015-01-01', 'anand.deshmukh@college.com', '9001122334', 'Pune'),

('Dipali', 'Gore', 5, 'Assistant Professor', 20000, '2016-02-12', 'dipali.gore@college.com', '7002233445', 'Mumbai');

-- WORKS TABLE

INSERT INTO Works (prof_id, dept_id, duration) VALUES

(1, 1, '5 years'),

(2, 2, '8 years'),

(3, 3, '6 years'),

(4, 4, '4 years'),

(5, 5, '7 years'),

(6, 1, '9 years'),

(7, 2, '3 years'),

(8, 3, '6 years'),

(9, 4, '10 years'),

(10, 5, '2 years');

```
-- SHIFT TABLE
INSERT INTO Shift (prof_id, shift, working_hours) VALUES
(1, 'Morning', 6),
(2, 'Evening', 8),
(3, 'Morning', 7),
(4, 'Evening', 6),
(5, 'Morning', 8),
(6, 'Morning', 6),
(7, 'Evening', 8),
(8, 'Morning', 7),
(9, 'Evening', 6),
(10, 'Morning', 7);

-- ============================================================
-- 3️⃣ CREATE VIEW, INDEX, SEQUENCE, SYNONYM
-- ============================================================

-- VIEW: Professor details with department name
CREATE VIEW vw_prof_details AS
SELECT
    p.prof_id, p.prof_fname, p.prof_lname, d.dept_name,
    p.designation, p.salary, p.city, p.doj
FROM
    Professors p
    JOIN Departments d ON p.dept_id = d.dept_id;

-- INDEX on city for faster search
CREATE INDEX idx_city_prof ON Professors(city);

-- SEQUENCE simulation (MySQL doesn't support real sequence, using table)
CREATE TABLE seq_prof (id INT AUTO_INCREMENT PRIMARY KEY);
```

```
INSERT INTO seq_prof VALUES (NULL);

-- SYNONYM (using VIEW alias)
CREATE VIEW prof_syn AS SELECT * FROM Professors;

-- ============================================================
-- 4️⃣ QUERIES
-- ============================================================

-- (1) Already inserted above ✅

-- (2) Professors from Pune & Mumbai with name starting with 'A' or 'D'
SELECT *
FROM Professors
WHERE city IN ('Pune', 'Mumbai')
  AND (prof_fname LIKE 'A%' OR prof_fname LIKE 'D%');

-- (3) Number of different professor cities
SELECT COUNT(DISTINCT city) AS number_of_cities
FROM Professors;

-- (4) 5% salary increase for professors who joined on 1-1-2015
UPDATE Professors
SET salary = salary * 1.05
WHERE doj = '2015-01-01';

-- (5) Delete professors living in Pune
DELETE FROM Professors WHERE city = 'Pune';

-- (6) Names of professors belonging to Pune or Mumbai
SELECT prof_fname, prof_lname
```

```
FROM Professors
WHERE city IN ('Pune', 'Mumbai');

-- (7) Professors who joined between 1-1-2015 and 1-1-2016
SELECT prof_fname, prof_lname, doj
FROM Professors
WHERE doj BETWEEN '2015-01-01' AND '2016-01-01';

-- (8) Professor having maximum salary & professors with salary between 10,000 and 20,000
SELECT prof_fname, prof_lname, salary
FROM Professors
WHERE salary = (SELECT MAX(salary) FROM Professors);

SELECT prof_fname, prof_lname, salary
FROM Professors
WHERE salary BETWEEN 10000 AND 20000;

-- (9) Professors name, salary, and doj in decreasing order of salary
SELECT prof_fname, prof_lname, salary, doj
FROM Professors
ORDER BY salary DESC;

-- (10) Professors name, doj, dept_id with salary 30000, 40000, or 50000
SELECT prof_fname, prof_lname, doj, dept_id
FROM Professors
WHERE salary IN (30000, 40000, 50000);

-- ========================================================
-- ✅ END OF ASSIGNMENT 2
-- ========================================================
```

Assign 3

```
-- Create Database
CREATE DATABASE BookDB;
USE BookDB;

-- ==========================
-- 1️⃣ CREATE TABLES
-- ==========================

CREATE TABLE Author (
    author_no INT PRIMARY KEY,
    author_name VARCHAR(50),
    country VARCHAR(30)
);

CREATE TABLE Publisher (
    pub_no INT PRIMARY KEY,
    pub_name VARCHAR(50),
    pub_city VARCHAR(30)
);

CREATE TABLE Book (
    ISBN INT PRIMARY KEY,
    book_title VARCHAR(50),
    author_no INT,
    pub_no INT,
    publisher_year YEAR,
    FOREIGN KEY (author_no) REFERENCES Author(author_no),
    FOREIGN KEY (pub_no) REFERENCES Publisher(pub_no)
);
```

```
CREATE TABLE Customer (
    cust_id INT PRIMARY KEY,
    cust_fname VARCHAR(50),
    cust_lname VARCHAR(50),
    company_name VARCHAR(50),
    city VARCHAR(30)
);

CREATE TABLE Orders (
    order_no INT PRIMARY KEY,
    cust_id INT,
    ISBN INT,
    order_date DATE,
    FOREIGN KEY (cust_id) REFERENCES Customer(cust_id),
    FOREIGN KEY (ISBN) REFERENCES Book(ISBN)
);

-- ==========================
-- 2️⃣ INSERT SAMPLE DATA
-- ==========================

INSERT INTO Author VALUES
(1, 'Robert Sheldon', 'USA'),
(2, 'James Smith', 'UK'),
(3, 'Priya Sharma', 'India');

INSERT INTO Publisher VALUES
(1, 'Pearson', 'London'),
(2, 'McGraw Hill', 'New York'),
(3, 'TechPress', 'Delhi');
```

```
INSERT INTO Book VALUES
(1111, 'Learn SQL', 1, 1, 2014),
(1234, 'Mastering MySQL', 2, 2, 2015),
(2222, 'Database Concepts', 3, 3, 2016);

INSERT INTO Customer VALUES
(101, 'Sagar', 'Mane', 'ABC Pvt Ltd', 'Pune'),
(102, 'Kunal', 'Patil', 'TechSoft', 'Mumbai'),
(103, 'Harsh', 'Kavade', 'DataCorp', 'Delhi');

INSERT INTO Orders VALUES
(501, 101, 1111, '2015-05-10'),
(502, 102, 1234, '2015-08-14');
-- Note: Customer 103 has not placed any order

-- ==========================
-- 3️⃣ QUERIES
-- ==========================

-- 1. Find Customer details and order details using NATURAL JOIN
SELECT *
FROM Customer NATURAL JOIN Orders;

-- 2. Find the book_title, author_name, country
SELECT b.book_title, a.author_name, a.country
FROM Book b
JOIN Author a ON b.author_no = a.author_no;

-- 3. Find the customer ID, name and order_no of customers who have never placed an order
SELECT c.cust_id, CONCAT(c.cust_fname, ' ', c.cust_lname) AS name, o.order_no
```

```
FROM Customer c
LEFT JOIN Orders o ON c.cust_id = o.cust_id
WHERE o.order_no IS NULL;

-- 4. Find the Title, ISBN, order_no of the books for which order is not placed
SELECT b.book_title AS Title, b.ISBN, o.order_no
FROM Book b
LEFT JOIN Orders o ON b.ISBN = o.ISBN
WHERE o.order_no IS NULL;

-- 5. Display cust_fname, title, author_no, publisher_year where ISBN = 1234
SELECT c.cust_fname, b.book_title AS title, b.author_no, b.publisher_year
FROM Customer c
JOIN Orders o ON c.cust_id = o.cust_id
JOIN Book b ON o.ISBN = b.ISBN
WHERE b.ISBN = 1234;

-- 6. Display the total number of books and customer name
SELECT c.cust_fname, COUNT(o.ISBN) AS Total_Books
FROM Customer c
LEFT JOIN Orders o ON c.cust_id = o.cust_id
GROUP BY c.cust_fname;

-- 7. List the cust_id, order_no and ISBN with books having title 'mysql'
SELECT c.cust_id, o.order_no, o.ISBN
FROM Customer c
JOIN Orders o ON c.cust_id = o.cust_id
JOIN Book b ON o.ISBN = b.ISBN
WHERE LOWER(b.book_title) LIKE '%mysql%';

-- 8. Find the names of all the companies that ordered books in the year 2015
```

```
SELECT DISTINCT c.company_name
FROM Customer c
JOIN Orders o ON c.cust_id = o.cust_id
WHERE YEAR(o.order_date) = 2015;

-- 9. Create view showing the author and book details
CREATE VIEW AuthorBookView AS
SELECT a.author_no, a.author_name, a.country, b.ISBN, b.book_title, b.publisher_year
FROM Author a
JOIN Book b ON a.author_no = b.author_no;

-- View the created view
SELECT * FROM AuthorBookView;

-- 10. Perform Manipulation on simple view - Insert, Update, Delete, Drop
-- Insert into view (affects base tables if allowed)
INSERT INTO AuthorBookView VALUES (4, 'Neha Desai', 'India', 3333, 'Intro to DBMS', 2018);

-- Update record in the view
UPDATE AuthorBookView
SET author_name = 'N. Desai'
WHERE author_no = 4;

-- Delete record from the view
DELETE FROM AuthorBookView WHERE author_no = 4;

-- Drop the view
DROP VIEW AuthorBookView;
```

Assign 4

```
-- ========================================
-- 1️⃣ CREATE DATABASE AND USE IT
-- ========================================
CREATE DATABASE LoanDB;
USE LoanDB;

-- ========================================
-- 2️⃣ CREATE TABLES
-- ========================================

CREATE TABLE Customer (
    Cust_name VARCHAR(50),
    AccNo INT PRIMARY KEY,
    Balance DECIMAL(12,2),
    City VARCHAR(30)
);

CREATE TABLE Loan (
    Loan_no INT PRIMARY KEY,
    Branch_name VARCHAR(50),
    Amount DECIMAL(12,2)
);

CREATE TABLE Borrower (
    Cust_name VARCHAR(50),
    Loan_no INT,
    FOREIGN KEY (Cust_name) REFERENCES Customer(Cust_name) ON DELETE CASCADE,
    FOREIGN KEY (Loan_no) REFERENCES Loan(Loan_no) ON DELETE CASCADE
);
```

```
CREATE TABLE Borrower_Category (
    Cust_name VARCHAR(50),
    AccNo INT,
    Loan_no INT,
    Branch_name VARCHAR(50),
    Amount DECIMAL(12,2),
    Category VARCHAR(20)
);

-- ========================================
-- 3️⃣ INSERT SAMPLE DATA
-- ========================================

INSERT INTO Customer VALUES
('Sagar Mane', 1001, 50000, 'Pune'),
('Kunal Patil', 1002, 30000, 'Mumbai'),
('Harshad Kavade', 1003, 40000, 'Delhi'),
('Priya Sharma', 1004, 60000, 'Pune');

INSERT INTO Loan VALUES
(501, 'Pune Branch', 90000),
(502, 'Mumbai Branch', 30000),
(503, 'Delhi Branch', 60000),
(504, 'Pune Branch', 150000);

INSERT INTO Borrower VALUES
('Sagar Mane', 501),
('Kunal Patil', 502),
('Harshad Kavade', 503),
('Priya Sharma', 504);
```

```
-- ========================================
-- 4️⃣ CREATE STORED PROCEDURE
-- ========================================
DELIMITER $$

CREATE PROCEDURE categorize_borrowers()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE v_cust_name VARCHAR(50);
    DECLARE v_accno INT;
    DECLARE v_loan_no INT;
    DECLARE v_branch VARCHAR(50);
    DECLARE v_amount DECIMAL(12,2);
    DECLARE v_category VARCHAR(20);

    -- Cursor to fetch borrower details
    DECLARE borrower_cur CURSOR FOR
        SELECT c.Cust_name, c.AccNo, l.Loan_no, l.Branch_name, l.Amount
        FROM Customer c
        JOIN Borrower b ON c.Cust_name = b.Cust_name
        JOIN Loan l ON b.Loan_no = l.Loan_no;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN borrower_cur;

    read_loop: LOOP
        FETCH borrower_cur INTO v_cust_name, v_accno, v_loan_no, v_branch, v_amount;
        IF done THEN
            LEAVE read_loop;
```

```
        END IF;

        -- Categorize loan amount
        IF v_amount > 100000 THEN
            SET v_category = 'Critical';
        ELSEIF v_amount BETWEEN 50000 AND 100000 THEN
            SET v_category = 'Moderate';
        ELSE
            SET v_category = 'Nominal';
        END IF;

        -- Insert categorized data
        INSERT INTO Borrower_Category (Cust_name, AccNo, Loan_no, Branch_name, Amount,
Category)
        VALUES (v_cust_name, v_accno, v_loan_no, v_branch, v_amount, v_category);
    END LOOP;

    CLOSE borrower_cur;
END $$

DELIMITER ;

-- ========================================
-- 5️⃣ EXECUTE THE PROCEDURE
-- ========================================
CALL categorize_borrowers();

-- ========================================
-- 6️⃣ VIEW THE RESULTS
-- ========================================
SELECT * FROM Borrower_Category;
```

Assign 5

```
DELIMITER $$

CREATE PROCEDURE find_loan_eligible_customers()
BEGIN
    -- Declare variables
    DECLARE v_name VARCHAR(100);
    DECLARE v_acc INT;
    DECLARE v_branch VARCHAR(100);
    DECLARE v_balance DECIMAL(15,2);
    DECLARE done INT DEFAULT 0;

    -- Declare cursor
    DECLARE cust_cursor CURSOR FOR
        SELECT c.Cust_name, c.Acc_no, a.branch_name, a.balance
        FROM Customer c
        JOIN Account a ON c.Acc_no = a.Acc_no
        WHERE c.Cust_name NOT IN (SELECT b.Cust_name FROM Borrower b);

    -- Declare handler for end of cursor
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    -- Open cursor
    OPEN cust_cursor;

    read_loop: LOOP
        FETCH cust_cursor INTO v_name, v_acc, v_branch, v_balance;
        IF done = 1 THEN
            LEAVE read_loop;
```

```
        END IF;

        -- Check if balance > 10000
        IF v_balance > 10000 THEN
            INSERT INTO loan_eligibility (Cust_name, Acc_no, branch_name, balance)
            VALUES (v_name, v_acc, v_branch, v_balance);
        END IF;
    END LOOP;

    CLOSE cust_cursor;

END$$

DELIMITER ;
```

How to Execute

In **MySQL Workbench**, just run:

```
CALL find_loan_eligible_customers();
```

Assign 6

```
DELIMITER $$

CREATE PROCEDURE find_loan_eligible_customers()
BEGIN
   -- Declare local variables
   DECLARE v_name VARCHAR(50);
   DECLARE v_acc_no INT;
   DECLARE v_branch VARCHAR(50);
   DECLARE v_balance DECIMAL(10,2);
   DECLARE done INT DEFAULT 0;

   -- Declare cursor for customers who have NOT taken a loan
   DECLARE cust_cursor CURSOR FOR
      SELECT c.Cust_name, c.Acc_no, a.branch_name, a.balance
      FROM Customer c
      JOIN Account a ON c.Acc_no = a.Acc_no
      WHERE c.Cust_name NOT IN (SELECT b.Cust_name FROM Borrower b);

   -- Declare handler to exit loop
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

   -- Open cursor
   OPEN cust_cursor;

   read_loop: LOOP
      FETCH cust_cursor INTO v_name, v_acc_no, v_branch, v_balance;
      IF done = 1 THEN
         LEAVE read_loop;
      END IF;
```

```
        -- Check balance > 10000, then insert
        IF v_balance > 10000 THEN
            INSERT INTO loan_eligibility (Cust_name, Acc_no, branch_name, balance)
            VALUES (v_name, v_acc_no, v_branch, v_balance);
        END IF;

    END LOOP;

    -- Close cursor
    CLOSE cust_cursor;

END$$

DELIMITER ;
```

To execute code

```
CALL find_loan_eligible_customers();
```

Assign 7

Perfect 👏 — this is a **comprehensive PL/SQL triggers problem**, where we'll handle all BEFORE and AFTER triggers for **INSERT**, **UPDATE**, and **DELETE** operations on the EMPLOYEE table.

Here's a **complete, clean, and well-commented solution** following your schema and conditions.

🧱 SCHEMA RECAP

Tables:

- **EMPLOYEE**(Emp_Id, First_Name, Last_Name, Email, Phone_No, Hire_Date, Job_Profile, Salary, HRA)
- **COMPANY_INFO**(Emp_Count, Salary_Expenses)
- **EMP_LOG**(Emp_Id, Old_Salary, New_Salary, Edit_Time, Job_Status)

✅ TRIGGERS IMPLEMENTATION

1️⃣ BEFORE INSERT Trigger

Removes extra spaces, converts JOB_PROFILE to uppercase.

```
CREATE OR REPLACE TRIGGER trg_before_insert_employee
BEFORE INSERT ON EMPLOYEE
FOR EACH ROW
BEGIN
   -- Trim spaces from names
   :NEW.First_Name := TRIM(:NEW.First_Name);
   :NEW.Last_Name  := TRIM(:NEW.Last_Name);

   -- Convert Job Profile to uppercase
   :NEW.Job_Profile := UPPER(:NEW.Job_Profile);
END;
/
```

2️⃣ AFTER INSERT Trigger

Updates EMP_LOG and COMPANY_INFO.

```
CREATE OR REPLACE TRIGGER trg_after_insert_employee
AFTER INSERT ON EMPLOYEE
FOR EACH ROW
BEGIN
   -- Insert into EMP_LOG table
   INSERT INTO EMP_LOG (Emp_Id, Old_Salary, New_Salary, Edit_Time, Job_Status)
   VALUES (:NEW.Emp_Id, NULL, :NEW.Salary, SYSDATE, 'ACTIVE');

   -- Update COMPANY_INFO table
   UPDATE COMPANY_INFO
   SET Emp_Count = Emp_Count + 1,
      Salary_Expenses = Salary_Expenses + :NEW.Salary;
END;
/
```

3️⃣ BEFORE UPDATE Trigger

Converts HRA (like 10%) into decimal (like 0.1).

```
CREATE OR REPLACE TRIGGER trg_before_update_employee
BEFORE UPDATE ON EMPLOYEE
FOR EACH ROW
BEGIN
   IF UPDATING('HRA') THEN
      -- Convert percentage (like 10) into decimal (0.1)
      IF :NEW.HRA > 1 THEN
         :NEW.HRA := :NEW.HRA / 100;
      END IF;
   END IF;
END;
/
```

4️⃣ AFTER UPDATE Trigger

Updates salary based on new HRA, logs change, and updates company info.

```
CREATE OR REPLACE TRIGGER trg_after_update_employee
AFTER UPDATE ON EMPLOYEE
FOR EACH ROW
DECLARE
   v_salary_diff NUMBER;
BEGIN
   IF UPDATING('HRA') THEN
      -- Recalculate total salary
      :NEW.Salary := :OLD.Salary + (:OLD.Salary * :NEW.HRA);

      -- Update EMP_LOG table
      INSERT INTO EMP_LOG (Emp_Id, Old_Salary, New_Salary, Edit_Time, Job_Status)
      VALUES (:OLD.Emp_Id, :OLD.Salary, :NEW.Salary, SYSDATE, 'UPDATED');

      -- Update COMPANY_INFO salary expenses
      v_salary_diff := :NEW.Salary - :OLD.Salary;
      UPDATE COMPANY_INFO
      SET Salary_Expenses = Salary_Expenses + v_salary_diff;
   END IF;
END;
/
```

---

5 BEFORE DELETE Trigger

Marks employee as deleted in EMP_LOG.

```
CREATE OR REPLACE TRIGGER trg_before_delete_employee
BEFORE DELETE ON EMPLOYEE
FOR EACH ROW
BEGIN
   UPDATE EMP_LOG
   SET Job_Status = 'DELETED',
```

```
    Edit_Time = SYSDATE
  WHERE Emp_Id = :OLD.Emp_Id;
END;
/
```

6️⃣ AFTER DELETE Trigger

Updates COMPANY_INFO to reflect one less employee.

```
CREATE OR REPLACE TRIGGER trg_after_delete_employee
AFTER DELETE ON EMPLOYEE
FOR EACH ROW
BEGIN
  UPDATE COMPANY_INFO
  SET Emp_Count = Emp_Count - 1,
    Salary_Expenses = Salary_Expenses - :OLD.Salary;
END;
/
```

⚙️ EXECUTION FLOW SUMMARY

Operation	BEFORE Trigger Action	AFTER Trigger Action
INSERT	Trim + Uppercase	Log entry + Update Company info
UPDATE	Convert HRA to decimal	Recalculate salary + Log + Update Company info
DELETE	Mark DELETED in log	Update Company info

Would you like me to include **table creation scripts** (with sample data) too so you can run all these triggers directly in Oracle SQL Developer or VS Code SQL extension?

Assign 9

Here are the MongoDB shell commands to perform the 12 CRUD operations you've listed.

---

1. Create Collection and Insert Documents

This command inserts five sample documents into the teachers collection. We use insertMany to add them all at once. Note the use of new ISODate() for proper date objects and nested objects for experience and salary.

JavaScript

```
db.teachers.insertMany([
 {
  "name": "Dr. Alice Smith",
  "qualifications": ["PhD", "M.Tech"],
  "deptno": 101,
  "deptname": "Computer",
  "designation": "Professor",
  "experience": { "industry": 5, "teaching": 15 },
  "salary": { "basic": 80000, "TA": 5000, "DA": 5000, "HRA": 20000 },
  "date_of_joining": new ISODate("2005-08-01T00:00:00Z"),
  "appointment_nature": "Permanent",
  "area_of_expertise": ["Machine Learning", "AI"]
 },
 {
  "name": "Bob Johnson",
  "qualifications": ["M.Tech"],
  "deptno": 102,
  "deptname": "IT",
  "designation": "Asst. Professor",
  "experience": { "industry": 2, "teaching": 6 },
  "salary": { "basic": 60000, "TA": 3000, "DA": 3000, "HRA": 15000 },
  "date_of_joining": new ISODate("2017-07-15T00:00:00Z"),
  "appointment_nature": "Permanent",
  "area_of_expertise": ["Database Systems", "Web Dev"],
  "publications": 12
 },
 {
  "name": "Carol Williams",
```

```
  "qualifications": ["PhD", "M.E."],
  "deptno": 103,
  "deptname": "E&TC",
  "designation": "Assoc. Professor",
  "experience": { "industry": 4, "teaching": 8 },
  "salary": { "basic": 70000, "TA": 4000, "DA": 4000, "HRA": 18000 },
  "date_of_joining": new ISODate("2015-01-10T00:00:00Z"),
  "appointment_nature": "Permanent",
  "area_of_expertise": ["VLSI", "Embedded Systems"]
},
{
  "name": "David Brown",
  "qualifications": ["M.Sc"],
  "deptno": 100,
  "deptname": "First Year",
  "designation": "Asst. Professor",
  "experience": { "industry": 1, "teaching": 3 },
  "salary": { "basic": 40000, "TA": 2000, "DA": 2000, "HRA": 10000 },
  "date_of_joining": new ISODate("2022-06-20T00:00:00Z"),
  "appointment_nature": "Adhoc",
  "area_of_expertise": ["Physics", "Maths"]
},
{
  "name": "Eve Davis",
  "qualifications": ["M.Tech"],
  "deptno": 101,
  "deptname": "Computer",
  "designation": "Asst. Professor",
  "experience": { "industry": 0, "teaching": 2 },
  "salary": { "basic": 35000, "TA": 1500, "DA": 1500, "HRA": 8000 },
  "date_of_joining": new ISODate("2023-08-01T00:00:00Z"),
```

```
  "appointment_nature": "Adhoc",
  "area_of_expertise": ["Data Structures"]
 }
])
```

---

2. Find All Teachers

Use find() with an empty query object {} to match all documents.

JavaScript

```
db.teachers.find({})
```

---

3. Find All Teachers of Computer Department

Provide a query filter to match the deptname field.

JavaScript

```
db.teachers.find({ "deptname": "Computer" })
```

---

4. Find Teachers of Computer, IT & First Year Departments

Use the $in operator to find documents where deptname matches any value in the provided array.

JavaScript

```
db.teachers.find({ "deptname": { $in: ["Computer", "IT", "First Year"] } })
```

---

5. Find Teachers of Computer, IT, E&TC with Salary 70k-100k

This query combines the $in operator for departments with a range query for salary. We use $gte (greater than or equal to) and $lte (less than or equal to).

Note: The query below assumes "salary" refers to the salary.basic field. If you mean *total salary* (basic + TA + DA + HRA), the query is more complex and requires $expr.

- **Query for Total Salary (Advanced):**

JavaScript

```
db.teachers.find({
 "deptname": { $in: ["Computer", "IT", "E&TC"] },
 $expr: {
  $and: [
   { $gte: [ { $add: ["$salary.basic", "$salary.TA", "$salary.DA", "$salary.HRA"] }, 70000 ] },
```

```
    { $lte: [ { $add: ["$salary.basic", "$salary.TA", "$salary.DA", "$salary.HRA"] }, 100000 ] }
  ]
 }
})
```

Here is the simpler query, filtering by **basic salary**:

JavaScript

```
db.teachers.find({
 "deptname": { $in: ["Computer", "IT", "E&TC"] },
 "salary.basic": { $gte: 70000, $lte: 100000 }
})
```

---

6. Update Experience with upsert

This command tries to find a teacher named "Bob Johnson" and set his teaching experience to 10. The { upsert: true } option means:

- If "Bob Johnson" is found, update him.
- If no document matches the filter { name: "Bob Johnson" }, insert a new document combining the filter and the $set operation.

Note: The update() method is deprecated. The modern methods are updateOne() (default behavior of update) or updateMany().

JavaScript

```
db.teachers.update(
 { "name": "Bob Johnson" },
 { $set: { "experience.teaching": 10 } },
 { "upsert": true }
)
```

---

7. Find Name and Experience, Sorted by Experience

This query uses:

1. **Projection:** The second argument { name: 1, experience: 1, _id: 0 } specifies which fields to return. 1 means include, and _id: 0 means exclude the default ID.
2. **Sort:** .sort({ "experience.teaching": -1 }) orders the results by teaching experience in descending order (-1). Use 1 for ascending.

JavaScript

```
db.teachers.find(
  {},
  { "name": 1, "experience": 1, "_id": 0 }
).sort({ "experience.teaching": -1 })
```

---

8. Use save() Method to Insert

The save() method inserts a document if it does **not** contain an _id field.

Note: save() is deprecated. The modern method is insertOne().

JavaScript

```
db.teachers.save({
  "name": "Frank White",
  "qualifications": ["M.E."],
  "deptno": 102,
  "deptname": "IT",
  "designation": "Asst. Professor",
  "experience": { "industry": 3, "teaching": 3 },
  "salary": { "basic": 45000, "TA": 2500, "DA": 2500, "HRA": 11000 },
  "date_of_joining": new ISODate("2021-09-01T00:00:00Z"),
  "appointment_nature": "Permanent",
  "area_of_expertise": ["Cloud Computing"]
})
```

---

9. Use update() to Change Designation (Multiple)

To update *all* documents that match a filter using the old update() method, you must add the { multi: true } option. This query finds all teachers with 10 or more years of teaching experience and sets their designation to "Professor".

Note: The modern, preferred method for this operation is updateMany().

JavaScript

```
db.teachers.update(
  { "experience.teaching": { $gte: 10 } },
  { $set: { "designation": "Professor" } },
  { "multi": true }
```

)

---

10. Use save() Method to Change Designation

Using save() for an update is a multi-step "read-modify-write" process. You fetch the *entire* document, modify it in your client (the shell), and then save the *entire* document back. save() works for updates if the document you pass to it **contains** an _id field.

JavaScript

// Step 1: Find and store the document in a variable

var teacher = db.teachers.findOne({ "name": "Carol Williams" })

// Step 2: Modify the document in the variable

teacher.designation = "Professor"

// Step 3: Save the modified document back to the collection

// Because 'teacher' has an _id, save() will perform an update.

db.teachers.save(teacher)

---

11. Delete Documents

This command finds all documents where appointment_nature is "Adhoc" and deletes them.

Note: The old method was remove(). The modern, preferred method is deleteMany().

JavaScript

db.teachers.deleteMany({ "appointment_nature": "Adhoc" })

---

12. Display First 3 Documents, Sorted by Experience, with pretty()

This command chains three methods:

1. .sort({ "experience.teaching": 1 }): Sorts all documents by teaching experience in ascending order (1).
2. .limit(3): Takes only the first 3 documents from the sorted list.
3. .pretty(): Formats the JSON output for easier reading.

JavaScript

db.teachers.find().sort({ "experience.teaching": 1 }).limit(3).pretty()

Assign 10

Here are the MongoDB commands for setting up the collection, implementing indexes, and solving the aggregation queries.

🛠️ **Setup: Insert the Data**

First, let's insert the documents into the purchase_orders collection.

JavaScript

```
db.purchase_orders.insertMany([
  {product: "toothbrush", total: 4.75, customer: "Mike"},
  {product: "guitar", total: 199.99, customer: "Tom"},
  {product: "milk", total: 11.33, customer: "Mike"},
  {product: "pizza", total: 8.50, customer: "Karen"},
  {product: "toothbrush", total: 4.75, customer: "Karen"},
  {product: "pizza", total: 4.75, customer: "Dave"},
  {product: "toothbrush", total: 4.75, customer: "Mike"}
])
```

📈 Implementing Indexes

Here are examples of the three main types of indexes.

1. Single Field Index

This is the most common index, speeding up queries on a single field. Let's index the customer field.

JavaScript

```
db.purchase_orders.createIndex({ customer: 1 })
```

(Here, 1 indicates ascending order. -1 would be descending.)

2. Compound Index

This index speeds up queries that filter on multiple fields. Let's create one for queries that look for a specific product and sort by total.

JavaScript

```
db.purchase_orders.createIndex({ product: 1, total: -1 })
```

3. Multikey Index

A multikey index is created on a field that contains an array value. Since your current schema doesn't have an array, one cannot be created.

However, if you had a document like this:

```
{ product: "milk", total: 11.33, customer: "Mike", tags: ["dairy", "food", "beverage"] }
```

You could create a multikey index on the tags field with this command:

```
db.purchase_orders.createIndex({ tags: 1 })
```

MongoDB would then create separate index entries for "dairy", "food", and "beverage", all pointing to this one document.

📊 Aggregation Queries

Here are the aggregation pipeline queries to answer your questions.

1. Find out how many toothbrushes were sold

We filter ($match) for the product and then count ($count) the results.

JavaScript

```
db.purchase_orders.aggregate([
  { $match: { product: "toothbrush" } },
  { $count: "toothbrush_sales_count" }
])
```

2. Find how much money has been earned by selling toothbrushes and pizza

We filter ($match) for both products, then group ($group) all results into one (_id: null) and sum ($sum) their totals.

JavaScript

```
db.purchase_orders.aggregate([
  { $match: { product: { $in: ["toothbrush", "pizza"] } } },
  { $group: {
      _id: null,
      total_earnings: { $sum: "$total" }
    }
  }
])
```

3. Find the list of all sold products

We group ($group) by the product field. This automatically creates a unique list of all products.

JavaScript

```
db.purchase_orders.aggregate([
  { $group: { _id: "$product" } }
])
```

(Alternatively, db.purchase_orders.distinct("product") achieves the same result outside of an aggregation pipeline.)

4. Find the total amount of money spent by each customer

We group ($group) by customer and sum ($sum) the total for each customer.

JavaScript

```
db.purchase_orders.aggregate([
  { $group: {
      _id: "$customer",
      total_spent: { $sum: "$total" }
    }
  }
])
```

5. Find how much has been spent on each product and sort it by amount spent

We group ($group) by product, sum ($sum) the totals, and then sort ($sort) the results in descending order (-1).

JavaScript

```
db.purchase_orders.aggregate([
  { $group: {
      _id: "$product",
      total_earnings: { $sum: "$total" }
    }
  },
  { $sort: { total_earnings: -1 } }
])
```

6. Find the product with the least earnings

This is similar to #5, but we sort in ascending order (1) and take only the first ($limit: 1) result.

JavaScript

```
db.purchase_orders.aggregate([
  { $group: {
      _id: "$product",
      total_earnings: { $sum: "$total" }
    }
  },
```

```
  { $sort: { total_earnings: 1 } },
  { $limit: 1 }
])
```

7. Find how much money each customer has spent on toothbrushes and pizza

We first filter ($match) for only those products, then group ($group) by customer and sum ($sum) the totals.

JavaScript

```
db.purchase_orders.aggregate([
  { $match: { product: { $in: ["toothbrush", "pizza"] } } },
  { $group: {
      _id: "$customer",
      total_spent: { $sum: "$total" }
    }
  }
])
```

8. Find the customer who has given highest business for the product toothbrush

We filter ($match) for "toothbrush", group ($group) by customer, sum ($sum) their spending, sort ($sort) descending, and take the top one ($limit: 1).

JavaScript

```
db.purchase_orders.aggregate([
  { $match: { product: "toothbrush" } },
  { $group: {
      _id: "$customer",
      total_spent_on_toothbrush: { $sum: "$total" }
    }
  },
  { $sort: { total_spent_on_toothbrush: -1 } },
  { $limit: 1 }
])
```

Assign 11

Your insert command for "Astronomy 101" was incomplete. I have completed it below with sample data so we can run queries.

1. 🛠️ Complete Insert Statements

Here are the three documents to be inserted into the classes collection.

JavaScript

```
db.classes.insertMany([
 {
  class : "Philosophy 101",
  startDate : new Date(2016, 1, 10),
  students : [
   {fName : "Dale", lName : "Cooper", age : 42},
   {fName : "Lucy", lName : "Moran", age : 35},
   {fName : "Tommy", lName : "Hill", age : 44}
  ],
  cost : 1600,
  professor : "Paul Slugman",
  topics : "Socrates,Plato,Aristotle,Francis Bacon",
  book: {
   isbn: "1133612105",
   title: "Philosophy : A Text With Readings",
   price: 165.42
  }
 },
 {
  class : "College Algebra",
  startDate : new Date(2016, 1, 11),
  students : [
   {fName : "Dale", lName : "Cooper", age : 42},
   {fName : "Laura", lName : "Palmer", age : 22},
   {fName : "Donna", lName : "Hayward", age : 21},
   {fName : "Shelly", lName : "Johnson", age : 24}
  ],
  cost : 1500,
```

```
  professor : "Rhonda Smith",
  topics : "Rational Expressions,Linear Equations,Quadratic Equations",
  book: {
   isbn: "0321671791",
   title: "College Algebra",
   price: 179.40
  }
 },
 {
  class : "Astronomy 101",
  startDate : new Date(2016, 1, 11),
  students : [
   {fName : "Bobby", lName : "Briggs", age : 21},
   {fName : "Audrey", lName : "Horne", age : 20},
   {fName : "James", lName : "Hurley", age : 21}
  ],
  cost : 1400,
  professor : "Dr. Jacoby",
  topics : "Stars,Planets,Galaxies",
  book: {
   isbn: "032182246X",
   title: "The Cosmic Perspective",
   price: 185.00
  }
 }
])
```

---

2. 🗺️ Map-Reduce Example: Count Students per Class

You haven't specified a query, so here is a classic Map-Reduce example: **Count the number of students in each class.**

The mapReduce command requires two JavaScript functions: map and reduce.

The map Function

This function is executed once for *each* document in the collection. Its job is to emit a key-value pair.

- **Key:** The field we want to group by (the class name: this.class).
- **Value:** The value we want to aggregate (the number of students: this.students.length).

JavaScript

```
var mapFunction = function() {
   emit(this.class, this.students.length);
};
```

The reduce Function

This function is executed for *each unique key* emitted by the map function. It takes an array of the values emitted for that key and "reduces" them to a single result.

- **Key:** The class name (e.g., "Philosophy 101").
- **Values:** An array of the counts (e.g., [3]).

JavaScript

```
var reduceFunction = function(keyClass, valuesStudentCounts) {
   // In this case, the array will only have one value (e.g., [3]),
   // so we just sum the array to get the final count.
   return Array.sum(valuesStudentCounts);
};
```

The mapReduce Command

Finally, we call db.collection.mapReduce() with our two functions and an out option to specify where to put the results. Using { inline: 1 } displays the results directly in the shell.

JavaScript

```
db.classes.mapReduce(
   mapFunction,
   reduceFunction,
   { out: { inline: 1 } }
)
```

3. 🏁 Result

The output shows the result for each unique key (_id) and its final reduced value.

JSON

```
{
```

```
  "results" : [
    { "_id" : "Astronomy 101", "value" : 3 },
    { "_id" : "College Algebra", "value" : 4 },
    { "_id" : "Philosophy 101", "value" : 3 }
  ],
  "ok" : 1
}
```

💡 **Modern Alternative: The Aggregation Pipeline**

It's important to know that mapReduce is considered a legacy feature. It is **much slower** and more complex than the modern **Aggregation Pipeline**.

The exact same result can be achieved with this single, more efficient command:

JavaScript

```
db.classes.aggregate([
 {
  $project: {
   _id: 0,
   class: "$class",
   student_count: { $size: "$students" }
  }
 }
])
```

Aggregation Output (cleaner and faster):

JSON

```
[
 { "class" : "Philosophy 101", "student_count" : 3 },
 { "class" : "College Algebra", "student_count" : 4 },
 { "class" : "Astronomy 101", "student_count" : 3 }
]
```