DBMS Problem Set Solutions - MySQL

November 7, 2025

Problem Statement 1

Database 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)

Q.1: Create above tables with appropriate constraints

```sql
-- Create branch table first as others depend on it
CREATE TABLE branch (
    branch_name VARCHAR(50) PRIMARY KEY,
    branch_city VARCHAR(50) NOT NULL,
    assets DECIMAL(15,2)
);

-- Create Account table
CREATE TABLE Account (
    Acc_no VARCHAR(20) PRIMARY KEY,
    branch_name VARCHAR(50),
    balance DECIMAL(15,2) NOT NULL CHECK (balance >= 0),
    FOREIGN KEY (branch_name) REFERENCES branch(branch_name)
);

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

-- Create Depositor table (linking customer and account)
CREATE TABLE Depositor (
    cust_name VARCHAR(100),
```

```
    acc_no VARCHAR(20),
    PRIMARY KEY (cust_name, acc_no),
    FOREIGN KEY (cust_name) REFERENCES customer(cust_name),
    FOREIGN KEY (acc_no) REFERENCES Account(Acc_no)
);

-- Create Loan table
CREATE TABLE Loan (
    loan_no VARCHAR(20) PRIMARY KEY,
    branch_name VARCHAR(50),
    amount DECIMAL(15,2) NOT NULL CHECK (amount > 0),
    FOREIGN KEY (branch_name) REFERENCES branch(branch_name)
);

-- Create Borrower table (linking customer and loan)
CREATE TABLE Borrower (
    cust_name VARCHAR(100),
    loan_no VARCHAR(20),
    PRIMARY KEY (cust_name, loan_no),
    FOREIGN KEY (cust_name) REFERENCES customer(cust_name),
    FOREIGN KEY (loan_no) REFERENCES Loan(loan_no)
);

-- --- Insert Sample Data ---
INSERT INTO branch VALUES ('Akurdi', 'Pune', 5000000);
INSERT INTO branch VALUES ('Kothrud', 'Pune', 7000000);
INSERT INTO branch VALUES ('Thane', 'Mumbai', 9000000);

INSERT INTO Account VALUES ('A101', 'Akurdi', 50000);
INSERT INTO Account VALUES ('A102', 'Kothrud', 80000);
INSERT INTO Account VALUES ('A103', 'Akurdi', 120000);

INSERT INTO customer VALUES ('Amit', 'MG Road', 'Pune');
INSERT INTO customer VALUES ('Sunita', 'FC Road', 'Pune');
INSERT INTO customer VALUES ('Rahul', 'Link Road', 'Mumbai');

INSERT INTO Depositor VALUES ('Amit', 'A101');
INSERT INTO Depositor VALUES ('Sunita', 'A102');
INSERT INTO Depositor VALUES ('Amit', 'A103');

INSERT INTO Loan VALUES ('L201', 'Akurdi', 150000);
INSERT INTO Loan VALUES ('L202', 'Thane', 250000);
INSERT INTO Loan VALUES ('L203', 'Akurdi', 30000);

INSERT INTO Borrower VALUES ('Amit', 'L201');
INSERT INTO Borrower VALUES ('Rahul', 'L202');
INSERT INTO Borrower VALUES ('Sunita', 'L203');
```

Q.2: Create view/alias for customer table as cust

```
CREATE VIEW cust AS
SELECT * FROM customer;
```

Q.3: Add customer phone number in Customer table

```sql
ALTER TABLE customer
ADD phone_number VARCHAR(15);
```

Q.4: Delete phone number attribute from Customer table

```sql
ALTER TABLE customer
DROP COLUMN phone_number;
```

Q.5: Find the names of all branches in loan relation

```sql
SELECT DISTINCT branch_name
FROM Loan;
```

Q.6: Find all customers who have a loan - Find their names, loan_no and loan amount

```sql
SELECT b.cust_name, b.loan_no, l.amount
FROM Borrower b
JOIN Loan l ON b.loan_no = l.loan_no;
```

Q.7: List all customers in alphabetical order who have loan from Akurdi branch

```sql
SELECT b.cust_name
FROM Borrower b
JOIN Loan l ON b.loan_no = l.loan_no
WHERE l.branch_name = 'Akurdi'
ORDER BY b.cust_name;
```

Q.8: Find average account balance at Akurdi branch

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

Q.9: Find no. of depositors at each branch

```sql
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;
```

Problem Statement 2

Database Schema: (Same as Problem 1)

Q.1: Create above tables with appropriate constraints

```
-- The DDL (CREATE TABLE) statements are identical to those
-- in Problem Statement 1, Q.1.
```

Q.2: Modify "assets" attribute of branch table to "Property"

```
-- MySQL syntax for renaming column
ALTER TABLE branch CHANGE COLUMN assets Property DECIMAL(15,2);
```

Q.3: 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;
```

Q.4: Find the average account balance at each branch

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

Q.5: Find the branches where average account balance > 12000

```
SELECT branch_name, AVG(balance) AS avg_balance
FROM Account
GROUP BY branch_name
HAVING AVG(balance) > 12000;
```

Q.6: Find number of tuples in customer relation

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

Q.7: Calculate total loan amount given by bank

```
SELECT SUM(amount) AS total_loan_amount
FROM Loan;
```

Q.8: Delete all loans with loan amount between 1300 and 1500

```sql
-- Note: This may fail if there are corresponding records in
-- the 'Borrower' table due to foreign key constraints.
-- You might need to delete from 'Borrower' first or
-- have 'ON DELETE CASCADE' set on the foreign key.

-- Step 1: Delete from Borrower (child table)
DELETE FROM Borrower
WHERE loan_no IN (
    SELECT loan_no FROM Loan WHERE amount BETWEEN 1300 AND 1500
);

-- Step 2: Delete from Loan (parent table)
DELETE FROM Loan
WHERE amount BETWEEN 1300 AND 1500;
```

Problem Statement 3

Database Schema:

- cust_mstr(cust_no, fname, lname)
- add_dets(code_no, add1, add2, state, city, pincode)

Q.1: Create above Tables with suitable data

```sql
-- Assuming cust_no is the primary key and code_no is a foreign key
CREATE TABLE cust_mstr (
    cust_no VARCHAR(10) PRIMARY KEY,
    fname VARCHAR(50),
    lname VARCHAR(50)
);

CREATE TABLE add_dets (
    address_id INT AUTO_INCREMENT PRIMARY KEY,
    code_no VARCHAR(10),
    add1 VARCHAR(100),
    add2 VARCHAR(100),
    state VARCHAR(50),
    city VARCHAR(50),
    pincode VARCHAR(10),
    FOREIGN KEY (code_no) REFERENCES cust_mstr(cust_no)
);

-- Sample Data
INSERT INTO cust_mstr VALUES ('C001', 'xyz', 'pqr');
INSERT INTO cust_mstr VALUES ('C002', 'abc', 'def');

INSERT INTO add_dets (code_no, add1, add2, state, city, pincode)
VALUES ('C001', '123 Main St', 'Apt 4B', 'Maharashtra', 'Pune', '411001');
INSERT INTO add_dets (code_no, add1, add2, state, city, pincode)
VALUES ('C002', '456 Park Ave', '', 'Delhi', 'New Delhi', '110001');
```

Q.2: Retrieve the address of customer Fname as 'xyz' and Lname as 'pqr'

```sql
SELECT a.add1, a.add2, a.state, a.city, a.pincode
FROM add_dets a
JOIN cust_mstr c ON a.code_no = c.cust_no
WHERE c.fname = 'xyz' AND c.lname = 'pqr';
```

Q.3: Create View on add_dets table and perform insert update delete

```sql
-- Create View
CREATE VIEW v_address_details AS
SELECT city, pincode
FROM add_dets;

-- Insert (Inserts into the base table add_dets)
-- Note: This will only work if other NOT NULL columns in add_dets
-- have default values, or are nullable.
INSERT INTO v_address_details (city, pincode)
VALUES ('Mumbai', '400001');

-- Update (Updates the base table)
UPDATE v_address_details
SET pincode = '400050'
WHERE city = 'Mumbai';

-- Delete (Deletes from the base table)
DELETE FROM v_address_details
WHERE city = 'Mumbai';
```

Q.4: Create following Tables

Schema:

- emp_mstr(e_mpno, f_name, l_name, m_name, dept, desg, branch_no)
- branch_mstr(name, b_no)

```sql
CREATE TABLE branch_mstr (
    b_no VARCHAR(10) PRIMARY KEY,
    name VARCHAR(50)
);

CREATE TABLE emp_mstr (
    e_mpno VARCHAR(10) PRIMARY KEY,
    f_name VARCHAR(50),
    l_name VARCHAR(50),
    m_name VARCHAR(50),
    dept VARCHAR(50),
    desg VARCHAR(50),
    branch_no VARCHAR(10),
    FOREIGN KEY (branch_no) REFERENCES branch_mstr(b_no)
);
```

Q.5: List the employee details along with branch names

```sql
SELECT e.*, b.name AS branch_name
FROM emp_mstr e
JOIN branch_mstr b ON e.branch_no = b.b_no;
```

Problem Statement 4

Part 1

Schema:

- cust_mstr(custno, fname, lname)
- acc_fd_cust_dets(codeno, acc_fd_no)
- fd_dets(fd_sr_no, amt)

Q.4.1: Create Tables

```sql
CREATE TABLE cust_mstr (
    custno VARCHAR(10) PRIMARY KEY,
    fname VARCHAR(50),
    lname VARCHAR(50)
);

CREATE TABLE fd_dets (
    fd_sr_no VARCHAR(10) PRIMARY KEY,
    amt DECIMAL(15,2)
);

CREATE TABLE acc_fd_cust_dets (
    id INT AUTO_INCREMENT PRIMARY KEY,
    codeno VARCHAR(10),
    acc_fd_no VARCHAR(10),
    FOREIGN KEY (codeno) REFERENCES cust_mstr(custno),
    FOREIGN KEY (acc_fd_no) REFERENCES fd_dets(fd_sr_no)
);
```

Q.4.2: List customer holding FD > 5000

```sql
SELECT DISTINCT c.fname, c.lname
FROM cust_mstr c
JOIN acc_fd_cust_dets a ON c.custno = a.codeno
JOIN fd_dets f ON a.acc_fd_no = f.fd_sr_no
WHERE f.amt > 5000;
```

Q.4.3: Create view on cust_mstr and acc_fd_cust_dets and perform DML

```sql
CREATE VIEW v_cust_fd AS
SELECT c.fname, a.acc_fd_no
FROM cust_mstr c
JOIN acc_fd_cust_dets a ON c.custno = a.codeno;
```

```
5
6  -- NOTE: Performing INSERT, UPDATE, or DELETE operations on a
7  -- multi-table join view is complex and often not allowed
8  -- (or requires "INSTEAD OF" triggers).
9  -- A simple SELECT query on the view:
10 SELECT * FROM v_cust_fd;
```

Part 2

Schema:

- emp_mstr(emp_no, f_name, l_name, m_name, dept)
- cntc_dets(code_no, cntc_type, cntc_data)

Q.4.4: Create Tables

```
1  CREATE TABLE emp_mstr (
2      emp_no VARCHAR(10) PRIMARY KEY,
3      f_name VARCHAR(50),
4      l_name VARCHAR(50),
5      m_name VARCHAR(50),
6      dept VARCHAR(50)
7  );
8
9  CREATE TABLE cntc_dets (
10     id INT AUTO_INCREMENT PRIMARY KEY,
11     code_no VARCHAR(10),
12     cntc_type VARCHAR(20),
13     cntc_data VARCHAR(100),
14     FOREIGN KEY (code_no) REFERENCES emp_mstr(emp_no)
15 );
```

Q.4.5: List employee details using left and right join

```
1  -- Left Outer Join
2  -- (Returns all employees, and contact details if they exist)
3  SELECT e.*, c.cntc_type, c.cntc_data
4  FROM emp_mstr e
5  LEFT JOIN cntc_dets c ON e.emp_no = c.code_no;
6
7  -- Right Outer Join
8  -- (Returns all contact details, and employee details if they exist)
9  SELECT e.*, c.cntc_type, c.cntc_data
10 FROM emp_mstr e
11 RIGHT JOIN cntc_dets c ON e.emp_no = c.code_no;
```

Problem Statement 5

Schema:

- Borrower(Roll_no, Name, DateofIssue, NameofBook, Status)

- Fine(Roll_no, Date, Amt)

This requires a MySQL stored procedure to handle book returns, calculate fines, and handle exceptions.

```
-- Create the Borrower table
CREATE TABLE Borrower (
    Roll_no VARCHAR(10),
    Name VARCHAR(100),
    DateofIssue DATE,
    NameofBook VARCHAR(200),
    Status CHAR(1),
    PRIMARY KEY (Roll_no, NameofBook)
);

-- Create the Fine table
CREATE TABLE Fine (
    Roll_no VARCHAR(10),
    Date DATE,
    Amt DECIMAL(8,2)
);

-- MySQL Stored Procedure
DELIMITER //

CREATE PROCEDURE sp_return_book(
    IN p_roll_no VARCHAR(10),
    IN p_book_name VARCHAR(200)
)
BEGIN
    DECLARE v_date_of_issue DATE;
    DECLARE v_days_diff INT;
    DECLARE v_fine_amt DECIMAL(8,2) DEFAULT 0;
    DECLARE CONTINUE HANDLER FOR NOT FOUND
        BEGIN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No active issued record found for this student and book.';
        END;

    -- Step 1: Find the book record for the user
    SELECT DateofIssue INTO v_date_of_issue
    FROM Borrower
    WHERE Roll_no = p_roll_no
      AND NameofBook = p_book_name
      AND Status = 'I';

    -- Step 2 & 3: Calculate fine
    SET v_days_diff = DATEDIFF(CURDATE(), v_date_of_issue);

    IF v_days_diff > 30 THEN
        SET v_fine_amt = ((v_days_diff - 30) * 50) + (15 * 5);
    ELSEIF v_days_diff >= 15 AND v_days_diff <= 30 THEN
        SET v_fine_amt = (v_days_diff - 14) * 5;
    ELSE
        SET v_fine_amt = 0;
    END IF;

    -- Step 4: If fine > 0, store in Fine table
    IF v_fine_amt > 0 THEN
```

```
54          INSERT INTO Fine (Roll_no, Date, Amt)
55          VALUES (p_roll_no, CURDATE(), v_fine_amt);
56      END IF;
57
58      -- Step 1 (cont.): Update status to 'R' (Returned)
59      UPDATE Borrower
60      SET Status = 'R'
61      WHERE Roll_no = p_roll_no
62        AND NameofBook = p_book_name
63        AND Status = 'I';
64
65      SELECT CONCAT('Book returned successfully. Fine: ', v_fine_amt) AS
        result;
66 END //
67
68 DELIMITER ;
69
70 -- How to run this procedure:
71 CALL sp_return_book('R101', 'Database Systems');
```

Problem Statement 6

PL/SQL block to calculate area of circle for radius 1 to 10 and store in table areas.

```
1  -- Step 1: Create the target table
2  CREATE TABLE areas (
3      radius INT,
4      area DECIMAL(10,4)
5  );
6
7  -- Step 2: MySQL Procedure to populate areas table
8  DELIMITER //
9
10 CREATE PROCEDURE populate_areas()
11 BEGIN
12     DECLARE v_radius INT DEFAULT 1;
13     DECLARE v_pi DECIMAL(10,8) DEFAULT 3.14159265;
14     DECLARE v_area DECIMAL(10,4);
15
16     WHILE v_radius <= 10 DO
17         SET v_area = v_pi * (v_radius * v_radius);
18         INSERT INTO areas (radius, area) VALUES (v_radius, v_area);
19         SET v_radius = v_radius + 1;
20     END WHILE;
21
22     SELECT 'Successfully populated areas table.' AS result;
23 END //
24
25 DELIMITER ;
26
27 -- Execute the procedure
28 CALL populate_areas();
29
30 -- Check the results:
31 -- SELECT * FROM areas;
```

Problem Statement 7

Schema:

- Stud_Marks(name, total_marks)
- Result(Roll, Name, Class)

Part 1: Create Stored Procedure proc_Grade

```
DELIMITER //

CREATE PROCEDURE proc_Grade(
    IN p_marks INT,
    OUT p_class VARCHAR(50)
)
BEGIN
    IF p_marks >= 990 AND p_marks <= 1500 THEN
        SET p_class = 'Distinction';
    ELSEIF p_marks >= 900 AND p_marks <= 989 THEN
        SET p_class = 'First Class';
    ELSEIF p_marks >= 825 AND p_marks <= 899 THEN
        SET p_class = 'Higher Second Class';
    ELSE
        SET p_class = 'Other';
    END IF;
END //

DELIMITER ;
```

Part 2: MySQL block to use the procedure

```
-- Setup: Create tables
CREATE TABLE Stud_Marks (name VARCHAR(100), total_marks INT);
CREATE TABLE Result (Roll INT AUTO_INCREMENT PRIMARY KEY,
                     Name VARCHAR(100), Class VARCHAR(50));

-- Sample Data
INSERT INTO Stud_Marks VALUES ('Anil', 1000);
INSERT INTO Stud_Marks VALUES ('Bina', 950);
INSERT INTO Stud_Marks VALUES ('Chetan', 850);
INSERT INTO Stud_Marks VALUES ('Dipti', 700);

-- MySQL block to process marks
DELIMITER //

CREATE PROCEDURE process_grades()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE v_name VARCHAR(100);
    DECLARE v_marks INT;
    DECLARE v_class VARCHAR(50);

    DECLARE c_students CURSOR FOR
        SELECT name, total_marks FROM Stud_Marks;
```

```
24
25      DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
26
27      OPEN c_students;
28
29      read_loop: LOOP
30          FETCH c_students INTO v_name, v_marks;
31          IF done THEN
32              LEAVE read_loop;
33          END IF;
34
35          -- Call the procedure to get the class
36          CALL proc_Grade(v_marks, v_class);
37
38          -- Insert into the Result table
39          INSERT INTO Result (Name, Class) VALUES (v_name, v_class);
40      END LOOP;
41
42      CLOSE c_students;
43      SELECT 'Result table populated successfully.' AS result;
44  END //
45
46  DELIMITER ;
47
48  -- Execute the procedure
49  CALL process_grades();
50
51  -- Check results:
52  -- SELECT * FROM Result;
```

Problem Statement 8

PL/SQL block using parameterized cursor to merge N_RollCall into O_RollCall, skipping duplicates.

```
1  -- Setup: Create tables
2  CREATE TABLE O_RollCall (
3      roll_no INT PRIMARY KEY,
4      name VARCHAR(100),
5      dept VARCHAR(50)
6  );
7  CREATE TABLE N_RollCall (
8      roll_no INT PRIMARY KEY,
9      name VARCHAR(100),
10     dept VARCHAR(50)
11 );
12
13 -- Sample Data
14 INSERT INTO O_RollCall VALUES (1, 'Rahul', 'Comp');
15 INSERT INTO O_RollCall VALUES (2, 'Priya', 'IT');
16
17 INSERT INTO N_RollCall VALUES (2, 'Priya', 'IT');
18 INSERT INTO N_RollCall VALUES (3, 'Amit', 'E&TC');
19 INSERT INTO N_RollCall VALUES (4, 'Sneha', 'Comp');
20
21 -- MySQL Stored Procedure
```

```
DELIMITER //

CREATE PROCEDURE merge_rollcall()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE v_roll INT;
    DECLARE v_name VARCHAR(100);
    DECLARE v_dept VARCHAR(50);
    DECLARE v_count INT;

    DECLARE c_new_roll CURSOR FOR
        SELECT roll_no, name, dept FROM N_RollCall;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN c_new_roll;

    read_loop: LOOP
        FETCH c_new_roll INTO v_roll, v_name, v_dept;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Check if data already exists
        SELECT COUNT(*) INTO v_count FROM O_RollCall WHERE roll_no = v_roll
    ;

        -- If count is 0, the record is new
        IF v_count = 0 THEN
            INSERT INTO O_RollCall (roll_no, name, dept)
            VALUES (v_roll, v_name, v_dept);
            SELECT CONCAT('Merged roll no: ', v_roll) AS result;
        ELSE
            SELECT CONCAT('Skipped duplicate: ', v_roll) AS result;
        END IF;
    END LOOP;

    CLOSE c_new_roll;
END //

DELIMITER ;

-- Execute the procedure
CALL merge_rollcall();

-- Check results:
-- SELECT * FROM O_RollCall;
-- (Should contain rolls 1, 2, 3, 4)
```

Problem Statement 9

Database trigger on Library table to track updates/deletes in Library_Audit table.

```
-- Step 1: Create the main table
CREATE TABLE Library (
    book_id INT PRIMARY KEY,
    title VARCHAR(200),
```

```
    author VARCHAR(100),
    status VARCHAR(20)
);

-- Step 2: Create the audit table
CREATE TABLE Library_Audit (
    audit_id INT AUTO_INCREMENT PRIMARY KEY,
    book_id INT,
    old_title VARCHAR(200),
    old_author VARCHAR(100),
    old_status VARCHAR(20),
    action_type VARCHAR(10),
    action_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Sample Data for main table
INSERT INTO Library VALUES (101, 'SQL Complete Ref', 'ABC', 'Available');
INSERT INTO Library VALUES (102, 'PL/SQL Guide', 'XYZ', 'Issued');

-- Step 3: Create trigger for UPDATE
DELIMITER //

CREATE TRIGGER trg_library_audit_update
AFTER UPDATE ON Library
FOR EACH ROW
BEGIN
    INSERT INTO Library_Audit (
        book_id,
        old_title,
        old_author,
        old_status,
        action_type
    )
    VALUES (
        OLD.book_id,
        OLD.title,
        OLD.author,
        OLD.status,
        'UPDATE'
    );
END //

-- Step 3b: Create trigger for DELETE
CREATE TRIGGER trg_library_audit_delete
AFTER DELETE ON Library
FOR EACH ROW
BEGIN
    INSERT INTO Library_Audit (
        book_id,
        old_title,
        old_author,
        old_status,
        action_type
    )
    VALUES (
        OLD.book_id,
        OLD.title,
        OLD.author,
```

```
        OLD.status,
        'DELETE'
    );
END //

DELIMITER ;

-- --- Test the trigger ---

-- 1. Perform an UPDATE
-- UPDATE Library SET status = 'Maintenance' WHERE book_id = 101;

-- 2. Perform a DELETE
-- DELETE FROM Library WHERE book_id = 102;

-- 3. Check the audit table
-- SELECT * FROM Library_Audit;
```

Problem Statement 10

Database Connectivity: Program to implement MySQL/Oracle connectivity with a front-end language.

Answer: This is a conceptual task. Below is an example using **Python** with the mysql-connector-python library to connect to a MySQL database.

```
import mysql.connector
from mysql.connector import Error

def create_connection():
    """ Create a database connection to a MySQL database """
    conn = None
    try:
        conn = mysql.connector.connect(
            host='localhost',
            database='bank_db',  # Assuming bank db from P1
            user='your_username',
            password='your_password'
        )
        if conn.is_connected():
            print('Connected to MySQL database')
        return conn
    except Error as e:
        print(f"Error while connecting to MySQL: {e}")
        return None

def add_customer(conn, cust_name, street, city):
    """ Add a new customer to the customer table """
    try:
        cursor = conn.cursor()
        query = "INSERT INTO customer(cust_name, cust_street, cust_city)
    VALUES(%s, %s, %s)"
        cursor.execute(query, (cust_name, street, city))
        conn.commit()
        print(f"Customer '{cust_name}' added successfully.")
        cursor.close()
```

```
    except Error as e:
        print(f"Failed to add customer: {e}")

def edit_customer(conn, old_name, new_city):
    """ Edit an existing customer's city """
    try:
        cursor = conn.cursor()
        query = "UPDATE customer SET cust_city = %s WHERE cust_name = %s"
        cursor.execute(query, (new_city, old_name))
        conn.commit()
        print(f"Customer '{old_name}' updated successfully.")
        cursor.close()
    except Error as e:
        print(f"Failed to update customer: {e}")

def delete_customer(conn, cust_name):
    """ Delete a customer from the customer table """
    try:
        cursor = conn.cursor()
        query = "DELETE FROM customer WHERE cust_name = %s"
        cursor.execute(query, (cust_name,))
        conn.commit()
        print(f"Customer '{cust_name}' deleted successfully.")
        cursor.close()
    except Error as e:
        print(f"Failed to delete customer: {e}")

def navigate_customers(conn):
    """ Select and display all customers """
    try:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM customer")
        rows = cursor.fetchall()

        print("\n--- All Customers ---")
        for row in rows:
            print(row)
        print("----------------------")
        cursor.close()
    except Error as e:
        print(f"Failed to fetch customers: {e}")

# --- Main execution ---
if __name__ == '__main__':
    db_conn = create_connection()

    if db_conn:
        # 1. Add
        add_customer(db_conn, 'Gaurav', 'Kothrud', 'Pune')

        # 2. Navigate (Show all)
        navigate_customers(db_conn)

        # 3. Edit
        edit_customer(db_conn, 'Gaurav', 'Mumbai')

        # 4. Delete
        delete_customer(db_conn, 'Gaurav')
```

```
88
89          # 5. Navigate (Show all again)
90          navigate_customers(db_conn)
91
92          db_conn.close()
93          print("MySQL connection is closed")
```

Problem Statement 11

MongoDB operations for ICEM database with Teachers and Students collections.

Q1: Find the information about all teachers

```
1 db.Teachers.find();
```

Q2: Find the information about all teachers of computer department

```
1 db.Teachers.find({ dname: "computer" });
```

Q3: Find info about all teachers of computer, IT, and E&TC

```
1 db.Teachers.find({
2     dname: { $in: ["computer", "IT", "E&TC"] }
3 });
```

Q4: Same as Q3, with salary >= 10000

```
1 db.Teachers.find({
2     dname: { $in: ["computer", "IT", "E&TC"] },
3     salary: { $gte: 10000 }
4 });
```

Q5: Find student information having roll_no = 2 or Sname=xyz

```
1 db.Students.find({
2     $or: [
3         { roll_no: 2 },
4         { Sname: "xyz" }
5     ]
6 });
```

Q6: Update experience of teacher-praveen to 10 (upsert)

```
db.Teachers.updateOne(
    { Tname: "praveen" },
    { $set: { experience: 10 } },
    { upsert: true }
);
```

Q7: Update the department of all teachers in IT to COMP

```
db.Teachers.updateMany(
    { dname: "IT" },
    { $set: { dname: "COMP" } }
);
```

Q8: Find the teachers name and their experience

```
// Use projection: 1 means include, 0 means exclude
db.Teachers.find(
    {},
    { Tname: 1, experience: 1, _id: 0 }
);
```

Q9: Using Save() method insert one entry in department

```
// Note: save() is deprecated. insertOne() is preferred.
// But using save() as requested:
db.department.save({
    dname: "Mechanical",
    dno: 5,
    location: "B-Wing"
});
```

Problem Statement 12

MongoDB operations for zipcode collection.

Q.1: Return States with Populations above 1 Lakh

```
db.zipcode.aggregate([
    {
        $group: {
            _id: "$state",
            totalPop: { $sum: "$pop" }
        }
    },
    {
        $match: {
            totalPop: { $gt: 100000 }
        }
```

```
    }
]);
```

Q.2: Create a single-field index on the city field

```
db.zipcode.createIndex({ city: 1 });
```

Q.3: Create a compound index on the state and city fields

```
db.zipcode.createIndex({ state: 1, city: 1 });
```

Q.4: Find the total population for each state

```
db.zipcode.aggregate([
    {
        $group: {
            _id: "$state",
            totalPop: { $sum: "$pop" }
        }
    }
]);
```

Q.5: Find the average population per city within each state

```
// This finds the average population of *zipcodes* within each state
db.zipcode.aggregate([
    {
        $group: {
            _id: "$state",
            avgPopPerZipcode: { $avg: "$pop" }
        }
    }
]);

// A more complex query finding the average of *city totals* per state:
/*
db.zipcode.aggregate([
    { $group: { _id: { state: "$state", city: "$city" }, cityPop: { $sum: "$pop" } } },
    { $group: { _id: "$_id.state", avgCityPop: { $avg: "$cityPop" } } }
]);
*/
```

Q.6: List the states having total population greater than 20,000

```
db.zipcode.aggregate([
    {
        $group: {
            _id: "$state",
            totalPop: { $sum: "$pop" }
```

```
        }
    },
    {
        $match: {
            totalPop: { $gt: 20000 }
        }
    }
]);
```

Q.7: Retrieve all cities in California with population > 10,000

```
// This finds all *zipcodes* in CA with pop > 10k
db.zipcode.find({
    state: "CA",
    pop: { $gt: 10000 }
});
```

Problem Statement 13

Database 1

Schema:

- Supplier(Sid, Sname, address)
- Parts(Pid, Pname, Color)
- Catalog(sid, pid, cost)

Q.1: Find name of all parts whose color is green

```
SELECT Pname FROM Parts WHERE Color = 'green';
```

Q.2: Find names of suppliers who supply some red parts

```
SELECT DISTINCT s.Sname
FROM Supplier s
JOIN Catalog c ON s.Sid = c.sid
JOIN Parts p ON c.pid = p.Pid
WHERE p.Color = 'red';
```

Q.3: Find names of all parts whose cost is more than Rs25

```
SELECT DISTINCT p.Pname
FROM Parts p
JOIN Catalog c ON p.Pid = c.pid
WHERE c.cost > 25;
```

Database 2

Schema:

- Person(pname, street, city)
- Company(cname, city)
- Manages(pname, mname, salary)

Note: The schema is missing a link between Person and Company. Assuming Manages *should be* Works(pname, cname, salary).

Q.1: Find street/city of employees of "Idea", in Pune, > 3000

```sql
-- Assuming a corrected schema: Works(pname, cname, salary)
CREATE TABLE Works (
    pname VARCHAR(100),
    cname VARCHAR(100),
    salary NUMBER,
    FOREIGN KEY (pname) REFERENCES Person(pname),
    FOREIGN KEY (cname) REFERENCES Company(cname)
);

-- The query:
SELECT p.street, p.city
FROM Person p
JOIN Works w ON p.pname = w.pname
WHERE w.cname = 'Idea'
  AND p.city = 'Pune'
  AND w.salary > 3000;
```

Database 3

Schema:

- Student(Rollno, name, address)
- Subject(sub_code, sub_name)
- Marks(Rollno, sub_code, marks)

Q.1: Find average marks of each student with name

```sql
SELECT s.name, AVG(m.marks) AS average_marks
FROM Student s
JOIN Marks m ON s.Rollno = m.Rollno
GROUP BY s.Rollno, s.name;
```

Q.2: Find how many students failed in "DBMS"

```sql
-- Assuming 'failed' means marks < 40
SELECT COUNT(DISTINCT m.Rollno) AS failed_student_count
FROM Marks m
JOIN Subject s ON m.sub_code = s.sub_code
WHERE s.sub_name = 'DBMS'
  AND m.marks < 40;
```

Problem Statement 14

MongoDB MapReduce operation to calculate total quantity sold for each product in grocery_sales.

```javascript
// Step 1: Define the Map function
// It emits the product name as the key and quantity as the value
var mapFunction = function() {
    emit(this.product_name, this.quantity_sold);
};

// Step 2: Define the Reduce function
// It sums all quantities for a given key (product name)
var reduceFunction = function(keyProduct, valuesQuantities) {
    return Array.sum(valuesQuantities);
};

// Step 3: Run the MapReduce operation
db.grocery_sales.mapReduce(
    mapFunction,
    reduceFunction,
    {
        out: "product_sales_total" // Output collection
    }
);

// Step 4: Query the result
// db.product_sales_total.find();
```

Problem Statement 15

Database Connectivity: Program to implement MongoDB connectivity with a front-end language.

Answer: This is a conceptual task. Below is an example using **Python** with the pymongo library.

```python
import pymongo
from pymongo import MongoClient

def create_connection():
    """ Create a connection to a MongoDB server """
    try:
        # Connect to the MongoDB server (default port 27017)
        client = MongoClient("mongodb://localhost:27017/")
```

```
9
10          # Select the database (e.g., ICEM from P11)
11          db = client["ICEM"]
12
13          print("Connected to MongoDB database 'ICEM'")
14          return db
15      except pymongo.errors.ConnectionFailure as e:
16          print(f"Could not connect to MongoDB: {e}")
17          return None
18
19  def add_student(db, roll_no, name, s_class):
20      """ Add a new student to the Students collection """
21      try:
22          students_coll = db["Students"]
23          student_doc = {
24              "Sname": name,
25              "roll_no": roll_no,
26              "class": s_class
27          }
28          result = students_coll.insert_one(student_doc)
29          print(f"Student added with _id: {result.inserted_id}")
30      except Exception as e:
31          print(f"Failed to add student: {e}")
32
33  def edit_student(db, roll_no, new_class):
34      """ Edit an existing student's class """
35      try:
36          students_coll = db["Students"]
37          query = { "roll_no": roll_no }
38          new_values = { "$set": { "class": new_class } }
39
40          result = students_coll.update_one(query, new_values)
41          print(f"Students matched: {result.matched_count}, Modified: {result
        .modified_count}")
42      except Exception as e:
43          print(f"Failed to update student: {e}")
44
45  def delete_student(db, roll_no):
46      """ Delete a student from the Students collection """
47      try:
48          students_coll = db["Students"]
49          query = { "roll_no": roll_no }
50
51          result = students_coll.delete_one(query)
52          print(f"Student deleted. Count: {result.deleted_count}")
53      except Exception as e:
54          print(f"Failed to delete student: {e}")
55
56  def navigate_students(db):
57      """ Select and display all students """
58      try:
59          students_coll = db["Students"]
60          print("\n--- All Students ---")
61          for student in students_coll.find():
62              print(student)
63          print("----------------------")
64      except Exception as e:
65          print(f"Failed to fetch students: {e}")
```

```
66
67 # --- Main execution ---
68 if __name__ == '__main__':
69     db = create_connection()
70
71     if db:
72         # 1. Add
73         add_student(db, 10, "Ravi", "TE-Comp")
74
75         # 2. Navigate
76         navigate_students(db)
77
78         # 3. Edit
79         edit_student(db, 10, "BE-Comp")
80
81         # 4. Delete
82         delete_student(db, 10)
83
84         # 5. Navigate
85         navigate_students(db)
86
87         # Close connection (client is not returned from create_connection,
88         # but in a real app, you'd close client)
89         # client.close()
```

Problem Statement 16

MongoDB queries for student performance data using aggregation and indexing.

Q.1: Calculate average marks per department

```
1 db.students.aggregate([
2     {
3         $group: {
4             _id: "$department",
5             avgMarks: { $avg: "$marks" }
6         }
7     }
8 ]);
```

Q.2: Find the highest marks in each department

```
1 db.students.aggregate([
2     {
3         $group: {
4             _id: "$department",
5             highestMark: { $max: "$marks" }
6         }
7     }
8 ]);
```

Q.3: Find the sum of marks in each department

```
db.students.aggregate([
    {
        $group: {
            _id: "$department",
            totalMarks: { $sum: "$marks" }
        }
    }
]);
```

Q.4: Count the number of students from each city

```
db.students.aggregate([
    {
        $group: {
            _id: "$city",
            studentCount: { $sum: 1 } // or $count: {}
        }
    }
]);
```

Q.5: Calculate sum of marks per department in ascending order

```
db.students.aggregate([
    {
        $group: {
            _id: "$department",
            totalMarks: { $sum: "$marks" }
        }
    },
    {
        $sort: { totalMarks: 1 } // 1 for ascending
    }
]);
```

Q.6: Create single and composite index on the department field

```
// Single field index
db.students.createIndex({ department: 1 });

// Composite index (e.g., department and city)
db.students.createIndex({ department: 1, city: 1 });
```

Problem Statement 17

Student Management System (SQL DDL, DML, Set Operators, Subqueries).

Q.1: Create and Insert

```sql
-- Create tables
CREATE TABLE students (
    student_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    age INT,
    marks INT,
    city VARCHAR(50)
);

CREATE TABLE alumni (
    alumni_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    city VARCHAR(50)
);

-- Insert sample data
INSERT INTO students (name, age, marks, city) VALUES ('Aditya', 20, 85, 'Pune');
INSERT INTO students (name, age, marks, city) VALUES ('Priya', 21, 92, 'Mumbai');
INSERT INTO students (name, age, marks, city) VALUES ('Rohan', 19, 78, 'Pune');
INSERT INTO students (name, age, marks, city) VALUES ('Sneha', 22, 95, 'Delhi');
INSERT INTO students (name, age, marks, city) VALUES ('Vikram', 23, 88, 'Mumbai');

INSERT INTO alumni (name, city) VALUES ('Sneha', 'Delhi');
INSERT INTO alumni (name, city) VALUES ('Karan', 'Pune');
INSERT INTO alumni (name, city) VALUES ('Priya', 'Mumbai');
```

[cite$_s$*tart*]

Q.2: SELECT statements

```sql
-- Arithmetic operator (+, /)
SELECT name, marks, marks + 10 AS adjusted_marks FROM students;
SELECT name, marks, ROUND((marks / 100) * 100, 2) AS percentage FROM students;

-- Comparison operator (>, =)
SELECT name, marks FROM students WHERE marks > 90;

-- Logical operator (AND)
SELECT name, marks, city FROM students
WHERE marks > 80 AND city = 'Pune';

-- SQL Functions (AVG, MAX, UPPER, LENGTH)
SELECT AVG(marks) AS avg_marks, MAX(age) AS max_age FROM students;
SELECT name, UPPER(name) AS upper_name, LENGTH(name) AS name_length
FROM students;

-- Sorting and Filtering (ORDER BY, WHERE, GROUP BY)
SELECT city, COUNT(*) AS student_count
FROM students
```

```sql
WHERE marks > 80
GROUP BY city
ORDER BY student_count DESC;
```

Q.3: UPDATE command

```sql
-- Increase marks for students from Pune
UPDATE students
SET marks = marks + 2
WHERE city = 'Pune';
```

Q.4: DELETE command

```sql
-- Remove students with marks below 80
DELETE FROM students
WHERE marks < 80;
```

Q.5: SET OPERATORS

```sql
-- UNION (All unique names from both tables)
SELECT name FROM students
UNION
SELECT name FROM alumni;

-- INTERSECT (Names present in both tables)
SELECT DISTINCT s.name
FROM students s
WHERE EXISTS (SELECT 1 FROM alumni a WHERE a.name = s.name);

-- EXCEPT (Students who are NOT alumni)
SELECT DISTINCT s.name
FROM students s
WHERE NOT EXISTS (SELECT 1 FROM alumni a WHERE a.name = s.name);
```

Q.6: Subquery with IN

```sql
-- Find students who are also listed as alumni
SELECT name, city
FROM students
WHERE name IN (SELECT name FROM alumni);
```