DBMS LAB Practicals

How to Install Mysql and MongoDB By Command Prompt

- Mysql Installation:-
- sudo apt-get update
  sudo apt install mysql-server or sudo apt-get install mysql-server-5.
  sudo mysql_secure_installation
  sudo apt install mysql-client
  service mysql status
  mysqladmin -p -u root version
  sudo mysql –u root –p

- How to Run Mysql Through Terminal:-
  mysql -u root -p

- MongoDB Installation:-
  1.sudo apt install Mongodb-server
  2.sudo apt install Mongodb-clients

- How to Run MongoDB Through Terminal:-
  mongo
- Mongo DB Connectivity:-

Mongo DB Connectivity Steps:
1. Open Eclipse
2. File - New – Java Project –Give Project Name – ok
3. In project Explorer window- right click on project name new-class- give
Class name- ok
4. Download mongo-java-driver-2.12.2.jar
5. In project Explorer window- right click on project name Build path-Configure build path- Libraries- Add External Jar - MongoDB-Java-Driver
6. Start Mongo server before running the program

Study material provided by: Vishwajeet Londhe

Join Community by clicking below links

Telegram Channel

https://t.me/SPPU_TE_BE_COMP

(for all engineering Resources)

WhatsApp Channel

(for all tech updates)

https://whatsapp.com/channel/0029ValjFriICVfpcV9HFc3b

Insta Page

(for all engg & tech updates)

https://www.instagram.com/sppu_engineering_update

Assignment No.- 2

2(A)

Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym, different constraints etc.

Ans.

```
-- Show all available databases
SHOW DATABASES;

-- Create a new database named 'employee'
CREATE DATABASE employee;

-- Use the newly created database
USE employee;

-- Create the emp_details table with various constraints
CREATE TABLE emp_details (
    emp_no INT(10) PRIMARY KEY,                 -- Primary key constraint on employee number
    emp_name VARCHAR(30) NOT NULL,              -- NOT NULL constraint on employee name
    emp_gender CHAR(1) CHECK (emp_gender IN ('M', 'F')),  -- CHECK constraint for gender (must be 'M' or 'F')
    emp_sal INT(10) CHECK (emp_sal > 0),        -- CHECK constraint to ensure positive salary
    emp_dept VARCHAR(20)
);

-- Alter the table to add a new column with a CHECK constraint
ALTER TABLE emp_details
ADD COLUMN emp_age INT(3) CHECK (emp_age > 18);  -- CHECK constraint to ensure minimum age is 18

-- Verify table structure after modification
DESC emp_details;

-- Insert sample data into emp_details table
INSERT INTO emp_details (emp_no, emp_name, emp_gender, emp_sal, emp_dept)
VALUES (1, 'Ram', 'M', 300000, 'designing'),
       (2, 'Soham', 'M', 300000, 'designing'),
```

```
    (3, 'Mohan', 'M', 250000, 'management'),
    (4, 'Om', 'M', 400000, 'coding');

-- Retrieve all records from emp_details table
SELECT * FROM emp_details;

-- Create a duplicate table emp_info using AS SELECT to copy specific columns from emp_details
CREATE TABLE emp_info AS SELECT emp_no, emp_name, emp_gender FROM emp_details;

-- Verify records in emp_info table
SELECT * FROM emp_info;

-- Truncate the emp_info table, removing all data but keeping the structure
TRUNCATE TABLE emp_info;

-- Verify the table is empty
SELECT * FROM emp_info;

-- Drop the emp_info table entirely
DROP TABLE emp_info;

-- Create views to display specific records from emp_details
CREATE VIEW emp_view1 AS SELECT * FROM emp_details;                -- View to show all records in emp_details
CREATE VIEW emp_view2 AS SELECT * FROM emp_details WHERE emp_dept = 'designing';  -- View to show records where department is 'designing'

-- Select records from the views to verify their contents
SELECT * FROM emp_view1;
SELECT * FROM emp_view2;

-- Update a record in emp_details and check that it affects the view as well
UPDATE emp_details SET emp_dept = 'coding' WHERE emp_name = 'Mohan';
SELECT * FROM emp_details;

-- Drop the views once done
DROP VIEW emp_view1;
DROP VIEW emp_view2;

-- Create an index on emp_no and emp_name columns for faster searching
CREATE INDEX emp_ind ON emp_details(emp_no, emp_name);
```

```
-- Show the created index on the emp_details table
SHOW INDEX FROM emp_details;

-- Note:
-- MySQL does not support SEQUENCE or SYNONYM objects, which are available in other RDBMS like Oracle.
-- In MySQL, you can auto-increment fields to mimic sequences.
-- MySQL does not support synonyms, but we use aliases in SELECT statements for similar functionality.
```

Practical- 2(B)

Write at least 10 SQL queries on the suitable database application using SQL DML statements.

Ans.

```
-- Show available databases
SHOW DATABASES;

-- Create a new database named 'student'
CREATE DATABASE student;

-- Use the newly created 'student' database
USE student;

-- Create the 'stud_tab' table with appropriate fields
CREATE TABLE stud_tab (
    stud_id INT(4),
    stud_name VARCHAR(20),
    stud_dept VARCHAR(10),
    stud_dob DATE,
    stud_address VARCHAR(10)
);

-- Verify table structure
DESC stud_tab;

-- 1. Insert records into 'stud_tab'
INSERT INTO stud_tab VALUES (1, 'Ram', 'Comp', '2002-11-05', 'Pune');
INSERT INTO stud_tab VALUES (2, 'Soham', 'IT', '2002-09-03', 'Nashik');
INSERT INTO stud_tab VALUES (3, 'Ramesh', 'Comp', '2002-03-19', 'Pune');
INSERT INTO stud_tab VALUES (4, 'Mohan', 'AI&DS', '2002-02-22', 'Nagpur');

-- 2. Select all records from 'stud_tab'
SELECT * FROM stud_tab;

-- Alter the table to add a new column 'shift'
```

```
ALTER TABLE stud_tab ADD shift VARCHAR(10);

-- 3. Update specific records based on condition
UPDATE stud_tab SET shift = 'first' WHERE stud_id = 1;
UPDATE stud_tab SET shift = 'second' WHERE stud_id = 2;
UPDATE stud_tab SET shift = 'first' WHERE stud_id = 3;
UPDATE stud_tab SET shift = 'first' WHERE stud_id = 4;

-- 4. Select all records again to see the updated 'shift' values
SELECT * FROM stud_tab;

-- 5. Insert a new record including the 'shift' column
INSERT INTO stud_tab VALUES (5, 'Omkar', 'ENTC', '2002-06-26', 'Pune', 'second');

-- 6. Delete records where address is 'Nagpur'
DELETE FROM stud_tab WHERE stud_address = 'Nagpur';

-- 7. Update 'stud_id' for a specific student based on 'stud_name'
UPDATE stud_tab SET stud_id = 4 WHERE stud_name = 'Omkar';

-- 8. Select records where 'stud_dob' is within a specific range
SELECT * FROM stud_tab WHERE stud_dob BETWEEN '2002-01-01' AND '2002-07-01';

-- Alter the table to add a new column 'stud_fees'
ALTER TABLE stud_tab ADD stud_fees INT(15);

-- 9. Update 'stud_fees' values for various students
UPDATE stud_tab SET stud_fees = 15000 WHERE stud_id = 1;
UPDATE stud_tab SET stud_fees = 20000 WHERE stud_id = 2;
UPDATE stud_tab SET stud_fees = 20000 WHERE stud_id = 3;
UPDATE stud_tab SET stud_fees = 15000 WHERE stud_id = 4;

-- 10. Select all records again to see the updated 'stud_fees'
SELECT * FROM stud_tab;

-- 11. Select students with the highest 'stud_fees'
SELECT * FROM stud_tab WHERE stud_fees = (SELECT MAX(stud_fees) FROM
stud_tab);

-- 12. Calculate the sum of all 'stud_fees' values
SELECT SUM(stud_fees) FROM stud_tab;

-- 13. Create a new table 'stud_info' based on selected columns from 'stud_tab'
CREATE TABLE stud_info AS SELECT stud_id, stud_name FROM stud_tab;

-- 14. Use UNION to combine 'stud_id' from both tables and remove duplicates
SELECT stud_id FROM stud_tab UNION SELECT stud_id FROM stud_info;

-- 15. Group by 'stud_dept' and calculate the average 'stud_fees' for each department
SELECT stud_dept, AVG(stud_fees) AS avg_fees FROM stud_tab GROUP BY stud_dept;

-- 16. Count the number of students in each department
```

```
SELECT stud_dept, COUNT(stud_id) AS student_count FROM stud_tab GROUP BY
stud_dept;

-- 17. Order students by 'stud_fees' in descending order to see the highest paying students
first
SELECT * FROM stud_tab ORDER BY stud_fees DESC;
```

Practical No- 3

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

Ans.

```
-- Set up the database
SHOW DATABASES;
USE Abhi;

-- Show existing tables
SHOW TABLES;

-- Create the _master, customer, capital, and state tables
CREATE TABLE _master (
    product_no INT,
    description VARCHAR(20),
    profit_per FLOAT,
    unit_measure VARCHAR(10),
    quantity INT,
    reorder INT,
    sell_price FLOAT,
    cost_price FLOAT,
    PRIMARY KEY(product_no)
);

CREATE TABLE customer (
    cust_no INT,
    cust_name VARCHAR(20),
    cust_add VARCHAR(20),
    phone_no INT,
    PRIMARY KEY(cust_no)
);
```

```
CREATE TABLE capital (
    cap_no INT,
    cap_name VARCHAR(20),
    state_no INT,
    PRIMARY KEY(cap_no)
);

CREATE TABLE state (
    state_no INT,
    state_name VARCHAR(20),
    state_code INT,
    capital VARCHAR(20),
    PRIMARY KEY(state_no)
);

-- Insert sample data into the capital and state tables
INSERT INTO capital VALUES (1, 'MH', 1), (2, 'RAJ', 2), (3, 'GOA', 3), (4,
'GUJ', 4), (5, 'KAR', 5);
INSERT INTO state VALUES (1, 'MH', 1, 'MUM'), (2, 'RAJ', 2, 'JAI'), (3, 'GOA',
3, 'PAN'), (4, 'GUJ', 4, 'SUR'), (5, 'KAR', 5, 'BAN');

-- Display all rows from capital and state tables
SELECT * FROM capital;
SELECT * FROM state;

-- 1. INNER JOIN: Get matching rows between capital and state tables based
on cap_no and state_no
SELECT capital.cap_no, state.state_no
FROM capital
INNER JOIN state ON capital.cap_no = state.state_no;

-- 2. Update state_no in the state table to demonstrate updates
UPDATE state SET state_no = "78" WHERE state_no = 1;
UPDATE state SET state_no = "58" WHERE state_no = 2;
UPDATE state SET state_no = "46" WHERE state_no = 3;
UPDATE state SET state_no = "489" WHERE state_no = 4;
UPDATE state SET state_no = "458" WHERE state_no = 5;

-- Insert additional values to show more join results
INSERT INTO state VALUES (5, 'MP', 5, 'BHO');

-- 3. INNER JOIN: Check the updated join between capital and state
```

```
SELECT capital.cap_no, state.state_no
FROM capital
INNER JOIN state ON capital.cap_no = state.state_no;

-- 4. LEFT JOIN: Show all capitals, including those without a matching state
SELECT capital.cap_no, state.state_no
FROM capital
LEFT JOIN state ON capital.cap_no = state.state_no;

-- 5. LEFT JOIN (incorrect column join example): Try a different join to
showcase NULL results
SELECT capital.cap_no, state.state_no
FROM capital
LEFT JOIN state ON capital.cap_no = state.state_name;

-- 6. RIGHT JOIN: Show all states, including those without a matching capital
SELECT capital.cap_no, state.state_no
FROM capital
RIGHT JOIN state ON capital.cap_no = state.state_no;

-- Additional retrieval of all data in capital and state tables
SELECT * FROM capital;
SELECT * FROM state;

-- 7. INNER JOIN with multiple columns: Retrieve details from both tables
based on matching state_no
SELECT capital.cap_no, capital.cap_name, state.capital, state.state_no
FROM capital
INNER JOIN state ON capital.cap_no = state.state_no;

-- 8. LEFT JOIN: Show all capitals with their state details, if available
SELECT capital.cap_no, capital.cap_name, state.capital, state.state_no
FROM capital
LEFT JOIN state ON capital.cap_no = state.state_no;

-- 9. RIGHT JOIN with UNION: Combine LEFT and RIGHT JOIN results to
simulate a FULL OUTER JOIN
SELECT capital.cap_no, capital.cap_name, state.capital, state.state_no
FROM capital
LEFT JOIN state ON capital.cap_no = state.state_no
UNION
SELECT capital.cap_no, capital.cap_name, state.capital, state.state_no
```

```
FROM capital
RIGHT JOIN state ON capital.cap_no = state.state_no;

-- 10. CROSS JOIN: Display all possible combinations of capital and state records
SELECT * FROM capital c1, state s1;

-- 11. CROSS JOIN with a condition: Show pairs with different cap_no and state_no
SELECT * FROM capital c1, state s1 WHERE c1.cap_no != s1.state_no;

-- 12. Sub-query: Find states with the same state_no as 'MH'
SELECT * FROM state WHERE state_no = (SELECT state_no FROM state WHERE state_name = 'MH');

-- 13. Sub-query: Retrieve details for the state 'GUJ'
SELECT * FROM state WHERE state_no = (SELECT state_no FROM state WHERE state_name = 'GUJ');

-- 14. Sub-query using capital: Find states by matching capital with a capital's state_no
SELECT * FROM state WHERE state_no = (SELECT state_no FROM capital WHERE cap_name = 'MH');
SELECT * FROM state WHERE state_no = (SELECT state_no FROM capital WHERE cap_name = 'GUJ');
SELECT * FROM state WHERE state_no = (SELECT state_no FROM capital WHERE cap_name = 'RAJ');
SELECT * FROM state WHERE state_no = (SELECT state_no FROM capital WHERE cap_name = 'KAR');
```

Practical 4-

Write a PL/SQL code block to calculate fine for a library book by accessing borrower information from the database.

Ans.

```
-- Use the database
USE Abhi;

-- Set the delimiter to allow for multi-line procedures
DELIMITER //

-- Create the stored procedure `B` to calculate fines based on overdue days
CREATE PROCEDURE B (roll_new INT, book_name VARCHAR(20))
BEGIN
    DECLARE X INT;  -- Variable to hold the number of overdue days
    DECLARE CONTINUE HANDLER FOR NOT FOUND BEGIN
        SELECT 'Borrower Not Found' AS Message;
    END;

    -- Calculate the number of days overdue
    SELECT DATEDIFF(CURDATE(), DOI) INTO X
    FROM Borrower
    WHERE roll_no = roll_new AND book_name = book_name;

    -- Check the overdue days and insert fine records accordingly
    IF (X > 15 AND X < 30) THEN
        INSERT INTO Fine (roll_no, fine_date, fine_amount)
        VALUES (roll_new, CURDATE(), X * 5);  -- Fine of 5 per day for overdue days
between 15 and 30
    ELSEIF (X >= 30) THEN
        INSERT INTO Fine (roll_no, fine_date, fine_amount)
        VALUES (roll_new, CURDATE(), X * 50);  -- Fine of 50 per day for overdue days
beyond 30
    END IF;

    -- Update the status of the book to 'returned'
    UPDATE Borrower
    SET status = 'returned'
    WHERE roll_no = roll_new AND book_name = book_name;
END //

-- Reset the delimiter back to semicolon
DELIMITER ;

-- Example calls to the stored procedure
CALL B(1, 'TOC');
CALL B(12, 'xyz');
CALL B(20, 'patil');

-- Verify data in Fine and Borrower tables
```

```
SELECT * FROM Fine;
SELECT * FROM Borrower;
```

Practical - 5:

Write a PL/SQL code block to study and implement stored procedure and function for displaying the result of student based on the grade obtained.

Ans.

```
-- Create and use the database
CREATE DATABASE Score;
USE Score;

-- Create tables to store student marks and results
CREATE TABLE stud_marks (
    name VARCHAR(20),
    total_marks INT(5)
);

CREATE TABLE Result (
    roll_no INT(3) PRIMARY KEY,
    name VARCHAR(20),
    class VARCHAR(20)
);

-- Insert sample data into stud_marks
INSERT INTO stud_marks VALUES ('Suresh', 995);
INSERT INTO stud_marks VALUES ('Harish', 865);
INSERT INTO stud_marks VALUES ('Samart', 920);
INSERT INTO stud_marks VALUES ('Mohan', 1000);
INSERT INTO stud_marks VALUES ('Soham', 745);

-- Display the data in stud_marks
SELECT * FROM stud_marks;

-- Insert corresponding results into Result table
INSERT INTO Result (roll_no, name) VALUES (1, 'Suresh');
INSERT INTO Result (roll_no, name) VALUES (2, 'Harish');
INSERT INTO Result (roll_no, name) VALUES (3, 'Samart');
INSERT INTO Result (roll_no, name) VALUES (4, 'Mohan');
INSERT INTO Result (roll_no, name) VALUES (5, 'Soham');

-- Display the data in Result table
SELECT * FROM Result;

-- Set the delimiter for the procedure creation
DELIMITER //
```

```
-- Create the stored procedure to determine the grade
CREATE PROCEDURE proc_Grade(IN r INT, OUT grade CHAR(25))
BEGIN
    DECLARE m INT;

    -- Fetch total marks based on roll number
    SELECT total_marks INTO m
    FROM stud_marks
    WHERE name = (SELECT name FROM Result WHERE roll_no = r);

    -- Determine the grade based on total marks
    IF m >= 990 AND m <= 1500 THEN
        SET grade = 'Distinction';
        UPDATE Result SET class = 'Distinction' WHERE roll_no = r;
    ELSEIF m >= 900 AND m <= 989 THEN
        SET grade = 'FirstClass';
        UPDATE Result SET class = 'FirstClass' WHERE roll_no = r;
    ELSEIF m >= 825 AND m <= 899 THEN
        SET grade = 'SecondClass';
        UPDATE Result SET class = 'SecondClass' WHERE roll_no = r;
    ELSE
        SET grade = '--';
        UPDATE Result SET class = '--' WHERE roll_no = r;
    END IF;
END //

-- Reset the delimiter
DELIMITER ;

-- Create a function that utilizes the procedure to return the grade
DELIMITER //
CREATE FUNCTION func_Grade(r INT)
RETURNS VARCHAR(25)
DETERMINISTIC
BEGIN
    DECLARE grade VARCHAR(25);
    CALL proc_Grade(r, grade);
    RETURN grade;
END //
DELIMITER ;

-- Test the function for each student
SELECT func_Grade(1) AS Grade_1;
SELECT func_Grade(2) AS Grade_2;
SELECT func_Grade(3) AS Grade_3;
SELECT func_Grade(4) AS Grade_4;
SELECT func_Grade(5) AS Grade_5;

-- Display the final results
SELECT * FROM Result;
```

Practical 6-

Write a PL/SQL code block to create a parameterized cursor for copying contents of one table into another by avoiding redundancy.

Ans.

```
-- Create and use the database
CREATE DATABASE class;
USE class;

-- Create the tables for roll call
CREATE TABLE O_RollCall (
    roll_no INT(3),
    name VARCHAR(20)
);

CREATE TABLE N_RollCall (
    roll_no INT(3),
    name VARCHAR(20)
);

-- Insert sample data into O_RollCall
INSERT INTO O_RollCall VALUES (1, 'Himanshu');
INSERT INTO O_RollCall VALUES (2, 'Ram');
INSERT INTO O_RollCall VALUES (3, 'Soham');
INSERT INTO O_RollCall VALUES (5, 'Mohan');
INSERT INTO O_RollCall VALUES (6, 'Om');
INSERT INTO O_RollCall VALUES (9, 'Yash');
INSERT INTO O_RollCall VALUES (11, 'Mayur');

-- Display the data in O_RollCall
SELECT * FROM O_RollCall;

-- Display the data in N_RollCall
SELECT * FROM N_RollCall;

-- Set the delimiter for procedure creation
DELIMITER //

-- Create a procedure to copy unique entries from O_RollCall to N_RollCall
CREATE PROCEDURE cursor_proc_p1()
BEGIN
    DECLARE fin INT DEFAULT 0;
    DECLARE old_roll INT;
    DECLARE old_name VARCHAR(20);

    -- Declare a cursor for selecting records from O_RollCall
    DECLARE old_csr CURSOR FOR SELECT roll_no, name FROM O_RollCall;
    -- Declare a handler for not found condition
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET fin = 1;
```

```
    -- Open the cursor
    OPEN old_csr;

    -- Start the loop to fetch and insert records
    read_loop: LOOP
        FETCH old_csr INTO old_roll, old_name;

        -- Check if no more rows are found
        IF fin = 1 THEN
            LEAVE read_loop;
        END IF;

        -- Check for redundancy before inserting
        IF NOT EXISTS (SELECT * FROM N_RollCall WHERE roll_no = old_roll) THEN
            INSERT INTO N_RollCall (roll_no, name) VALUES (old_roll, old_name);
        END IF;
    END LOOP;

    -- Close the cursor
    CLOSE old_csr;
END //

-- Create a parameterized procedure to copy entries from O_RollCall to N_RollCall based
on roll_no
CREATE PROCEDURE cursor_proc_p2(IN r1 INT)
BEGIN
    DECLARE r2 INT;
    DECLARE exit_loop BOOLEAN DEFAULT FALSE;

    -- Declare a cursor to select roll_no greater than r1
    DECLARE c1 CURSOR FOR SELECT roll_no FROM O_RollCall WHERE roll_no > r1;
    -- Declare a handler for not found condition
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET exit_loop = TRUE;

    -- Open the cursor
    OPEN c1;

    -- Start the loop to fetch and insert records
    e_loop: LOOP
        FETCH c1 INTO r2;

        -- Check if no more rows are found
        IF exit_loop THEN
            LEAVE e_loop;
        END IF;

        -- Check for redundancy before inserting
        IF NOT EXISTS (SELECT * FROM N_RollCall WHERE roll_no = r2) THEN
            INSERT INTO N_RollCall SELECT * FROM O_RollCall WHERE roll_no = r2;
        END IF;
    END LOOP e_loop;
```

```
    -- Close the cursor
    CLOSE c1;
END //

-- Call procedures to test the functionality
CALL cursor_proc_p1();
SELECT * FROM O_RollCall;
SELECT * FROM N_RollCall;

CALL cursor_proc_p2(5);
SELECT * FROM N_RollCall;  -- Check results after calling cursor_proc_p2(5)

CALL cursor_proc_p2(3);
SELECT * FROM N_RollCall;  -- Check results after calling cursor_proc_p2(3)
```

Practical 7-

Write a PL/SQL code block to create triggers on the Library table to keep track of updating and deletion of records.

Ans.

```
-- Create and use the database
CREATE DATABASE lib;
USE lib;

-- Create the library table
CREATE TABLE library (
    bno INT(5),
    bname VARCHAR(40),
    author VARCHAR(20),
    allowed_days INT(5)
);

-- Create the audit table to track changes
CREATE TABLE library_audit (
    bno INT(5),
    old_allowed_days INT(5),
    new_allowed_days INT(5),
    change_timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Insert sample data into the library
INSERT INTO library VALUES (1, 'Database Systems', 'Connally T', 10);
INSERT INTO library VALUES (2, 'System Programming', 'John Donovan', 20);
INSERT INTO library VALUES (3, 'Computer Network & Internet', 'Douglas E. Comer', 18);
INSERT INTO library VALUES (4, 'Agile Project Management', 'Ken Schwaber', 24);
```

```
INSERT INTO library VALUES (5, 'Python for Data Analysis', 'Wes McKinney', 12);

-- Display the data in the library
SELECT * FROM library;

-- Set the delimiter for trigger creation
DELIMITER //

-- Create a trigger to audit updates
CREATE TRIGGER tr_update_library_audit
BEFORE UPDATE ON library
FOR EACH ROW
BEGIN
    INSERT INTO library_audit (bno, old_allowed_days, new_allowed_days)
    VALUES (OLD.bno, OLD.allowed_days, NEW.allowed_days);
END //

-- Create a trigger to audit deletions
CREATE TRIGGER tr_delete_library_audit
BEFORE DELETE ON library
FOR EACH ROW
BEGIN
    INSERT INTO library_audit (bno, old_allowed_days, new_allowed_days)
    VALUES (OLD.bno, OLD.allowed_days, NULL);
END //

DELIMITER ;

-- Perform updates on the library table
UPDATE library SET allowed_days = 15 WHERE bno = 1;
UPDATE library SET allowed_days = 25 WHERE bno = 2;
UPDATE library SET allowed_days = 13 WHERE bno = 3;
UPDATE library SET allowed_days = 19 WHERE bno = 4;
UPDATE library SET allowed_days = 17 WHERE bno = 5;

-- Display the updated library table
SELECT * FROM library;

-- Display the audit log
SELECT * FROM library_audit;
```

Practical no-8

Database Connectivity:
Write a program to implement MySQL/Oracle database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc.)

Ans.

```
package MySQLConnectivity;

import java.sql.*;
import java.util.Scanner;

public class MySQL {

    public static void main(String[] args) {
        // Declare variables
        int n, sno;
        String name, telephone, gender;

        // Initialize Scanner for user input
        Scanner in = new Scanner(System.in);

        // Database connection parameters
        String url = "jdbc:mysql://localhost:3306/your_database"; // Replace with your
database name
        String user = "your_username"; // Replace with your MySQL username
        String password = "your_password"; // Replace with your MySQL password

        try (Connection con = DriverManager.getConnection(url, user, password);
            Statement stmt = con.createStatement();
            PreparedStatement pstmInsert = con.prepareStatement("INSERT INTO Personal
(sno, name, telephone, gender) VALUES (?, ?, ?, ?)");
            PreparedStatement pstmDelete = con.prepareStatement("DELETE FROM
Personal WHERE sno = ?");
            PreparedStatement pstmUpdate = con.prepareStatement("UPDATE Personal SET
name = ?, telephone = ?, gender = ? WHERE sno = ?")) {

            // Insertion
            System.out.print("Enter the number of records you want to insert: ");
            n = in.nextInt();
            for (int i = 0; i < n; i++) {
                System.out.print("\nData" + (i + 1) + "\nEnter Sno: ");
                sno = in.nextInt();
                pstmInsert.setInt(1, sno);
                System.out.print("Enter Name: ");
                name = in.next();
                pstmInsert.setString(2, name);
```

```
            System.out.print("Enter Telephone: ");
            telephone = in.next();
            pstmInsert.setString(3, telephone);
            System.out.print("Enter Gender: ");
            gender = in.next();
            pstmInsert.setString(4, gender);
            pstmInsert.executeUpdate();
        }

        // Display after insertion
        displayRecords(stmt, "After Insertion");

        // Search operation
        System.out.print("Enter Sno to search: ");
        sno = in.nextInt();
        searchRecord(stmt, sno);

        // Update operation
        System.out.print("Enter Sno to update: ");
        sno = in.nextInt();
        System.out.print("Enter new Name: ");
        name = in.next();
        System.out.print("Enter new Telephone: ");
        telephone = in.next();
        System.out.print("Enter new Gender: ");
        gender = in.next();
        pstmUpdate.setString(1, name);
        pstmUpdate.setString(2, telephone);
        pstmUpdate.setString(3, gender);
        pstmUpdate.setInt(4, sno);
        pstmUpdate.executeUpdate();

        // Display after update
        displayRecords(stmt, "After Update");

        // Deletion
        System.out.print("Enter the number of records you want to delete: ");
        n = in.nextInt();
        for (int i = 0; i < n; i++) {
            System.out.print("\nData" + (i + 1) + "\nEnter Sno: ");
            sno = in.nextInt();
            pstmDelete.setInt(1, sno);
            pstmDelete.executeUpdate();
        }

        // Display after deletion
        displayRecords(stmt, "After Deletion");

    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        in.close();
```

```
        }
    }

    // Method to display records
    private static void displayRecords(Statement stmt, String message) throws
SQLException {
        ResultSet rs = stmt.executeQuery("SELECT * FROM Personal;");
        System.out.println("\n" + message);
        System.out.println("Sno\tName\tTelephone\tGender");
        while (rs.next()) {
            System.out.println(rs.getInt(1) + "\t" + rs.getString(2) + "\t" + rs.getString(3) + "\t" +
rs.getString(4));
        }
    }

    // Method to search for a record
    private static void searchRecord(Statement stmt, int sno) throws SQLException {
        ResultSet rsSearch = stmt.executeQuery("SELECT * FROM Personal WHERE sno =
" + sno + ";");
        System.out.println("\nSearch Result");
        System.out.println("Sno\tName\tTelephone\tGender");
        if (rsSearch.next()) {
            System.out.println(rsSearch.getInt(1) + "\t" + rsSearch.getString(2) + "\t" +
rsSearch.getString(3) + "\t" + rsSearch.getString(4));
        } else {
            System.out.println("No record found with Sno: " + sno);
        }
    }
}
```

GROUP B: NO SQL Databases
Practical No- 9

MongoDB Queries:
Design and Develop MongoDB Queries using CRUD operations. (Use
CRUD operations, SAVE method, logical operators etc.)

Ans.

```
// Show available databases
show dbs

// Use the database named 'book'
use book

// Show collections in the database
show collections;

// Create a collection named 'library'
db.createCollection("library");
```

```
// Insert documents into the 'library' collection
db.library.insert({"bid":1,"name":"C++"});
db.library.insert({"bid":2,"name":"SEPM","author":"Pressman"});
db.library.insert({"bid":3,"name":"CN","author":"Forouzan","cost":700});

// Find and pretty-print all documents
db.library.find().pretty();

// Remove a document by 'bid'
db.library.remove({"bid":1});

// Count documents in the collection
db.library.countDocuments(); // Prefer countDocuments() over count()

// Find all documents again
db.library.find().pretty();

// Insert a document again
db.library.insert({"bid":1,"name":"C++"});

// Pretty-print documents after insertion
db.library.find().pretty();

// Sort documents by 'bid'
db.library.find().sort({"bid":1});

// Insert more documents
db.library.insert({"bid":4,"name":"SPOS","author":"Pearson","cost":500});

// Find documents sorted by 'bid'
db.library.find().pretty();
db.library.find().sort({"bid":1});

// Find documents with both 'name' and 'cost' conditions
db.library.find({$and:[{"name":"CN"},{"cost":700}]}).pretty();

// Insert additional documents
db.library.insert({"bid":5,"name":"TOC","author":"Addison-Wesley","cost":600});
db.library.insert({"bid":6,"name":"AI","author":"McGraw Hill Education","cost":800});

// Find all documents again
db.library.find().pretty();

// Find documents with cost of 500 or 800
db.library.find({$or:[{"cost":500},{"cost":800}]}).pretty();

// Find documents where cost is not equal to 500
db.library.find({"cost":{$ne:500}});

// Find documents using $nor operator
db.library.find({$nor:[{"cost":500},{"author":"Forouzan"}]});
```

```
// Find documents using $not operator
db.library.find({"cost":{$not:{$gt:800}}});

// Insert another document
db.library.insert({"bid":7,"name":"CC","author":"Wiley Publications","cost":400});

// Display all documents
db.library.find();

// Update documents with specific cost values
db.library.updateMany({'cost':400}, {$set: {'cost':600}}); // Using updateMany to update all matching documents
db.library.updateMany({'cost':800}, {$set: {'cost':1200}});

// Find and pretty-print all documents after updates
db.library.find().pretty();

// Example of using the save() method (to update or insert)
db.library.save({"bid":3, "name":"Updated CN", "author":"Forouzan", "cost":700}); // This will update if 'bid' exists or insert if it doesn't
```

Practical No- 10

MongoDB - Aggregation and Indexing:
Design and Develop MongoDB Queries using aggregation and indexing with suitable example using MongoDB.

Ans.

```
// Show available databases
show dbs

// Use the database named 'customer'
use customer

// Insert documents into the 'cust_table' collection
db.cust_table.insert({Item_id: 1, Cust_Name: "Ram", Product: "Milk", Amount: 40});
db.cust_table.insert({Item_id: 2, Cust_Name: "Ram", Product: "Parle_G", Amount: 50});
db.cust_table.insert({Item_id: 3, Cust_Name: "Mohan", Product: "Lays Chips", Amount: 40});
db.cust_table.insert({Item_id: 4, Cust_Name: "Shivam", Product: "Mentos", Amount: 10});
db.cust_table.insert({Item_id: 5, Cust_Name: "Mohan", Product: "Maggie", Amount: 60});

// Aggregation queries to calculate totals by customer name
db.cust_table.aggregate([
   { $group: { _id: "$Cust_Name", total: { $sum: "$Amount" } } }
]);
db.cust_table.aggregate([
   { $group: { _id: "$Cust_Name", total: { $avg: "$Amount" } } }
```

```
]);
db.cust_table.aggregate([
    { $group: { _id: "$Cust_Name", total: { $min: "$Amount" } } }
]);
db.cust_table.aggregate([
    { $group: { _id: "$Cust_Name", total: { $max: "$Amount" } } }
]);
db.cust_table.aggregate([
    { $group: { _id: "$Cust_Name", total: { $first: "$Amount" } } }
]);
db.cust_table.aggregate([
    { $group: { _id: "$Cust_Name", total: { $last: "$Amount" } } }
]);
// The $push operator should be spelled correctly
db.cust_table.aggregate([
    { $group: { _id: "$Cust_Name", total: { $push: "$Amount" } } }
]);
db.cust_table.aggregate([
    { $group: { _id: "$Cust_Name", total: { $sum: 1 } } } // This counts the number of
documents per customer
]);
db.cust_table.aggregate([
    { $group: { _id: "$Cust_Name", total: { $addToSet: "$Amount" } } }
]);

// Create an index on the 'Item_id' field (index values should be 1 for ascending or -1 for
descending)
db.cust_table.createIndex({'Item_id': 1});

// You should only create a unique index or another index if necessary, using valid field
values.
// Create another index if needed (for example on 'Cust_Name')
db.cust_table.createIndex({'Cust_Name': 1});

// Get the list of indexes on 'cust_table'
db.cust_table.getIndexes();

// If you want to drop an index, you should specify the name, not the field itself.
// Example of dropping the index
db.cust_table.dropIndex("Item_id_1"); // Use the index name shown in getIndexes()

// Check the remaining indexes
db.cust_table.getIndexes();
```

Practical No- 11

MongoDB - Map reduces operations:
Implement Map reduces operation with suitable example using MongoDB.

Ans.

```
show dbs
use bill
db.pay.insert({Cust_ID:"A123",Product:"Milk",Amount:40,Status:"P"});
db.pay.insert({Cust_ID:"A123",Product:"Parle_G",Amount:50,Status:"NP"});
db.pay.insert({Cust_ID:"A123",Product:"Lays Chips",Amount:40,Status:"P"});
db.pay.insert({Cust_ID:"B123",Product:"Mentos",Amount:10,Status:"P"});
db.pay.insert({Cust_ID:"B123",Product:"Maggie",Amount:60,Status:"NP"});
db.pay.find() // Optional, to see inserted documents

// MapReduce for non-paid statuses
db.pay.mapReduce(
    function() { emit(this.Cust_ID, this.Amount); },
    function(key, values) { return Array.sum(values); },
    { query: { "Status": "NP" }, out: "Bill_Amount" }
);

// MapReduce for all statuses
var mapFunc1 = function() { emit(this.Cust_ID, this.Amount); };
var reduceFunc1 = function(keyCustID, valuePrices) { return Array.sum(valuePrices); };
db.pay.mapReduce(mapFunc1, reduceFunc1, { out: "Map" });

// Display the results
db.Bill_Amount.find().pretty();
db.Map.find().pretty();
```

Practical No -12

Database Connectivity:
Write a program to implement MongoDB database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc.)

Ans

```
package MongoDB;

import java.util.Scanner;
import com.mongodb.client.*;
import org.bson.Document;

public class MongoDB {
    public static void displayData(MongoCollection<Document> collection, String field) {
        for (Document doc : collection.find()) {
            System.out.print(field + ": " + doc.get(field) + "\t");
```

```
    }
    System.out.println();
}

public static void main(String[] args) {
    MongoClient mongoClient = MongoClients.create("mongodb://localhost:27017");
    System.out.println("Connected to the database successfully");

    Scanner sc = new Scanner(System.in);
    String ans = "y";

    while (ans.equalsIgnoreCase("y")) {
        MongoDatabase database = mongoClient.getDatabase("Info");
        MongoCollection<Document> collection = database.getCollection("Personal");

        System.out.print("Enter the number of records you want to insert: ");
        int n = sc.nextInt();
        sc.nextLine(); // Consume newline

        for (int i = 0; i < n; i++) {
            Document info = new Document();
            System.out.println("Enter Data " + (i + 1));
            System.out.print("Enter name: ");
            String name = sc.nextLine();
            info.append("Name", name);

            System.out.print("Enter age: ");
            int age = sc.nextInt();
            info.append("Age", age);

            System.out.print("Enter Mobile Number: ");
            String mobileNo = sc.next();
            info.append("Mobile Number", mobileNo);
            sc.nextLine(); // Consume newline

            collection.insertOne(info);
        }

        System.out.println("Insert Operation");
        displayData(collection, "Name");
        displayData(collection, "Mobile Number");
        displayData(collection, "Age");

        System.out.print("Enter the number of records you want to delete: ");
        n = sc.nextInt();
        sc.nextLine(); // Consume newline

        for (int i = 0; i < n; i++) {
            System.out.print("Enter name of the record to delete: ");
            String nameToDelete = sc.nextLine();
            collection.deleteOne(new Document("Name", nameToDelete));
        }

        System.out.println("Delete Operation");
        displayData(collection, "Name");
        displayData(collection, "Mobile Number");
        displayData(collection, "Age");

        System.out.print("Do you want to drop the database? (y/n): ");
        ans = sc.nextLine();
```

```
            if (ans.equalsIgnoreCase("y")) {
                mongoClient.getDatabase("Info").drop();
                System.out.println("Database Dropped");
            }

            System.out.print("Do you want to continue? (y/n): ");
            ans = sc.nextLine();
        }

        sc.close();
        mongoClient.close();
    }
}
```