SRM Institute of Science and Technology
Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204
Department of Computer Applications
Circular – 2023-24
BCA DS 3rd Sem

Data Base Management System (UDS23302J)

Lab Manual

Lab 1: Design a Database and Create Required Tables

Aim: To design a database schema and implement it by creating tables using SQL Data Definition Language (DDL) statements.

Procedure:

1. Identify the entities, attributes, and relationships for the given problem.
2. Draw an Entity-Relationship (ER) diagram.
3. Map the ER diagram to a relational schema.
4. Use SQL CREATE TABLE statements to define the tables, specifying data types for each attribute.

Source Code:

```
-- Create a database (if it doesn't exist)
CREATE DATABASE IF NOT EXISTS YourDatabaseName;

-- Use the database
USE YourDatabaseName;

-- Create a table (example: Students)
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DateOfBirth DATE,
    Major VARCHAR(50)
);

-- Create another table (example: Courses)
CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(100),
    Credits INT
);

-- Create a table to represent the relationship between students and courses
CREATE TABLE StudentCourses (
    StudentID INT,
    CourseID INT,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID),
    PRIMARY KEY (StudentID, CourseID) -- Composite primary key
);
```

Input:

Problem statement for which the database needs to be designed (e.g., "Design a database for a university to manage student records and course enrollments").

Expected Output:

A database schema implemented in a database management system (e.g., MySQL, PostgreSQL) with the specified tables and columns.

Lab 2: SQL Data Definition Language (DDL) using Student Database

Aim: To practice using SQL DDL commands to define and modify the structure of a database and its tables.

Procedure:

1. Create a database named "StudentDB".
2. Create tables such as "Student," "Course," and "Enrollment" with appropriate attributes.
3. Use ALTER TABLE to add, modify, or delete columns.
4. Use DROP TABLE to delete tables.

Source Code:

```
-- Create the StudentDB database
CREATE DATABASE StudentDB;

-- Use the StudentDB database
USE StudentDB;

-- Create the Student table
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(100),
    Major VARCHAR(50)
);

-- Create the Course table
CREATE TABLE Course (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(100),
    Credits INT
);

-- Create the Enrollment table
CREATE TABLE Enrollment (
    StudentID INT,
    CourseID INT,
    EnrollmentDate DATE,
    PRIMARY KEY (StudentID, CourseID),
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);

-- Alter the Student table to add an Email column
ALTER TABLE Student ADD Email VARCHAR(100);

-- Alter the Student table to modify the Major column
ALTER TABLE Student MODIFY Major VARCHAR(100);

-- Alter the Student table to drop the Major column
ALTER TABLE Student DROP COLUMN Major;

-- Drop the Enrollment Table
DROP TABLE Enrollment;
```

Input:

5. Specifications for the Student, Course, and Enrollment tables (attributes and data types).

Expected Output:

6. The StudentDB database with the created tables, modified as per the ALTER TABLE commands.

Lab 3: Apply Constraints like Primary Key, Foreign Key, NOT NULL to the Tables

Aim: To understand and apply various constraints to enforce data integrity in a database.

Procedure:

1. Create tables with primary key constraints to uniquely identify each record.
2. Implement foreign key constraints to establish relationships between tables.
3. Use the NOT NULL constraint to ensure that certain columns cannot have null values.
4. Use UNIQUE constraint to ensure that values in a column are unique.
5. Use CHECK constraint to ensure that values in a column satisfy a specific condition.

Source Code:

```
-- Create a database
CREATE DATABASE ConstraintsDemo;
USE ConstraintsDemo;

-- Create a table with constraints
CREATE TABLE Departments (
    DeptID INT PRIMARY KEY,
    DeptName VARCHAR(50) NOT NULL,
    Location VARCHAR(50) UNIQUE
);

CREATE TABLE Employees (
    EmpID INT PRIMARY KEY,
    EmpName VARCHAR(100) NOT NULL,
    DeptID INT,
    Salary DECIMAL(10, 2) CHECK (Salary > 0),
    FOREIGN KEY (DeptID) REFERENCES Departments(DeptID)
);
```

Input:

6. Table structures and the constraints to be applied to each table.

Expected Output:

7. Tables created with the specified constraints enforced. The database should prevent operations that violate these constraints.

Lab 4: SQL Data Manipulation Language (DML) Commands using Employee Database

Aim: To practice using SQL DML commands to manipulate data within a database.

Procedure:

1. Create an "Employee" table with attributes like EmpID, EmpName, DeptID, and Salary.
2. Use the INSERT statement to add new employee records.
3. Use the SELECT statement to retrieve employee data.
4. Use the UPDATE statement to modify employee information.
5. Use the DELETE statement to remove employee records.

Source Code:

```
-- Create a database
CREATE DATABASE EmployeeDB;
USE EmployeeDB;

-- Create the Employee table
CREATE TABLE Employee (
    EmpID INT PRIMARY KEY,
    EmpName VARCHAR(100),
    DeptID INT,
    Salary DECIMAL(10, 2)
);

-- Insert data into the Employee table
INSERT INTO Employee (EmpID, EmpName, DeptID, Salary) VALUES
(101, 'John Doe', 1, 50000.00),
(102, 'Jane Smith', 2, 60000.00),
(103, 'Bob Johnson', 1, 55000.00);

-- Select all employees
SELECT * FROM Employee;

-- Select employees in department 1
SELECT * FROM Employee WHERE DeptID = 1;

-- Update John Doe's salary
UPDATE Employee SET Salary = 52000.00 WHERE EmpID = 101;

-- Delete Bob Johnson
DELETE FROM Employee WHERE EmpID = 103;
```

Input:

6. Employee data to be inserted, updated, or deleted.
7. Queries to retrieve specific employee information.

Expected Output:

8. The Employee table with the manipulated data, and the results of the SELECT queries.

Lab 5: SQL Data Control Commands

Aim: To understand and use SQL Data Control Language (DCL) commands to manage user privileges and access control in a database.

Procedure:

1. Create users using the CREATE USER statement.
2. Grant privileges to users using the GRANT statement (e.g., SELECT, INSERT, UPDATE, DELETE).
3. Revoke privileges from users using the REVOKE statement.

Source Code:

```
-- Create a user
CREATE USER 'user1'@'localhost' IDENTIFIED BY 'password123';

-- Grant SELECT privilege on all tables in EmployeeDB
GRANT SELECT ON EmployeeDB.* TO 'user1'@'localhost';

-- Grant INSERT and UPDATE privileges on the Employee table
GRANT INSERT, UPDATE ON EmployeeDB.Employee TO 'user1'@'localhost';

-- Revoke INSERT privilege from user1
REVOKE INSERT ON EmployeeDB.Employee FROM 'user1'@'localhost';

-- Show grants for a user
SHOW GRANTS FOR 'user1'@'localhost';
```

Input:

4. Usernames and passwords for new users.
5. Privileges to be granted or revoked.

Expected Output:

6. Users created with the specified privileges. The database system should enforce these privileges, allowing or denying access as appropriate.

Lab 6: Case Study Submission for ER Diagram

Aim: To design an Entity-Relationship (ER) diagram for a given case study.

Procedure:

1. Analyze the case study to identify entities, attributes, and relationships.
2. Draw an ER diagram representing the entities, their attributes, and the relationships between them.
3. Specify the cardinality and participation constraints for each relationship.

Source Code:

4. This lab does not involve SQL code but rather the creation of a diagram. Tools like draw.io, Lucidchart, or even hand-drawn diagrams can be used. The output is a visual representation.

Input:

5. A case study description (e.g., "Design a database for an online bookstore").

Expected Output:

6. A complete and accurate ER diagram for the given case study.

Lab 7: SQL using Joins

Aim: To use SQL JOIN clauses to combine data from multiple tables.

Procedure:

1. Create two or more related tables (e.g., "Customers" and "Orders").
2. Use INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN to retrieve data from the tables based on the relationships.

Source Code:

```
-- Create the Customers table
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(100),
    City VARCHAR(50)
);

-- Create the Orders table
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    Amount DECIMAL(10, 2),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

-- Insert sample data
INSERT INTO Customers (CustomerID, CustomerName, City) VALUES
(1, 'Alice Smith', 'New York'),
(2, 'Bob Johnson', 'Los Angeles'),
(3, 'Charlie Brown', 'New York');

INSERT INTO Orders (OrderID, CustomerID, OrderDate, Amount) VALUES
(101, 1, '2023-01-15', 100.00),
(102, 1, '2023-02-20', 200.00),
(103, 2, '2023-03-10', 150.00),
(104, 4, '2024-03-10', 250.00);

-- Inner Join
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate,
Orders.Amount
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

-- Left Join
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

-- Right Join
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
RIGHT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

Input:

3. Data in the Customers and Orders tables.
4. Queries using different types of joins.

Expected Output:

5. Result sets showing the combined data from the tables, according to the type of join used.

Lab 8: SQL using Aggregate Functions and Set Operations

Aim: To use SQL aggregate functions and set operations to analyze and combine data.

Procedure:

1. Use aggregate functions like COUNT(), SUM(), AVG(), MIN(), and MAX() to calculate summary statistics.
2. Use GROUP BY clause to group rows based on one or more columns.
3. Use set operations like UNION, INTERSECT, and EXCEPT (or MINUS in some databases) to combine the results of multiple SELECT queries.

Source Code:

```
-- Create a table
CREATE TABLE Sales (
    SaleID INT PRIMARY KEY,
    ProductID INT,
    Quantity INT,
    SaleDate DATE,
    Region VARCHAR(50)
);

-- Insert sample data
INSERT INTO Sales (SaleID, ProductID, Quantity, SaleDate, Region) VALUES
(1, 101, 10, '2023-01-01', 'North'),
(2, 102, 5, '2023-01-15', 'North'),
(3, 101, 12, '2023-02-01', 'South'),
(4, 103, 8, '2023-02-15', 'South'),
(5, 102, 10, '2023-03-01', 'North'),
(6, 101, 7, '2023-03-15', 'East');

-- Calculate the total quantity sold
SELECT SUM(Quantity) AS TotalQuantitySold FROM Sales;

-- Calculate the average quantity sold per sale
SELECT AVG(Quantity) AS AverageQuantitySold FROM Sales;

-- Find the maximum quantity sold in a single sale
SELECT MAX(Quantity) AS MaxQuantitySold FROM Sales;

-- Count the number of sales in each region
SELECT Region, COUNT(*) AS NumberOfSales FROM Sales GROUP BY Region;

-- Example of UNION (assuming another table Sales2 with same columns)
--  SELECT ProductID FROM Sales
--  UNION
--  SELECT ProductID FROM Sales2;
```

Input:

4. Data in the Sales table.
5. Queries using aggregate functions and set operations.

Expected Output:

6. Result sets showing the calculated statistics and the combined results of the queries.

Lab 9: PL/SQL Conditional and Iterative Statements

Aim: To write PL/SQL programs using conditional (IF, ELSIF, ELSE) and iterative (LOOP, WHILE, FOR) statements.

Procedure:

1. Write PL/SQL blocks to perform different actions based on conditions.
2. Use iterative statements to repeat a block of code multiple times.

Source Code:

```
-- Example of an IF-ELSIF-ELSE statement
DECLARE
    grade CHAR(1);
    marks NUMBER := 85;
BEGIN
    IF marks >= 90 THEN
        grade := 'A';
    ELSIF marks >= 80 THEN
        grade := 'B';
    ELSIF marks >= 70 THEN
        grade := 'C';
    ELSE
        grade := 'D';
    END IF;
    DBMS_OUTPUT.PUT_LINE('Grade: ' || grade);
END;
/

-- Example of a FOR loop
BEGIN
    FOR i IN 1..5 LOOP
        DBMS_OUTPUT.PUT_LINE('Iteration: ' || i);
    END LOOP;
END;
/

-- Example of a WHILE loop
DECLARE
  counter NUMBER := 1;
BEGIN
  WHILE counter <= 5 LOOP
    DBMS_OUTPUT.PUT_LINE('Counter: ' || counter);
    counter := counter + 1;
  END LOOP;
END;
/
```

Input:

3. Values for variables used in the PL/SQL blocks.
4. Conditions for the conditional statements.
5. Loop parameters for the iterative statements.

Expected Output:

6. Output from the PL/SQL blocks, showing the results of the conditional and iterative logic.

Lab 10: PL/SQL Functions and Procedures

Aim: To create and use PL/SQL functions and procedures to modularize code and improve reusability.

Procedure:

1. Write PL/SQL functions that return a value.
2. Write PL/SQL procedures that perform a specific task but do not return a value directly (they might use OUT parameters).
3. Call the functions and procedures from other PL/SQL blocks.

Source Code:

```
-- Example of a function
CREATE OR REPLACE FUNCTION CalculateArea (radius NUMBER)
RETURN NUMBER IS
    area NUMBER;
BEGIN
    area := 3.14159 * radius * radius;
    RETURN area;
END;
/

-- Example of a procedure
CREATE OR REPLACE PROCEDURE GreetUser (username VARCHAR2, greeting OUT
VARCHAR2) IS
BEGIN
    greeting := 'Hello, ' || username || '!';
    DBMS_OUTPUT.PUT_LINE(greeting);
END;
/

-- Calling the function and procedure
DECLARE
    radius NUMBER := 5;
    area_of_circle NUMBER;
    user_greeting VARCHAR2(200);
BEGIN
    area_of_circle := CalculateArea(radius);
    DBMS_OUTPUT.PUT_LINE('Area of circle: ' || area_of_circle);
    GreetUser('John', user_greeting);
    -- DBMS_OUTPUT.PUT_LINE(user_greeting);  -- The procedure itself prints.
END;
/
```

Input:

4. Input values for the functions and procedures.

Expected Output:

5. Output from the PL/SQL blocks, showing the results of calling the functions and procedures.

Lab 11: Case Study Submission for Normalization

Aim: To apply normalization techniques to a database schema to reduce data redundancy and improve data integrity.

Procedure:

1. Analyze a given database schema.
2. Identify any data redundancies and functional dependencies.
3. Normalize the schema to a specified normal form (e.g., 2NF, 3NF, BCNF).
4. Document the normalization process, showing the original schema, the identified problems, and the normalized schema.

Source Code:

5. This lab does not involve SQL code but rather the application of normalization principles.

Input:

6. A description of a database schema or a set of tables.

Expected Output:

7. A normalized database schema, along with documentation of the normalization process.

Lab 12: PL/SQL Cursor

Aim: To use PL/SQL cursors to process multiple rows of a query result.

Procedure:

1. Declare a cursor to select data from one or more tables.
2. Open the cursor.
3. Fetch data from the cursor into variables.
4. Process the fetched data.
5. Close the cursor.

Source Code

```
-- Example of using an explicit cursor
DECLARE
  CURSOR emp_cursor IS
    SELECT EmpName, Salary FROM Employee WHERE DeptID = 1;
  emp_name VARCHAR2(100);
  emp_salary NUMBER;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_name, emp_salary;
    EXIT WHEN emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Name: ' || emp_name || ', Salary: ' || emp_salary);
  END LOOP;
  CLOSE emp_cursor;
```

```
END;
/
```

Input:

6. Queries that return multiple rows of data.

Expected Output:

7. Output from the PL/SQL block, showing the processed data from the cursor.

Lab 13: PL/SQL Trigger

Aim: To create and use PL/SQL triggers to automatically perform actions in response to database events.

Procedure:

1. Identify the event that will trigger the trigger (e.g., INSERT, UPDATE, DELETE).
2. Determine the timing of the trigger (e.g., BEFORE, AFTER).
3. Write the PL/SQL code for the trigger's action.
4. Create the trigger using the CREATE TRIGGER statement.

Source Code:

```
-- Create a table to log changes
CREATE TABLE EmployeeAudit (
    AuditID INT PRIMARY KEY,
    EmpID INT,
    OldSalary NUMBER,
    NewSalary NUMBER,
    UpdateDate DATE
);

-- Create a trigger to log salary changes
CREATE OR REPLACE TRIGGER LogSalaryChange
AFTER UPDATE OF Salary ON Employee
FOR EACH ROW
BEGIN
    INSERT INTO EmployeeAudit (AuditID, EmpID, OldSalary, NewSalary,
UpdateDate) VALUES
    (SEQ_AUDIT.NEXTVAL, :OLD.EmpID, :OLD.Salary, :NEW.Salary, SYSDATE);
END;
/

-- Create a sequence for AuditID if it does not exist
CREATE SEQUENCE SEQ_AUDIT
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;
/

-- Example of updating a salary to fire the trigger
-- UPDATE Employee SET Salary = 65000 WHERE EmpID = 101;
```

Input:

5. Database events that will activate the triggers.

Expected Output:

6. The trigger should automatically execute its defined action when the specified event occurs. For example, in the above code, when the salary of an employee is updated, a new record is inserted into the EmployeeAudit table.

Lab 14: Case Study Submission for Recovery

Aim: To understand database recovery techniques and develop a recovery plan for a given case study.

Procedure:

1. Analyze a case study involving potential database failures.
2. Identify the types of failures that could occur (e.g., transaction failure, system failure, media failure).
3. Describe the recovery techniques that can be used (e.g., transaction logging, checkpoints, backups).
4. Develop a recovery plan for the case study, specifying the actions to be taken in response to different types of failures.

Source Code:

5. This lab does not involve SQL code but rather a plan of action.

Input:

6. A case study describing a database system and its potential failure scenarios.

Expected Output:

7. A detailed recovery plan for the given case study.

Lab 15: Case Study Submission for Database Backups

Aim: To understand the importance of database backups and develop a backup strategy for a given case study.

Procedure:

1. Analyze a case study to determine the requirements for database backups.
2. Identify the types of backups that can be used (e.g., full backups, incremental backups, differential backups).
3. Develop a backup strategy, specifying the frequency of backups, the type of backups to be used, and the storage location for the backups.
4. Consider other factors such as backup retention policies and disaster recovery planning.

Source Code:

5. This lab does not involve SQL code, but rather a backup plan.

Input:

6. A case study describing a database system and its data backup needs.

Expected Output:

7. A comprehensive backup strategy for the given case study.