RNS INSTITUTE OF TECHNOLOGY

Affiliated to VTU, Recognized by GOK, Approved by AICTE, New Delhi
(NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE)
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph: (080) 28611880, 28611881 URL: www.rnsit.ac.in

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DATABASE MANAGEMENT SYSTEMS LABORATORY MANUAL

For Fourth Semester B.E-2022 Batch [VTU/NEP, 2022-26 Syllabus]

Subject Code – BCS403

NAME : .....

USN : .....

SECTION : .....

BATCH: .....

Database Management Systems Laboratory (BCS403)

Program Outcomes (POs) defined by NBA

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PSOs of CSE program

1. The graduates of the Program will have solid foundation in the principles and practices of computer science, including mathematics, science and basic engineering.
2. The graduates of the Program will have skills to function as members of multi-disciplinary teams and to communicate effectively using modern tools.
3. The graduates of the Program will be prepared for their careers in the software industry or pursue higher studies and continue to develop their professional knowledge.
4. The graduates of the program will practice the profession with ethics, integrity, leadership and social responsibility.

Students will be able to:

CO No.	Statement	PO/PSO
CO1	Design a simple database for a given scenario.	PO1, PO2, PO3, PO4, PO5, PO6, PO9, PO10, PO11, PSO1, PSO2, PSO3, PSO4
CO2	Apply various Structured Query Language (SQL) and NOSQL statements for database manipulation.	PO1, PO2, PO3, PO4, PO5, PO9, PO10, PO11, PSO1, PSO2, PSO3
CO3	Analyze the fundamental principles and concepts underlying Relational Database Management Systems (RDBMS) and NoSQL databases.	PO1, PO2, PO3, PO4, PO9, PO10, PO11, PSO1, PSO2, PSO3
CO4	Develop database applications for the given real-world problem.	PO1, PO2, PO3, PO4, PO6, PO8, PO9, PO10, PO11, PO12, PSO1, PSO2, PSO3, PSO4.

Strength of CO Mapping to PO/PSOs with Justification:

COs	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3	PSO 4
BCS 403.1	2	2	3	2	1	2			1	2	1		1	1	2	1
BCS 403.2	2	2	3	2	1				1	1	1		1	1	2	
BCS 403.3	2	3	2	2					1	1	1		2	1	2	
BCS 403.4	3	2	2	2	3	1		1	1	2	1	1	2	2	2	2

DATABASE MANAGEMENT SYSTEM LABORATOR - BCS403

INTERNAL EVALUATION SHEET

EVALUATION (MAX MARKS 50)			
TEST A	**REGULAR EVALUATION B**	**RECORD C**	**TOTAL MARKS A+B+C**
10	**10**	**5**	**25**

R1: REGULAR LAB EVALUATION WRITE UP RUBRIC (MAX MARKS 10)				
Sl. No.	**Parameters**	**Good**	**Average**	**Needs improvement**
a.	**Understanding of problem (3 marks)**	Clear understanding of database schema while designing and implementing the database (3)	Database schema is understood clearly but few mistakes while designing and implementing structure (2)	Database schema is not clearly understood while designing the program (1)
b.	**Writing SQL Commands (4 marks)**	Structured Query Language (SQL) commands handles all possible constraints (4)	Average constraints are defined and verified. (3)	SQL commands does not handle all constraints (1)
c.	**Result and documentation (3 marks)**	Meticulous documentation and all conditions are taken care (3)	Acceptable documentation shown (2)	Documentation does not take care all conditions (1)

R2: REGULAR LAB EVALUATION VIVA RUBRIC (MAX MARKS 10)					
Sl. No.	**Parameter**	**Excellent**	**Good**	**Average**	**Needs Improvement**
a.	**Conceptual understanding (10 marks)**	Answers 80% of the viva questions asked (10)	Answers 60% of the viva questions asked (7)	Answers 30% of the viva questions asked (4)	Unable to relate the concepts (1)

R3: REGULAR LAB PROGRAM EXECUTION RUBRIC (MAX MARKS 10)				
Sl. No.	**Parameters**	**Excellent**	**Good**	**Needs Improvement**
a.	**Design, implementation and demonstration (5 marks)**	Commands follows syntax and semantics of Structured Query Language. Demonstrates the complete knowledge of the program written (5)	Commands has few logical errors, moderately demonstrates all possible concepts implemented in database (3)	Syntax and semantics of SQL Commands are not clear (1)
b.	**Result and documentation (5 marks)**	All test cases are successful, all errors are debugged with own practical knowledge and clear documentation according to the guidelines (5)	Moderately debugs the programs , few test case are unsuccessful and Partial documentation (3)	Test cases are not taken care , unable to debug the errors and no proper documentation (1)

R4: RECORD EVALUATION RUBRIC (MAX MARKS 20)					
Sl. No.	**Parameter**	**Excellent**	**Good**	**Average**	**Needs Improvement**
a.	**Documentation (10 marks)**	Meticulous record writing including program, comments and test cases as per the guidelines mentioned (20)	Write up contains program and test cases, but comments are not included (18)	Write up contains only program (15)	Program written with few mistakes (10)

TEST /LAB INTERNALS MARKS (MAX MARKS 10)

TEST #	Write up 6	Execution 28	Viva 6	Sign	Total 40	Avg. 40	Final 10
TEST-1						⬚ / 40	⬚ / 10
TEST-2							

REGULAR LAB EVALUATION (MAX MARKS 15)

Lab program	Date of Execution	Additional programs	Write up (10)	Exen. (10)	Viva (10)	Total 30	Teacher Signature
1							
2							
3							
4							
5							
6							
7							
Total Marks		⬚ / 210	⬚ / 30			⬚ / 15	

Final Marks obtained from test (10) + regular evaluation (15)	⬚ / 25	**Lab in charge :**
		HOD :

1. Create a table called Employee & execute the following. Employee(EMPNO,ENAME,JOB, MANAGER_NO, SAL, COMMISSION)

1. Create a user and grant all permissions to theuser.

2. Insert the any three records in the employee table contains attributes EMPNO,ENAME JOB, MANAGER_NO, SAL, COMMISSION and use rollback. Check the result.

3. Add primary key constraint and not null constraint to the employee table.

4. Insert null values to the employee table and verify the result.

OUTPUT

```
1-- Create the Employee table
CREATE TABLE Employee (
    EMPNO INT,
    ENAME VARCHAR(50),
    JOB VARCHAR(50),
    MANAGER_NO INT,
    SAL DECIMAL(10,2),
    COMMISSION DECIMAL(10,2)
);

-- Create a user
CREATE USER theuser IDENTIFIED BY password;

-- Grant all permissions to the user
GRANT ALL PRIVILEGES ON Employee TO theuser;
```

Note: This series of SQL commands should accomplish what you've asked for. Make sure to replace 'password' with a secure password for the user 'theuser'.

```
2- Insert three records into the Employee table
INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES
    (1, 'John Doe', 'Manager', NULL, 50000.00, 5000.00),
    (2, 'Jane Smith', 'Developer', 1, 40000.00, NULL),
    (3, 'Michael Johnson', 'Salesperson', 1, 30000.00, 2000.00);

-- Rollback to undo the inserts
ROLLBACK;

3- Add Primary Key constraint to EMPNO column

ALTER TABLE Employee
```

```
ADD CONSTRAINT PK_Employee_EMPNO PRIMARY KEY (EMPNO);

-- Add Not Null constraints
ALTER TABLE Employee
ALTER COLUMN EMPNO INT NOT NULL,
ALTER COLUMN ENAME VARCHAR(50) NOT NULL,
ALTER COLUMN JOB VARCHAR(50) NOT NULL,
ALTER COLUMN SAL DECIMAL(10,2) NOT NULL;

4-- Inserting null values

INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)
VALUES
   (NULL, 'Chris Brown', NULL, NULL, 45000.00, NULL);

-- Selecting all records to verify
SELECT * FROM Employee;
```

2. Create a table called Employee that contain attributes EMPNO, ENAME, JOB, MGR,SAL & execute the following.

1. Add a column commission with domain to the Employee table.

2. Insert any five records into the table.

3. Update the column details of job

4. Rename the column of Employ table using alter command.

5. Delete the employee whose Empno is 105.

```
1-- Create the Employee table
CREATE TABLE Employee (
   EMPNO INT,
   ENAME VARCHAR(50),
   JOB VARCHAR(50),
   MGR INT,
   SAL DECIMAL(10,2)
);

-- Add a new column 'commission' to the Employee table
ALTER TABLE Employee
ADD COMMISSION DECIMAL(10,2);
```

2 -- Insert five records into the Employee table

```
INSERT INTO Employee (EMPNO, ENAME, JOB, MGR, SAL, COMMISSION)
VALUES
   (101, 'John Doe', 'Manager', NULL, 50000.00, 2000.00),
   (102, 'Jane Smith', 'Developer', 101, 40000.00, 1500.00),
   (103, 'Michael Johnson', 'Salesperson', 101, 30000.00, NULL),
   (104, 'Emily Brown', 'Analyst', 102, 45000.00, 2500.00),
   (105, 'David Lee', 'Intern', 102, 25000.00, NULL);
```

3 -- Update the job details for a specific employee

```
UPDATE Employee
SET JOB = 'Senior Developer'
WHERE EMPNO = 102;
```

4 ALTER TABLE Employee

```
RENAME COLUMN Employ TO Employee;
```

5 -- Delete the employee whose Empno is 105

```
DELETE FROM Employee
WHERE EMPNO = 105;
```

3.Queries using aggregate functions(COUNT,AVG,MIN,MAX,SUM), Groupby, Orderby. Employee(E_id, E_name, Age, Salary)

1. Create Employee table containing all Records E_id, E_name, Age, Salary.

2. Count number of employee names from employee table

3. Find the Maximum age from employee table.

4. Find the Minimum age from employee table.

5. Find salaries of employee in Ascending Order.

6. Find grouped salaries of employees.

1 CREATE TABLE Employee (

```
   E_id INT,
   E_name VARCHAR(50),
   Age INT,
   Salary DECIMAL(10,2) );
```

2 SELECT COUNT(E_name) AS Num_of_Employees

FROM Employee;

3 SELECT MAX(Age) AS Max_Age

FROM Employee;

4 SELECT MIN(Age) AS Min_Age

FROM Employee;

5 SELECT E_name, Salary

FROM Employee
ORDER BY Salary ASC;

6 SELECT Salary, COUNT(*) AS Num_of_Employees

FROM Employee
GROUP BY Salary;

4. Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old & new Salary.

CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

```
CREATE OR REPLACE TRIGGER salary_difference_trigger
BEFORE INSERT OR UPDATE OR DELETE ON CUSTOMERS
FOR EACH ROW
DECLARE
   v_old_salary NUMBER;
   v_new_salary NUMBER;
BEGIN
   IF INSERTING OR UPDATING THEN
      v_old_salary := NVL(:OLD.SALARY, 0);
      v_new_salary := NVL(:NEW.SALARY, 0);

      DBMS_OUTPUT.PUT_LINE('Salary difference for ' || :NEW.NAME || ': ' || (v_new_salary
- v_old_salary));
   END IF;

   IF DELETING THEN
      v_old_salary := NVL(:OLD.SALARY, 0);

      DBMS_OUTPUT.PUT_LINE('Salary difference for ' || :OLD.NAME || ': ' || (-
v_old_salary));
```

```
    END IF;
END;
/
```

Note: This trigger named **salary_difference_trigger** will execute for each row affected by INSERT, UPDATE, or DELETE operations on the **CUSTOMERS** table. Inside the trigger, it checks if it's an INSERT or UPDATE operation to calculate the salary difference between the old and new salary values. For DELETE operations, it calculates the salary difference using only the old salary value. The **DBMS_OUTPUT.PUT_LINE** function is used to display the salary difference in the database output.

5. **Create cursor for Employee table & extract the values from the table. Declare the variables , Open the cursor & extract the values from the cursor. Close the cursor. Employee(E_id, E_name, Age, Salary)**

```
-- Declare variables to store values fetched from the cursor
DECLARE
    v_E_id Employee.E_id%TYPE;
    v_E_name Employee.E_name%TYPE;
    v_Age Employee.Age%TYPE;
    v_Salary Employee.Salary%TYPE;

    -- Declare cursor for the Employee table
    CURSOR emp_cursor IS
        SELECT E_id, E_name, Age, Salary
        FROM Employee;
BEGIN
    -- Open the cursor
    OPEN emp_cursor;

    -- Fetch values from the cursor
    LOOP
        FETCH emp_cursor INTO v_E_id, v_E_name, v_Age, v_Salary;

        -- Exit the loop if no more rows to fetch
        EXIT WHEN emp_cursor%NOTFOUND;

        -- Process fetched values (You can perform any operation here)
        DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_E_id || ', Employee Name: ' || v_E_name
|| ', Age: ' || v_Age || ', Salary: ' || v_Salary);
    END LOOP;

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

Note: This PL/SQL block declares variables to store values fetched from the cursor. It then declares a cursor **emp_cursor** to select all columns from the **Employee** table. The cursor is opened, and a loop fetches values from the cursor into the declared variables. Inside the loop, you can perform any operations on the fetched values. Finally, the cursor is closed. Adjust column names and types according to your actual schema.

6. Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.

```
CREATE OR REPLACE PROCEDURE Merge_RollCall_Data AS
  CURSOR N_RollCall_Cursor IS
    SELECT *
    FROM N_RollCall;

  v_N_RollCall_Record N_RollCall%ROWTYPE;
BEGIN
  FOR v_N_RollCall_Record IN N_RollCall_Cursor LOOP
    -- Check if data already exists in O_RollCall table
    SELECT COUNT(*)
    INTO v_Count
    FROM O_RollCall
    WHERE EMPNO = v_N_RollCall_Record.EMPNO
    AND ROLL_DATE = v_N_RollCall_Record.ROLL_DATE;

    -- If data doesn't exist, insert into O_RollCall
    IF v_Count = 0 THEN
      INSERT INTO O_RollCall (EMPNO, ROLL_DATE, STATUS)
      VALUES (v_N_RollCall_Record.EMPNO, v_N_RollCall_Record.ROLL_DATE,
v_N_RollCall_Record.STATUS);
    END IF;
  END LOOP;
  COMMIT;
END Merge_RollCall_Data;
/
```

Note: This PL/SQL procedure **Merge_RollCall_Data** uses a cursor to iterate through each record in the **N_RollCall** table. For each record, it checks if the corresponding data already exists in the **O_RollCall** table based on **EMPNO** and **ROLL_DATE**. If the data doesn't exist, it inserts the record into the **O_RollCall** table. Finally, it commits the transaction to make the changes permanent.

7. **Install an Open Source NoSQL Data base MangoDB & perform basic CRUD(Create, Read, Update & Delete) operations. Execute MangoDB basic Queries using CRUD operations.**

MongoDB is actually its own product, not an open-source version of MongoDB.

1. **Installation**
   - Download MongoDB from Official Website: MongoDB Download Center
   - Follow the Installation instruction provided for your Operating system.
2. **Starting MongoDB**
   After installation, start MongoDb by running the appropriate command for your OS. For example

   sudo service mongod start (Linux)
   mongod (Windows)

3. **Access MongoDB Shell**
   - Open a new terminal or command prompt and run the mongo command to access the MongoDB shell.

Performing Basic CRUD Operations:

Create(Insert) Operations:

```
// Insert a document into a collection named 'users'
db.users.insertOne({ name: "John", age: 30, city: "New York" });

// Find all documents in the 'users' collection
db.users.find();

// Find documents with a specific condition
db.users.find({ age: { $gte: 25 } });  // Find users with age greater than or equal to 25

// Update a document in the 'users' collection
db.users.updateOne({ name: "John" }, { $set: { age: 35 } });

// Delete a document from the 'users' collection
db.users.deleteOne({ name: "John" });
```

Executing MongoDB Queries:

- You can execute the CRUD operations directly in the MongoDB shell as shown above

- Additionally you can also execute queries programmatically using MongoDb drivers in your preferred programming language (eg. Python, Node.js, Java)
- Remember to adjust collection names and field values according to your actual schema and requirements.