Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(NAAC Accredited & ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

Department of Artificial Intelligence and Data Science

LAB MANUAL

DATABASE MANAGEMENT SYSTEM(BCS403)

Academic Year:2025-26

IV Semester

Name:_____

USN:_____ Batch:_____

Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(NAAC Accredited & ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

Department of Artificial Intelligence and Data Science

LAB MANUAL

DATABASE MANAGEMENT SYSTEM(BCS403)

Academic Year: 2025-26

IV Semester

Course coordinator
Nagapushpa B M
Associate Professor
Dept. Of AD

HOD
Dr.Gavisiddappa
Prof & Head
Dept. Of AD

CHANNABASAVESHWARA INSTITUTE OF TECHNOLOGY
(Affiliated toVTU,Belgaum & Approved by AICTE,NewDelhi)
(NAAC Accredited & ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka

DEPARTMENT OF ARTIFICIAL OF INTELLIGENCE AND DATA SCIENCE

VISION OF THE DEPARTMENT

To establish center of excellence by nurturing Artificial Intelligence and Data Science engineers through continuous education, research and industrial collaboration to serve the needs of society.

MISSION OF THE DEPARTMENT

1. By providing a nurturing environment together with competent human resources in order to develop a diverse pool of skilled Artificial Intelligence and Data Science engineers who uphold professional readiness for the industry.
2. By providing state of the art facilities and cutting edge technology, we create an optimal learning and research environment, enhancing the capabilities of our Artificial Intelligence and Data Science students and faculty.
3. By providing avenues for industry partnerships and research collaborations, we facilitate practical experience and innovation, enabling our students to address real world challenges in Artificial Intelligence and Data Science.
4. By providing a strong foundation in professional ethics and commitment to social responsibility, we equip our students with the values needed to make a positive societal impact as industry ready professionals in the field of Artificial Intelligence and Data Science.

PRACTICAL COMPONENT OF IPCC

Sl.NO	Experiments
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 the user. 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 constraints and not null constraints to the employee table. 4. Insert null values to the employee table and verify the result.
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.
3.	Queries using aggregate functions(COUNT, AVG, MIN, SUM), Group by, Order by. **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.
4.	Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performer on the CUSTOMERS table. This trigger will display the salary difference between the old & new Salary. **CUSTOMERS(ID, NAME, AGE, ADDRESS, SALARY)**
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)**
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 exists in the second table then the data should be skipped.
7.	Install an Open Source NoSQL Data base MangoDB& perform basics CRUD(Create, Read, Update & Delete) operations. Execute MangoDB basic Quries using CRUD operations.

Course outcomes:

At the end of the course, the student will be able to:

- Describe the basic elements of a relational database management system
- Design entity relationship for the given scenario.
- Apply various Structured Query Language (SQL) statements for database manipulation.
- Analyse various normalization forms for the given application.
- Develop database applications for the given real world problem.
- Understand the concepts related to NoSQL databases.

Assessment Details

15 marks for the conduction of the experiment and preparation of laboratory record, and 10 marks for the test to be
conducted after the completion of all the laboratory sessions.

- On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voce
and marks shall be awarded on the same day.
- The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the
laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are
added and scaled down to 15 marks.
- The laboratory test (duration 02/03 hours) after completion of all the experiments shall be conducted for 50 marks
and scaled down to 10 marks.
- Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC for 25 marks.
- The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IPCC.

SYLLABUS
BCS403

SEMESTER – I V

Subject Code: BCS403	CIE Marks: 50
Number of Lecture Hours/Week: 3:0:2:0	SEE Marks: 50
Exam Hours: 03	Credits – 04

Programs List:

PART A

List of problems for which student should develop program and execute in the laboratory.

1. create a table called employee &execute the following.

   Employee(EMPNO,ENAME,JOB,MANAGER_NO,SAL,COMMISSION)

   i. Create a user and grant all permissions to the user.

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

   iii. Add primary key constraints and not null constraints to the employee table.

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

2. Create a table called Employee that contain attributes EMPNO,ENAME, JOB, MGR, SAL & execute the following.
   i. Add a column commission with domain to the Employee table.
   ii. Insert any five records into the table.
   iii. Update the column details of job
   iv. Rename the column of Employ table using alter command.
   v. Delete the employee whose Empno is 105.

3. Queries using aggregate functions(COUNT, AVG, MIN, SUM), Group by, Order by.
   Employee(E_id, E_name, Age, Salary)
   i. Create Employee table containing all records E_id, E_name, Age, Salary.
   ii. Count number of employee names from employee table.
   iii. Find the Maximum age from employee table.
   iv. Find the Minimum age from employee table.

v. Find salaries of employee in Ascending Order.

vi Find grouped salaries of employees.

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

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

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)

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 exists in the second table then the data should be skipped.

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

INTRODUCTION

What is DBMS?

Database Management Systems (DBMS) are software systems used to store, retrieve and run queries on data. A DBMS serves as an interface between an end-user and a database, allowing users to create, read, update, and delete data in the database.

What is SQL?

- SQL is a database language. SQL is used widely and almost all Relational Database Management Systems can recognize it.

- SQL contains a set of commands that enable you to create a database. You can also use it to execute commands in your Relational Database Management System.

- SQL has certain advantages which have helped it thrive from the 1970s until now. It is widely accepted by both people and platforms, in part because of the following features:

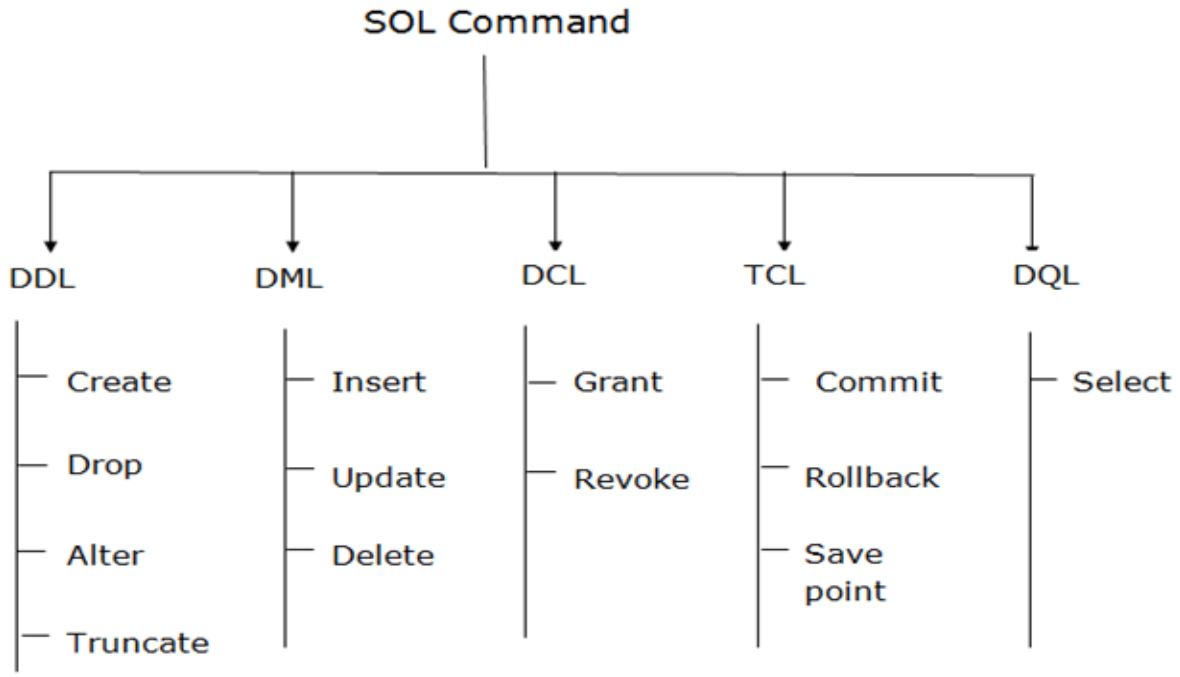
    - SQL is fast
    - SQL is a very high-level language
    - SQL is a platform-independent language
    - SQL is a standardized language
    - SQL is a portable language

DATA TYPES:

- Numeric: NUMBER, NUMBER (S, P), INTEGER, INT, FLOAT, DECIMAL.
- Character: CHAR(n), VARCHAR(n), VARCHAR2(n), CHAR VARYING(n).
- Bit String: BLOB, CLOB.
- Boolean: true, false and null
- Date and Time: DATE(YYYY-MM-DD) TIME (HH:MM: SS).
- Timestanmp: DATE+ TIME.

There are five types of SQL statements. They are:

- 1. DATA DEFINITION LANGUAGE (DDL)
- 2. DATA MANIPULATION LANGUAGE (DML)
- 3. DATA RETRIEVAL LANGUAGE (DRL)
- 4. TRANSATIONAL CONTROL LANGUAGE (TCL)
- 5. DATA CONTROL LANGUAGE (DCL)



1. Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

a. CREATE: It is used to create a new table in the database.

Syntax: CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);

Example:

CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);

b. DROP: It is used to delete both the structure and record stored in the table.

Syntax: DROP TABLE table_name;

Example: DROP TABLE EMPLOYEE;

c. ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

Syntax: To add a new column in the table

ALTER TABLE table_name ADD column_name COLUMN-definition;

To modify existing column in the table:

ALTER TABLE table_name MODIFY(column_definitions....);

EXAMPLE

1. ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));
2. ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));

d. TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

Syntax:TRUNCATE TABLE table_name;

Example: TRUNCATE TABLE EMPLOYEE;

2. Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

a. INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

Syntax:

```
INSERT INTO TABLE_NAME
(col1, col2, col3,.... col N)
VALUES (value1, value2, value3, .... valueN);
```

Or

```
INSERT INTO TABLE_NAME
VALUES (value1, value2, value3, .... valueN);
```

For example:

INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");

b. UPDATE: This command is used to update or modify the value of a column in the table.

Syntax:UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]

For example:

```
1. UPDATE students
2. SET User_Name = 'Sonoo'
3. WHERE Student_Id = '3'
```

c. DELETE: It is used to remove one or more row from a table.

Syntax: DELETE FROM table_name [WHERE condition];

For example:

```
DELETE FROM javatpoint
WHERE Author="Sonoo";
```

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database.

Example:

```
GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;
```

b. Revoke: It is used to take back permissions from the user.

Example: REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax: COMMIT;

Example:

```
1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. COMMIT;
```

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax: ROLLBACK;

Example:

```
DELETE FROM CUSTOMERS
WHERE AGE = 25;
ROLLBACK;
```

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax: SAVEPOINT SAVEPOINT_NAME;

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

```
SELECT expressions
FROM TABLES
WHERE conditions;
```

For example:

```
1. SELECT emp_name
2. FROM employee
3. WHERE age > 20;
```

EXPERIMENTS

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 the user.**
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 c**
4. **onstraints and not null constraints to the employee table.**
5. **Insert null values to the employee table and verify the result.**

⟶

```
CREATE DATABASE your_database_name;
USE your_database_name;

CREATE TABLE employee (
        EMPNO INT,
        ENAME VARCHAR(255),
        JOB VARCHAR(255),
        MANAGER_NO INT,
        SAL DECIMAL(10,2),
        COMMISSION DECIMAL(10,2)
 );
```

--1. Create a user and grant all permissions to the user.

```
CREATE USER your_user_name IDENTIFIED BY 'your_password';
GRANT ALL PRIVILEGES ON your_database_name.employee TO your_user_name;
```

--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.

```
INSERT INTO employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL,
COMMISSION)
VALUES
        (1, 'John Doe', 'Manager', NULL, 50000, 10000),
        (2, 'Jane Smith', 'Assistant', 1, 30000, NULL),
        (3, 'Alice Johnson', 'Clerk', 2, 25000, 500);
ROLLBACK;
```

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

```
ALTER TABLE employee
```

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

ALTER TABLE employee MODIFY EMPNO INT NOT NULL,
MODIFY ENAME VARCHAR(255) NOT NULL,
MODIFY JOB  VARCHAR(255) NOT NULL,
MODIFY  SAL DECIMAL(10,2) NOT NULL;
```

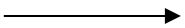
--4. Insert null values to the employee table and verify the result

```
INSERT INTO employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL,
COMMISSION)
VALUES (4, NULL, 'Intern', 2, NULL, NULL);
```

-- Display the updated Employee table

```
SELECT * FROM employee;
```

2. **Create a table called Employee that contain attributes EMPNO,ENAME, JOB, MGR, SAL & execute the following.**
   Add a column commission with domain to the Employee table.
   1. **Insert any five records into the table.**
   2. **Update the column details of job.**
   3. **Rename the column of Employ table using alter command.**
   4. **Delete the employee whose Empno is 105.**



```
-- Create the Employee table

    CREATE TABLE Employee (
      EMPNO INT,
      ENAME VARCHAR(255),
      JOB VARCHAR(255),
      MGR INT,
      SAL DECIMAL(10, 2)
    );
-- 1. Add a column 'commission' to the Employee table

    ALTER TABLE Employee
    ADD COLUMN commission DECIMAL(10, 2);

-- 2. Insert five records into the table
    INSERT INTO Employee (EMPNO, ENAME, JOB, MGR, SAL, commission)
    VALUES
      (101, 'John', 'Manager', NULL, 50000, 10000),
      (102, 'Jane', 'Assistant', 101, 30000, NULL),
      (103, 'Alice', 'Clerk', 102, 25000, 500),
      (104, 'Bob', 'Developer', 101, 60000, 20000),
      (105, 'Charlie', 'Sales', 101, 40000, NULL);

-- 3. Update the 'JOB' column details

    UPDATE Employee
    SET JOB = 'Sales Manager'
    WHERE EMPNO = 105;

-- 4. Rename the column 'ENAME' of the Employee table

    ALTER TABLE Employee
    CHANGE COLUMN ENAME employee_nameVARCHAR(255);
-- 5. Delete the employee whose EMPNO is 105

    DELETE FROM Employee
    WHERE EMPNO = 105;

-- Display the updated Employee table
     SELECT * FROM Employee;
```

3. Queries using aggregate functions (COUNT, AVG, MIN, SUM), Group by, Order by.

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.**

⟶

```
-- . Create the Employee table

        CREATE TABLE Employee (
        E_id INT,
        E_nameVARCHAR(255),
           Age INT,
           Salary DECIMAL(10, 2)
        );

-- 1. Insert some records into the Employee table (assuming you already have data)

        INSERT INTO Employee (E_id, E_name, Age, Salary)
        VALUES
           (1, 'John', 30, 50000),
           (2, 'Jane', 35, 60000),
           (3, 'Alice', 25, 45000),
           (4, 'Bob', 40, 70000),
           (5, 'Charlie', 28, 55000);

-- 2. Count the number of employee names
        SELECT COUNT(E_name) AS count FROM Employee;

-- 3. Find the Maximum age
        SELECT MAX(Age) AS max_age FROM Employee;
-- 4. Find the Minimum age

        SELECT MIN(Age) AS min_age FROM Employee;

-- 5. Find salaries of employees in Ascending Order

        SELECT E_name, Salary
        FROM Employee
        ORDER BY Salary ASC;
-- 6. Find grouped salaries of employees

        SELECT Salary, COUNT(*) AS num_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)

- A trigger in SQL is a special type of stored procedure that is automatically executed or "triggered" when specific events occur in a database table.
- They are often defined to respond to INSERT, UPDATE, or DELETE operations on a table.

Key Components of a Trigger

1. **Event**: The specific database operation that causes the trigger to fire (e.g., INSERT, UPDATE, DELETE).
2. **Timing**: When the trigger is fired relative to the event (BEFORE or AFTER).
3. **Scope**: Whether the trigger is row-level (FOR EACH ROW) or statement-level (applies to the whole statement).
4. **Action**: The SQL statements or PL/SQL code that is executed when the trigger fires.

Syntax

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER}
{INSERT | UPDATE | DELETE}
ON table_name
FOR EACH ROW
BEGIN
  -- trigger logic
END;
```

Trigger Definition:

- **CREATE OR REPLACE TRIGGER display_salary_difference:** Creates a new trigger or replaces an existing one named display_salary_difference.
- **AFTER INSERT OR UPDATE OR DELETE ON CUSTOMERS:** Specifies that the trigger will fire after INSERT, UPDATE, or DELETE operations on the CUSTOMERS table.
- **FOR EACH ROW:** Indicates that the trigger is a row-level trigger.

Declare Variables:

- old_salary, new_salary, and salary_diff are declared to hold the salary values and their difference.

DML Operation Check:

- IF INSERTING THEN ... ELSIF UPDATING THEN ... ELSIF DELETING THEN ... END IF;: This block checks the type of DML operation and sets the old_salary and new_salary accordingly.

Calculate Salary Difference:

- salary_diff := new_salary - old_salary;: Calculates the difference between the new and old salary.

Display Salary Difference:

- DBMS_OUTPUT.PUT_LINE('Salary difference: ' || salary_diff);: Uses the DBMS_OUTPUT.PUT_LINE procedure to display the salary difference.
- DBMS_OUTPUT.PUT_LINE is used to display the output in Oracle.
- Make sure SET SERVEROUTPUT ON is enabled in your SQL*Plus or SQL Developer session to see the output.

```
DELIMITER //
CREATE TRIGGER salary_difference_trigger
AFTER INSERT ON customers1
FOR EACH ROW
BEGIN
```

```
   DECLARE old_salary DECIMAL(10, 2);
   DECLARE new_salary DECIMAL(10, 2);


   SET old_salary = (SELECT SALARY FROM customers WHERE ID = NEW.ID);
   SET new_salary = NEW.SALARY;

   IF old_salary IS NOT NULL THEN
            SET @message_text = CONCAT('Salary difference for ID ', NEW.ID, ': ', new_salary -
old_salary);
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = @message_text;

   END IF;
END;
//


CREATE TRIGGER salary_difference_update_trigger
AFTER UPDATE ON customers1
FOR EACH ROW
BEGIN
   DECLARE old_salary DECIMAL(10, 2);
   DECLARE new_salary DECIMAL(10, 2);

   SET old_salary = OLD.SALARY;
   SET new_salary = NEW.SALARY;

   IF old_salary IS NOT NULL THEN
            SET @message_text = CONCAT('Salary difference for ID ', OLD.ID, ': ', new_salary -
old_salary);
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = @message_text;

   END IF;
END;
//

CREATE TRIGGER salary_difference_delete_trigger
AFTER DELETE ON customers1
FOR EACH ROW
BEGIN
   DECLARE old_salary DECIMAL(10, 2);

   SET old_salary = OLD.SALARY;

   IF old_salary IS NOT NULL THEN
            SET @message_text = CONCAT('Salary deleted for ID ', OLD.ID, ': ', old_salary);
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = @message_text;

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

```
SQL> select * from customers;

        ID NAME              AGE ADDRESS                        SALARY
---------- ---------- ---------- ------------------------- ----------
         1 suhas              30 Tumkur                         50000
         2 lakshmi            40 Gubbi                          80000
         3 pavan              50 Bangalore                      70000
         4 ramachari          40 Tumkur                         90000
```

```
Trigger created.

SQL> UPDATE CUSTOMERS SET SALARY = 75000 WHERE ID = 1;
Old salary: 50000
New salary: 75000
Salary difference: 25000

1 row updated.

SQL>
```

```
1 row updated.

SQL> DELETE FROM CUSTOMERS WHERE ID = 1;
Old salary: 75000

1 row deleted.

SQL>
```

```
1 row created.

SQL> select * from employee;

      E_ID E_NAME            AGE     SALARY
---------- ---------- ---------- ----------
         1 John               30      50000
         1 John               30      50000
         2 suhas              30      50000
         3 rashmika           30      50000

SQL>
```

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

- A cursor in SQL is a database object used to retrieve, manipulate, and navigate through a result set row-by-row.
- It acts as a pointer that allows for operations on individual rows

returned by a query, making it possible to perform complex logic that involves processing each row separately.

Types of Cursors:

Implicit Cursors: Automatically created by Oracle when a SQL statement is executed. These are managed by the database and do not require explicit declaration.

Explicit Cursors: Defined by the user and provide more control over the context area. These are useful for more complex operations that require processing of individual rows.

Cursor Operations:

1. **Declare:** Define the cursor and the SQL query it will use.
2. **Open:** Initialize the cursor and execute the query to establish the result set.
3. **Fetch**: Retrieve the next row from the cursor's result set into variables.
4. **Close:** Release the cursor and free the associated resources**.**

Variable Declaration:

- **v_E_id, v_E_name, v_Age, v_Salary:** Variables to hold the values fetched from the Employee table.
- Their types are derived from the corresponding columns in the Employee table using **%TYPE.**

Cursor Declaration:

emp_cursor: A cursor that selects the E_id, E_name, Age, and Salary columns from the Employee table.

Open Cursor:

OPEN emp_cursor;: Opens the cursor for fetching.

Fetch Values in Loop:

The LOOP is used to fetch each row from the cursor.

- **FETCH emp_cursor INTO v_E_id, v_E_name, v_Age, v_Salary;**
- **Fetches the current row into the declared variables.**
- **EXIT WHEN emp_cursor % NOTFOUND;: Exits the loop when there are no more rows to fetch.**
- **DBMS_OUTPUT.PUT_LINE: Displays the fetched values.**

Close Cursor:

CLOSE emp_cursor;: Closes the cursor after fetching all the rows.

DBMS_OUTPUT.PUT_LINE is a procedure in Oracle PL/SQL used to display output from PL/SQL blocks.

Make sure SET SERVEROUTPUT ON is enabled in your SQL*Plus or SQL Developer session to see the output

```
CREATE TABLE Employee(
    E_id INT,
   E_name VARCHAR(255),
     Age INT,
    Salary DECIMAL(10, 2)
   );
```

```
INSERT INTO Employee (E_id, E_name, Age, Salary)
   VALUES
     (1, 'John', 30, 50000),
      (2, 'Jane', 35, 60000),
     (3, 'Alice', 25, 45000),
      (4, 'Bob', 40, 70000),
      (5, 'Charlie', 28, 55000);
DELIMITER //
CREATE PROCEDURE fetch_employee_data()
BEGIN

DECLARE emp_id INT;
DECLARE emp_name VARCHAR(255);
DECLARE emp_age INT;
DECLARE emp_salary DECIMAL(10, 2);

DECLARE emp_cursor CURSOR FOR
SELECT E_i d, E_name, Age, Salary
FROM Employee;

DECLARE CONTINUE HANDLER FOR NOT FOUND
SET @finished = 1;

OPEN emp_cursor;

SET @finished = 0;

cursor_loop: LOOP

FETCH emp_cursor INTO emp_id, emp_name, emp_age, emp_salary;

IF @finished = 1 THEN
LEAVE cursor_loop;
END IF;

SELECT CONCAT('Employee ID: ', emp_id, 'Name: ', emp_name, 'Age:',
emp_age,'Salary:', emp_salary) AS Employee_Info;
END LOOP;
CLOSE emp_cursor;
END//
DELIMITER ;
```

```
 25  END;
 26  /
E_id: 1, E_name: John, Age: 30, Salary: 50000
E_id: 1, E_name: John, Age: 30, Salary: 50000
E_id: 2, E_name: suhas, Age: 30, Salary: 50000
E_id: 3, E_name: rashmika, Age: 30, Salary: 50000

PL/SQL procedure successfully completed.
```

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

→

```
DECLARE
v_count NUMBER;
   CURSOR c_new_rollcall IS
      SELECT id, name, roll
      FROM N_RollCall2;
BEGIN
   FOR new_rec IN c_new_rollcall LOOP
      -- Check if the record already exists in O_RollCall2
      SELECT COUNT(*)
      INTO   v_count
      FROM   O_RollCall2
WHERE  id = new_rec.id;

      -- If record doesn't exist, insert it
      IF v_count = 0 THEN
         INSERT INTO O_RollCall2 (id, name, roll)
         VALUES (new_rec.id, new_rec.name, new_rec.roll);
         DBMS_OUTPUT.PUT_LINE('Record inserted: ' || new_rec.id);
      ELSE
         DBMS_OUTPUT.PUT_LINE('Record skipped: ' || new_rec.id);
      END IF;
   END LOOP;
   COMMIT;
END;

select * from N_RollCall2;
select * from O_RollCall2;

INSERT INTO N_RollCall2 (id, name, roll)
VALUES (1121, 'satya12333', 111111111);


delete from o_rollcall2 where id=121;
```

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

```
use bookDB

db.createCollection("ProgrammingBooks")

db.ProgrammingBooks.insertMany([
  {
   title: "Clean Code: A Handbook of Agile Software Craftsmanship",
   author: "Robert C. Martin",
   category: "Software Development",
   year: 2008
  },
  {
   title: "JavaScript: The Good Parts",
   author: "Douglas Crockford",
   category: "JavaScript",
   year: 2008
  },
  {
   title: "Design Patterns: Elements of Reusable Object-Oriented Software",
   author: "Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides",
   category: "Software Design",
   year: 1994
  },
  {
   title: "Introduction to Algorithms",
   author: "Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein",
   category: "Algorithms",
   year: 1990
  },
  {
   title: "Python Crash Course: A Hands-On, Project-Based Introduction to Programming",
   author: "Eric Matthes",
   category: "Python",
   year: 2015
  }
])


// Display all documents in the 'ProgrammingBooks' collection
db.ProgrammingBooks.find().pretty()

db.ProgrammingBooks.insertOne({
  title: "The Pragmatic Programmer: Your Journey to Mastery",
  author: "David Thomas, Andrew Hunt",
```

```
  category: "Software Development",
  year: 1999
})

db.ProgrammingBooks.find().pretty()

//find books published after the year 2000
db.ProgrammingBooks.find({ year: { $gt: 2000 } }).pretty()

//change the author of a book
db.ProgrammingBooks.updateOne(
  { title: "Clean Code: A Handbook of Agile Software Craftsmanship" },
  { $set: { author: "Robert C. Martin (Uncle Bob)" } }
)

db.ProgrammingBooks.find({ year: { $eq: 2008 } }).pretty()

//update multiple books
db.ProgrammingBooks.updateMany(
  { year: { $lt: 2010 } },
  { $set: { category: "Classic Programming Books" } }
)

db.ProgrammingBooks.find({ year: { $lt: 2010 } }).pretty()

//Delete a Single Document
db.ProgrammingBooks.deleteOne({ title: "JavaScript: The Good Parts" })

//delete multiple documents
db.ProgrammingBooks.deleteMany({ year: { $lt: 1995 } })

//delete multiple documents
db.ProgrammingBooks.deleteMany({})

//delete collection
db.ProgrammingBooks.drop()
```